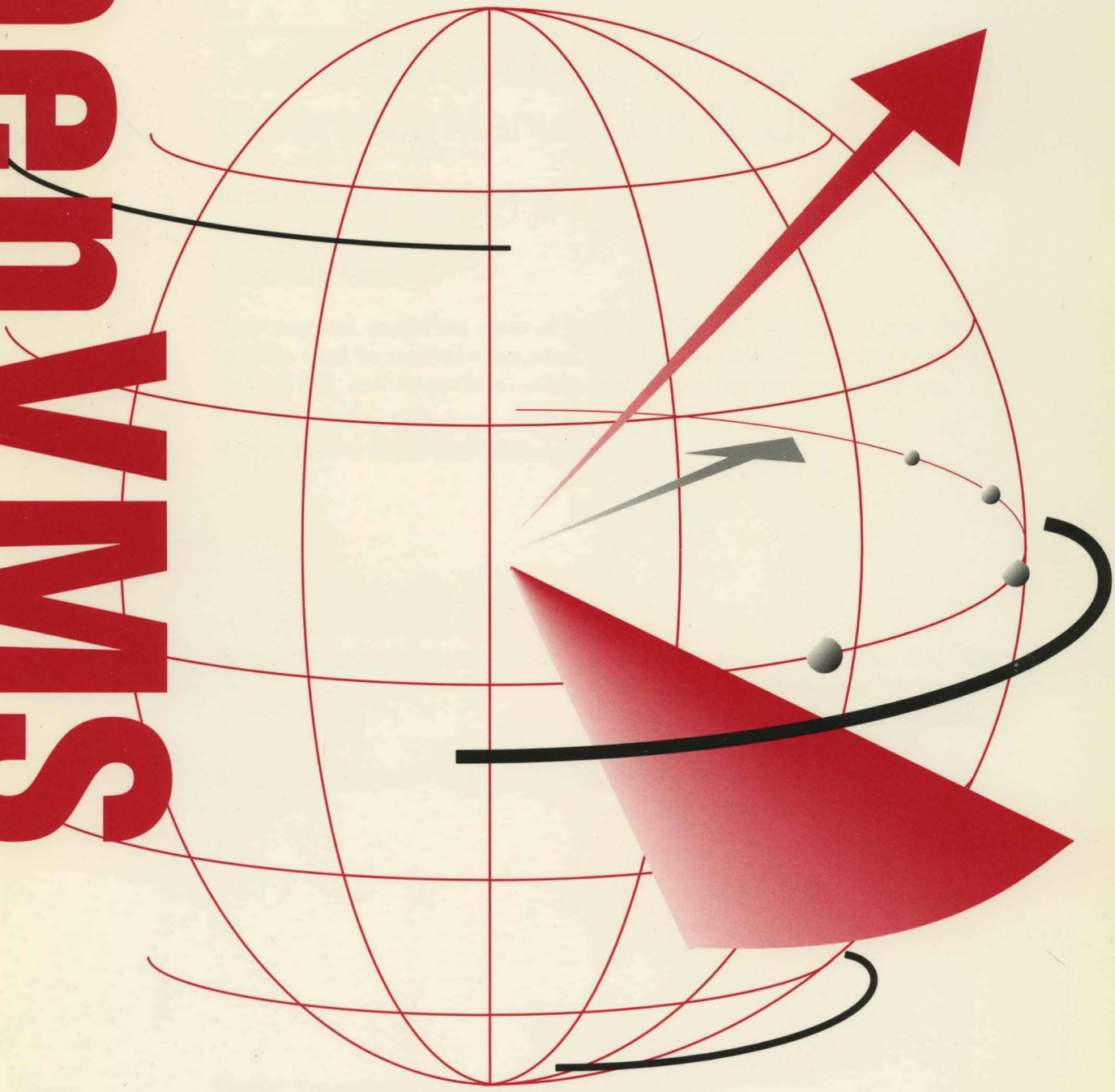
digital

OpenVMS System Services Reference Manual: GETQUI-Z

OpenVMS

OpenVMS



OpenVMS System Services Reference Manual: GETQUI–Z

Order Number: AA–QSBNA–TE

December 1995

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

This manual is in two parts. This second part contains the system services from $GETQUI through Z.

Revision/Update Information: This manual supersedes the *OpenVMS System Services Reference Manual* for OpenVMS AXP Version 6.1 and OpenVMS VAX Version 6.1.

Software Version: OpenVMS Alpha Version 7.0
OpenVMS VAX Version 7.0

Digital Equipment Corporation
Maynard, Massachusetts

December 1995

Digital Equipment Corporation makes no representations that the use of its products in the manner described in this publication will not infringe on existing or future patent rights, nor do the descriptions contained in this publication imply the granting of licenses to make, use, or sell equipment or software in accordance with the description.

Possession, use, or copying of the software described in this publication is authorized only pursuant to a valid written license from Digital or an authorized sublicensor.

Digital conducts its business in a manner that conserves the environment and protects the safety and health of its employees, customers, and the community.

© Digital Equipment Corporation 1995. All rights reserved.

The following are trademarks of Digital Equipment Corporation: AXP, Bookreader, DEC Fortran, DECdns, DECdtn, DECnet, DECnet/OSI, DECwindows, Digital, HSC, MASSBUS, MicroVAX, MicroVAX II, MSCP, OpenVMS, RA, StorageWorks, TA, TMSCP, TURBOchannel, ULTRIX, VAX, VAX C, VAX DOCUMENT, VAXcluster, VMS, VMScluster, VT, and the DIGITAL logo.

ZK6244

The following are third-party trademarks:

Oracle is a registered trademark, and Oracle CODASYL DBMS and Oracle Rdb are trademarks of Oracle Corporation.

OSI is a registered trademark of CA Management, Inc.

This document is available on CD–ROM.

Contents

Preface vii

System Service Descriptions

$GETQUI SYS2–3
$GETQUIW SYS2–46
$GET_REGION_INFO (Alpha Only) SYS2–47
$GETSYI SYS2–51
$GETSYIW SYS2–70
$GETTIM SYS2–71
$GETUAI SYS2–72
$GETUTC SYS2–84
$GET_ALIGN_FAULT_DATA (Alpha Only) SYS2–85
$GET_ARITH_EXCEPTION (Alpha Only) SYS2–87
$GET_SECURITY SYS2–89
$GET_SYS_ALIGN_FAULT_DATA (Alpha Only) SYS2–97
$GOTO_UNWIND (Alpha Only) SYS2–99
$GRANTID SYS2–101
$HASH_PASSWORD SYS2–105
$HIBER SYS2–108
$IDTOASC SYS2–110
$IEEE_SET_FP_CONTROL (Alpha Only) SYS2–113
$INIT_SYS_ALIGN_FAULT_REPORT (Alpha Only) SYS2–115
$INIT_VOL SYS2–118
$IO_CLEANUP (Alpha Only) SYS2–131
$IO_PERFORM (Alpha Only) SYS2–132
$IO_PERFORMW (Alpha Only) SYS2–135
$IO_SETUP (Alpha Only) SYS2–136
$LCKPAG SYS2–139
$LCKPAG_64 (Alpha Only) SYS2–142
$LKWSET SYS2–145
$LKWSET_64 (Alpha Only) SYS2–148
$MGBLSC SYS2–151
$MGBLSC_64 (Alpha Only) SYS2–157
$MGBLSC_GPFN_64 (Alpha Only) SYS2–163
$MOD_HOLDER SYS2–169
$MOD_IDENT SYS2–172
$MOUNT SYS2–176
$MTACCESS SYS2–191

$NUMTIM SYS2–194
$NUMUTC SYS2–196
$PARSE_ACL SYS2–198
$PERM_DIS_ALIGN_FAULT_REPORT (Alpha Only) SYS2–201
$PERM_REPORT_ALIGN_FAULT (Alpha Only) SYS2–202
$PROCESS_AFFINITY (Alpha Only) SYS2–204
$PROCESS_CAPABILITIES (Alpha Only) SYS2–209
$PROCESS_SCAN SYS2–214
$PURGWS SYS2–227
$PURGE_WS (Alpha Only) SYS2–229
$PUTMSG SYS2–231
$QIO SYS2–239
$QIOW SYS2–245
$READEF SYS2–246
$RELEASE_VP (VAX Only) SYS2–248
$REM_HOLDER SYS2–249
$REM_IDENT SYS2–251
$RESCHED SYS2–253
$RESTORE_VP_EXCEPTION (VAX Only) SYS2–254
$RESTORE_VP_STATE (VAX Only) SYS2–256
$RESUME SYS2–258
$REVOKID SYS2–260
$RMSRUNDWN SYS2–264
$SAVE_VP_EXCEPTION (VAX Only) SYS2–266
$SCAN_INTRUSION SYS2–268
$SCHDWK SYS2–273
$SCHED SYS2–276
$SETAST SYS2–281
$SETCLUEVT (Alpha Only) SYS2–282
$SETDDIR SYS2–285
$SETDFPROT SYS2–287
$SETEF SYS2–289
$SETEXV SYS2–290
$SETIME SYS2–292
$SETIMR SYS2–294
$SET_IMPLICIT_AFFINITY (Alpha Only) SYS2–297
$SETPRA SYS2–301
$SETPRI SYS2–303
$SETPRN SYS2–307
$SETPRT SYS2–308
$SETPRT_64 (Alpha Only) SYS2–311
$SETPRV SYS2–314
$SETRWM SYS2–319
$SETSHLV SYS2–321
$SETSTK SYS2–323
$SETSWM SYS2–325
$SETUAI SYS2–327

$SET_RESOURCE_DOMAIN SYS2–339
$SET_SECURITY SYS2–344
$SHOW_INTRUSION SYS2–351
$SIGNAL_ARRAY_64 ..... SYS2–356
$SNDERR SYS2–358
$SNDJBC SYS2–359
$SNDJBCW SYS2–417
$SNDOPR SYS2–418
$START_ALIGN_FAULT_REPORT (Alpha Only) SYS2–432
$START_TRANS SYS2–435
$START_TRANSW SYS2–439
$STOP_ALIGN_FAULT_REPORT (Alpha Only) SYS2–440
$STOP_SYS_ALIGN_FAULT_REPORT (Alpha Only) SYS2–441
$SUBSYSTEM SYS2–442
$SUSPND SYS2–444
$SYNCH SYS2–447
$TIMCON SYS2–449
$TRNLNM SYS2–451
$TSTCLUEVT (Alpha Only) SYS2–456
$ULKPAG SYS2–458
$ULKPAG_64 (Alpha Only) SYS2–460
$ULWSET SYS2–463
$ULWSET_64 (Alpha Only) SYS2–465
$UNWIND SYS2–468
$UPDSEC SYS2–470
$UPDSEC_64 (Alpha Only) SYS2–475
$UPDSECW SYS2–480
$UPDSEC_64W (Alpha Only) SYS2–481
$VERIFY_PROXY SYS2–482
$WAITFR SYS2–487
$WAKE SYS2–488
$WFLAND SYS2–490
$WFLOR SYS2–492

A Obsolete Services

Index

Tables

SYS2–1 Region Summary Buffer Format SYS2–49
SYS2–2 Format of the IEEE Floating-Point Control Register (Alpha Only) ... SYS2–113
SYS2–3 Flags Used with $PROCESS_SCAN SYS2–222
SYS2–4 User Privileges SYS2–314
SYS2–5 CPU Time Limit Decision Table SYS2–382
SYS2–6 Working Set Decision Table SYS2–405

Preface

Intended Audience

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

System Services Support for OpenVMS Alpha 64-bit Addressing

Alpha

As of Version 7.0, the OpenVMS Alpha operating system provides support for 64-bit virtual memory addresses, which makes the 64-bit virtual address space defined by the Alpha architecture available to the OpenVMS Alpha operating system and to application programs. In the 64-bit virtual address space, both process-private and system virtual address space extend beyond 2 GB. By using 64-bit address features, programmers can create images that map and access data beyond the previous limits of 32-bit virtual addresses.

New OpenVMS system services are available, and many existing services have been enhanced to manage 64-bit address space. The system services descriptions in this manual indicate the services that accept 64-bit addresses. A list of the OpenVMS system services that accept 64-bit addresses is available in the *OpenVMS Alpha Guide to 64-Bit Addressing*.

This section briefly describes how 64-bit addressing support affects OpenVMS system services. For complete information about OpenVMS Alpha 64-bit addressing features, see the *OpenVMS Alpha Guide to 64-Bit Addressing*.

64-Bit System Services Terminology

32-bit system service

A 32-bit system service is a system service that only supports 32-bit addresses on any of its arguments that specify addresses. If passed by value on OpenVMS Alpha, a 32-bit virtual address is actually a 64-bit address that is sign-extended from 32-bits.

64-bit friendly interface

A 64-bit friendly interface is an interface that can be called with all 64-bit addresses. A 32-bit system service interface is 64-bit friendly if, without a change in the interface, it needs no modification to handle 64-bit addresses. The internal code that implements the system service might need modification, but the system service interface will not.

64-bit system service

A 64-bit system service is a system service that is defined to accept all address arguments as 64-bit addresses (not necessarily 32-bit sign-extended values). Also, a 64-bit system service uses the entire 64 bits of all virtual addresses passed to it.

Use of the _64 Suffix

The 64-bit system services include the **_64** suffix for services that accept 64-bit addresses by reference. For promoted services, this distinguishes the 64-bit capable version from its 32-bit counterpart. For new services, it is a visible reminder that a 64-bit wide address cell will be read/written.

Sign-Extension Checking

OpenVMS system services that do not support 64-bit addresses and all user-written system services that are not explicitly enhanced to accept 64-bit addresses will receive sign-extension checking. Any argument passed to these services that is not properly sign-extended will cause the error status SS$_ARG_GTR_32_BITS to be returned.♦

Document Structure

The *OpenVMS System Services Reference Manual* is a two-part manual. The first part contains information on A through $GETMSG; the second part contains information on $GETQUI through Z.

Related Documents

The *OpenVMS Programming Interfaces: Calling a System Routine* manual contains useful information for anyone who wants to call system services.

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 OpenVMS language.

The following documents may also be useful:

- *OpenVMS Programming Concepts Manual*
- *Guide to OpenVMS File Applications*
- *OpenVMS Guide to System Security*
- *DECnet for OpenVMS Networking Manual*
- *OpenVMS Record Management Services Reference Manual*
- *OpenVMS I/O User's Reference Manual*
- *OpenVMS Alpha Guide to 64-Bit Addressing*
- *OpenVMS Alpha Guide to Upgrading Privileged-Code Applications*

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

How To Order Additional Documentation

Use the following table to order additional documentation or information. If you need help deciding which documentation best meets your needs, call 800-DIGITAL (800-344-4825).

Telephone and Direct Mail Orders

Location	Call	Fax	Write
U.S.A.	DECdirect 800–DIGITAL 800–344–4825	Fax: 800–234–2298	Digital Equipment Corporation P.O. Box CS2008 Nashua, NH 03061
Puerto Rico	809–781–0505	Fax: 809–749–8300	Digital Equipment Caribbean, Inc. 3 Digital Plaza, 1st Street, Suite 200 P.O. Box 11038 Metro Office Park San Juan, Puerto Rico 00910–2138
Canada	800–267–6215	Fax: 613–592–1946	Digital Equipment of Canada, Ltd. Box 13000 100 Herzberg Road Kanata, Ontario, Canada K2K 2A6 Attn: DECdirect Sales
International	—	—	Local Digital subsidiary or approved distributor
Internal Orders	DTN: 264–4446 603–884–4446	Fax: 603–884–3960	U.S. Software Supply Business Digital Equipment Corporation 10 Cotton Road Nashua, NH 03063–1260

ZK–7654A–GE

For additional information about OpenVMS products and services, access the Digital OpenVMS World Wide Web site. Use the following URL:

`http://www.openvms.digital.com`

Reader's Comments

Digital welcomes your comments on this manual.

Print or edit the online form SYS$HELP:OPENVMSDOC_COMMENTS.TXT and send us your comments by:

Internet	**openvmsdoc@zko.mts.dec.com**
Fax	603 881-0120, Attention: OpenVMS Documentation, ZK03-4/U08
Mail	OpenVMS Documentation Group, ZKO3-4/U08 110 Spit Brook Rd. Nashua, NH 03062-2698

Conventions

In this manual, every use of OpenVMS Alpha means the OpenVMS Alpha operating system, every use of OpenVMS VAX means the OpenVMS VAX operating system, and every use of OpenVMS means both the OpenVMS Alpha operating system and the OpenVMS VAX operating system.

The following conventions are used to identify information specific to OpenVMS Alpha or to OpenVMS VAX:

Alpha	The Alpha icon denotes the beginning of information specific to OpenVMS Alpha.

VAX	The VAX icon denotes the beginning of information specific to OpenVMS VAX.
♦	The diamond symbol denotes the end of a section of information specific to OpenVMS Alpha or to OpenVMS VAX.

In this manual, every use of DECwindows and DECwindows Motif refers to DECwindows Motif for OpenVMS software.

The following conventions are also used in this manual:

Ctrl/*x*	A sequence such as Ctrl/*x* indicates that you must hold down the key labeled Ctrl while you press another key or a pointing device button.
. . .	Horizontal ellipsis points in examples indicate one of the following possibilities: • Additional optional arguments in a statement have been omitted. • The preceding item or items can be repeated one or more times. • Additional parameters, values, or other information can be entered.
. . .	Vertical ellipsis points indicate the omission of items from a code example or command format; the items are omitted because they are not important to the topic being discussed.
( )	In command format descriptions, parentheses indicate that, if you choose more than one option, you must enclose the choices in parentheses.
[ ]	In command format descriptions, brackets indicate optional elements. You can choose one, none, or all of the options. (Brackets are not optional, however, in the syntax of a directory name in an OpenVMS file specification or in the syntax of a substring specification in an assignment statement.)
{ }	In command format descriptions, braces surround a required choice of options; you must choose one of the options listed.
boldface text	Boldface text represents the introduction of a new term or the name of an argument, an attribute, or a reason (user action that triggers a callback). Boldface text is also used to show user input in Bookreader versions of the manual.
italic text	Italic text emphasizes important information and indicates complete titles of manuals and variables. Variables include information that varies in system messages (Internal error *number*), in command lines (/PRODUCER=*name*), and in command parameters in text (where *device-name* contains up to five alphanumeric characters).
UPPERCASE TEXT	Uppercase text indicates a command, the name of a routine, the name of a file, or the abbreviation for a system privilege.
-	A hyphen in code examples indicates that additional arguments to the request are provided on the line that follows.

numbers	All numbers in text are assumed to be decimal unless otherwise noted. Nondecimal radixes—binary, octal, or hexadecimal—are explicitly indicated.

System Service Descriptions

System services provide basic operating system functions, interprocess communication, and various control resources.

Condition values returned by system services may provide information; that is, they do not indicate only whether the service completed successfully. The usual condition value indicating success is SS$_NORMAL, but others are defined. For example, the condition value SS$_BUFFEROVERF, which is returned when a character string returned by a service is longer than the buffer provided to receive it, is a success code. This condition value gives the program additional information.

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

The particular condition values that each service can return are described in the Condition Values Returned section of each individual service description.

Returns

OpenVMS 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.

$GETQUI
Get Queue Information

Returns information about queues and the jobs initiated from those queues.

The $GETQUI service completes asynchronously; for synchronous completion, use the Get Queue Information and Wait ($GETQUIW) service.

For additional information about system service completion, refer to the Synchronize ($SYNCH) service.

Format

SYS$GETQUI [efn] ,func [,context] [,itmlst] [,iosb] [,astadr] [,astprm]

Arguments

efn

OpenVMS 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

OpenVMS 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.

context

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

Address of a longword containing the number of a context stream for this call to the $GETQUI system service. If the argument is unspecified or 0, the service uses the default context stream (#0).

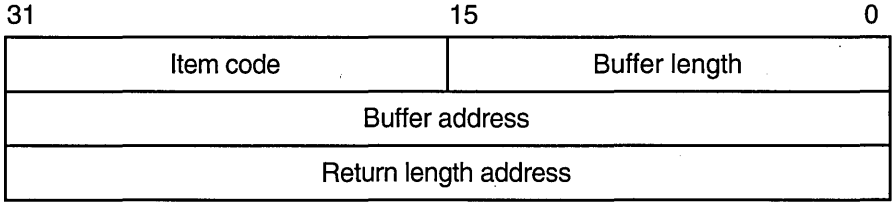
To generate a new context stream, the specified longword must contain –1. $GETQUI then modifies the longword to hold the context number for that stream of operation. The context is marked with the caller's mode (user, supervisor, executive, or kernel). Any attempt to use that context in successive calls is checked and no call from a mode outside the recorded mode is allowed access.

To clean up a context, make a $GETQUI call using the QUI$_CANCEL_OPERATION function code and specify the address of the context number as the **context** argument.

itmlst

OpenVMS 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–5186A–GE

The following table defines the item descriptor fields.

Descriptor Field	Definition
Buffer length	A word specifying the length of the buffer; the buffer either supplies information to $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.

Descriptor Field	Definition
Item code	A word containing an item code, which identifies the nature of the information supplied for $GETQUI or which is received from $GETQUI. Each item code has a symbolic name; the $QUIDEF macro defines these symbolic names.
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.

The item codes' symbolic names have the following format:

QUI$_code

There are two types of item code:

- **Input value item code.** The $GETQUI service has only five input value item codes: QUI$_SEARCH_FLAGS, QUI$_SEARCH_JOB_NAME, QUI$_SEARCH_NAME, QUI$_SEARCH_NUMBER, and QUI$_SEARCH_USERNAME. 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 are 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 can 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 $GETQUI return one of these names. For these item codes, the buffer must specify or be prepared to receive a string containing from 1 to 31 characters, exclusive of spaces, tabs, and null characters, which are ignored. Allowable characters in the string are uppercase alphabetic characters, lowercase alphabetic characters (which are converted to uppercase), numeric characters, the dollar sign ($), and the underscore (_).

See the Item Codes section for a description of the $GETQUI item codes.

iosb

OpenVMS 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 in the section Condition Values Returned in the I/O Status Block section.

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

OpenVMS usage:	ast_procedure
type:	procedure value
access:	call without stack unwinding
mechanism:	by reference

AST service routine to be executed when $GETQUI completes. The **astadr** argument is the address of this routine.

If specified, the AST routine executes at the same access mode as the caller of $GETQUI.

astprm

OpenVMS 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.

Function Codes

This section lists each of the $GETQUI function codes, describes the function, and lists the related item codes.

QUI$_CANCEL_OPERATION

This request terminates a wildcard operation that may have been initiated by a previous call to $GETQUI by releasing the $GETQUI context block (GQC) associated with the specified context stream.

A specific context stream can be selected and other streams are unaffected.

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 value item codes; you can specify both:

QUI$_SEARCH_NAME
QUI$_SEARCH_NUMBER

You can specify the following input value item code:

QUI$_SEARCH_FLAGS

You can specify the following output value 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 can specify the following input value item codes:

QUI$_SEARCH_FLAGS
QUI$_SEARCH_JOB_NAME

QUI$_SEARCH_NUMBER
QUI$_SEARCH_USERNAME

You can specify the following output value item codes:

QUI$_ACCOUNT_NAME
QUI$_AFTER_TIME
QUI$_ASSIGNED_QUEUE_NAME
QUI$_CHARACTERISTICS
QUI$_CHECKPOINT_DATA
QUI$_CLI
QUI$_COMPLETED_BLOCKS
QUI$_CONDITION_VECTOR
QUI$_CPU_LIMIT
QUI$_ENTRY_NUMBER
QUI$_FILE_COUNT
QUI$_FORM_NAME
QUI$_FORM_STOCK
QUI$_JOB_COMPLETION_QUEUE
QUI$_JOB_COMPLETION_TIME
QUI$_JOB_COPIES
QUI$_JOB_COPIES_DONE
QUI$_JOB_FLAGS
QUI$_JOB_NAME
QUI$_JOB_PID
QUI$_JOB_RETENTION_TIME
QUI$_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$_PROCESSOR
QUI$_QUEUE_FLAGS
QUI$_QUEUE_NAME
QUI$_QUEUE_STATUS
QUI$_REQUEUE_QUEUE_NAME
QUI$_RESTART_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 can specify the following input value item code:

QUI$_SEARCH_FLAGS

You can specify the following output value 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 value item codes. You can specify both:

QUI$_SEARCH_NAME
QUI$_SEARCH_NUMBER

You can specify the following input value item code:

QUI$_SEARCH_FLAGS

You can specify the following output value 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 can specify the following input value item code:

QUI$_SEARCH_FLAGS

You can specify the following output value 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$_ENTRY_NUMBER
QUI$_FILE_COUNT
QUI$_FORM_NAME
QUI$_FORM_STOCK
QUI$_INTERVENING_BLOCKS
QUI$_INTERVENING_JOBS

QUI$_JOB_COMPLETION_QUEUE
QUI$_JOB_COMPLETION_TIME
QUI$_JOB_COPIES
QUI$_JOB_COPIES_DONE
QUI$_JOB_FLAGS
QUI$_JOB_NAME
QUI$_JOB_PID
QUI$_JOB_RETENTION_TIME
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_MANAGER
This request returns information about a specific queue manager, or the next queue manager in a wildcard operation.

The $GETQUI service indicates that it has returned information about all the queue managers contained in the current wildcard sequence by returning the condition value JBC$_NOMOREQMGR in the I/O status block. If no queue manager matching the name string is found, $GETQUI returns the condition value JBC$_NOSUCHQMGR in the first longword of the I/O status block.

You must specify the following input value item code:

QUI$_SEARCH_NAME

You can specify the following input value item code:

QUI$_SEARCH_FLAGS

You can specify the following output value item codes:

QUI$_MANAGER_NAME
QUI$_MANAGER_NODES
QUI$_MANAGER_STATUS
QUI$_QUEUE_DIRECTORY
QUI$_SCSNODE_NAME

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 value item code:

QUI$_SEARCH_NAME

You can specify the following input value item code:

QUI$_SEARCH_FLAGS

You can specify the following output value 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 value item code:

QUI$_SEARCH_NAME

You can specify the following output value item code:

QUI$_QUEUE_NAME

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$_DISPLAY_ENRY, QUI$_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. However, if the time specified at submission has passed, the job executes immediately and $GETQQU returns the system time at which the job was submitted.

(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$_DISPLAY_ENTRY, QUI$_DISPLAY_QUEUE function codes)

QUI$_AUTOSTART_ON
When you specify QUI$_AUTOSTART_ON for a batch queue, $GETQUI returns, as a character string in a comma-separated list, the names of the nodes on which the specified autostart queue can be run. Each node name is followed by a double colon (::).

When you specify QUI$_AUTOSTART_ON for an output queue, $GETQUI returns, as a character string in a comma-separated list, the names of the nodes and devices to which the specified autostart queue's output can be sent. Each node name is followed by a double colon (::). Each device name may be followed by the optional colon [:].

For more information on the autostart feature, see the *OpenVMS System Manager's Manual.*

(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, $GETQUI returns, as an OpenVMS RMS file name component, the name of the command language interpreter used to execute the specified batch job. The file specification returned assumes the logical name SYS$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 *OpenVMS System Manager's Manual*.

(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 *OpenVMS System Manager's Manual*.

(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 *OpenVMS System Manager's Manual*.

(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$_FILE_COUNT
When you specify QUI$_FILE_COUNT, $GETQUI returns, as a longword integer value, the number of files in a specified job.

(Valid for QUI$_DISPLAY_ENTRY, QUI$_DISPLAY_JOB function codes)

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 OpenVMS 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 *OpenVMS System Manager's Manual.*

(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 *OpenVMS System Manager's Manual.*

(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 size (in blocks) of files associated with pending jobs in the queue that were skipped during the current call to $GETQUI. These jobs were not reported because they did not match the selection criterion in effect for the call to $GETQUI.

The value of QUI$_INTERVENING_BLOCKS is 0 when (1) the job is not a pending job, or (2) the job that matches the selection criterion is the first pending job in the queue, or (3) the preceding pending job in the queue was reported in the previous call to $GETQUI.

This item code applies only to output queues.

In a wildcard sequence of calls to $GETQUI using the QUI$_DISPLAY_JOB function code, only information about jobs that match the $GETQUI selection criteria is returned.

(Valid for QUI$_DISPLAY_JOB function code)

QUI$_INTERVENING_JOBS
When you specify QUI$_INTERVENING_JOBS, $GETQUI returns, as a longword integer value, the number of pending jobs in the queue that were skipped during the current call to $GETQUI. These jobs were not reported because they did not match the selection criterion in effect for the call to $GETQUI.

The value of QUI$_INTERVENING_JOBS is 0 when (1) the job is not a pending job, or (2) the job that matches the selection criterion is the first pending job in the queue, or (3) the preceding pending job in the queue was reported in the previous call to $GETQUI.

This item code applies only to output queues.

In a wildcard sequence of calls to $GETQUI using the QUI$_DISPLAY_JOB function code, only information about jobs that match the $GETQUI selection criteria is returned.

(Valid for QUI$_DISPLAY_JOB function code)

QUI$_JOB_COMPLETION_QUEUE
When you specify QUI$_JOB_COMPLETION_QUEUE, $GETQUI returns, as a character string, the name of the queue on which the specified job executed. Because a queue name can include up to 31 characters, the buffer length of the item descriptor should specify 31 (bytes).

This item code has a value only if the QUI$_JOB_RETAINED bit is set in the QUI$_JOB_STATUS longword item code.

(Valid for QUI$_DISPLAY_ENTRY, QUI$_DISPLAY_JOB function codes)

QUI$_JOB_COMPLETION_TIME
When you specify QUI$_JOB_COMPLETION_TIME, $GETQUI returns, as a quadword absolute time value, the system time at which the execution of the specified job completed.

This item code has a value only if the QUI$_JOB_RETAINED bit is set in the QUI$_JOB_STATUS longword item code.

(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_ERROR_RETENTION	The user requested that the job be retained in the queue, if the job completes unsuccessfully. If the queue is set to retain all jobs because the QUI$V_QUEUE_RETAIN_ALL bit of the QUI$_QUEUE_FLAGS item code is set, the job may be held in the queue even if it completes successfully. For more information about user-specified job retention, see the /RETAIN qualifier for the PRINT or SUBMIT command in the *OpenVMS DCL Dictionary*.
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.
QUI$V_JOB_LOWERCASE	Job is to be printed on printer that can print both uppercase and lowercase letters.

Symbolic Name	Description
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$V_JOB_RETENTION	The user requested that the job be retained in the queue regardless of the job's completion status. For more information about user-specified job retention, see the /RETAIN qualifier for the PRINT or SUBMIT command in the *OpenVMS DCL Dictionary*.
QUI$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$_JOB_RETENTION_TIME
When you specify QUI$_JOB_RETENTION_TIME, $GETQUI returns, as a quadword time value, the system time until which the user requested the job be retained in the queue. The system time may be expressed in either an absolute or delta time format.

For more information, see the /RETAIN qualifier for PRINT or SUBMIT in the *OpenVMS DCL Dictionary*.

(Valid for QUI$_DISPLAY_ENTRY, QUI$_DISPLAY_JOB function codes)

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.

Symbol 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$_ENTRY_NUMBER QUI$_INTEVENING_BLOCKS QUI$_INTEVENING_JOBS QUI$_JOB_SIZE QUI$_JOB_STATUS

Symbol Name	Description
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.
QUI$V_JOB_STALLED	Execution of the job is stalled because the physical device on which the job is printing is stalled.
QUI$V_JOB_STARTING	The job has been scheduled for execution. Confirmation of execution has not been received.
QUI$V_JOB_SUSPENDED	Execution of the job is suspended because the queue on which it is executing is paused.
QUI$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, $GETQUI returns, as an OpenVMS RMS file name component, the name of the device control library for the specified queue. The library specification assumes the device and directory name SYS$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 OpenVMS 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$_MANAGER_NAME
When you specify QUI$_MANAGER_NAME, $GETQUI returns, as a character string, the queue manager name. Because a queue manager name can include up to 31 characters, the buffer length field of the item descriptor should specify 31 (bytes).

(Valid for QUI$_DISPLAY_MANAGER function code)

QUI$_MANAGER_NODES
When you specify QUI$_MANAGER_NODES, $GETQUI returns, as a comma separated list, the names of the nodes on which this queue manager runs.

(Valid for QUI$_DISPLAY_MANAGER function code)

QUI$_MANAGER_STATUS
When you specify QUI$_MANAGER_STATUS, $GETQUI returns the specified queue manager'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$V_MANAGER_FAILOVER	Queue manager is in the process of failing over to another node.
QUI$V_MANAGER_RUNNING	Queue manager is running.
QUI$V_MANAGER_START_PENDING	Queue manager can start up whenever a node on which it can run is booted.
QUI$V_MANAGER_STARTING	Queue manager is in the process of starting up.
QUI$V_MANAGER_STOPPING	Queue manager is in the process of shutting down.
QUI$V_MANAGER_STOPPED	Queue manager is stopped.

(Valid for QUI$_DISPLAY_MANAGER function code)

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 for batch and 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.

(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 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$_PENDING_JOB_BLOCK_COUNT
When you specify QUI$_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.

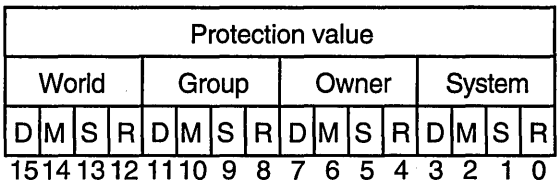
(Valid for QUI$_DISPLAY_ENTRY, QUI$_DISPLAY_JOB function codes)

QUI$_PROCESSOR
When you specify QUI$_PROCESSOR, $GETQUI returns, as an OpenVMS RMS file name component, the name of the symbiont image that executes print jobs initiated from the specified queue. The file name assumes the device and directory name SYS$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$_DISPLAY_ENTRY, QUI$_DISPLAY_QUEUE function codes)

QUI$_PROTECTION
When you specify QUI$_PROTECTION, $GETQUI returns, as a word, the specified queue's protection mask.

The following diagram illustrates the protection mask.



ZK-3823A-GE

Bits 0 through 15 specify the protection value—the four types of access (read, submit, manage, and delete) to be granted to the four classes of user (System, Owner, Group, World). Set bits deny access and clear bits allow access.

(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$_QUEUE_DIRECTORY
When you specify QUI$_QUEUE_DIRECTORY, $GETQUI returns a string containing the device and directory specification of the queue database directory for this queue manager.

(Valid for QUI$_DISPLAY_MANAGER 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_AUTOSTART	Queue is designated as an autostart queue.
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.

Symbolic Name	Description
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.
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.

Symbolic Name	Description
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$_DISPLAY_ENTRY, QUI$_DISPLAY_QUEUE function codes)

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. Some of these bits describe the queue's state, others provide additional status information. 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_AUTOSTART_INACTIVE	Autostart queue is stopped due to failure or manual intervention and needs to be manually started.
QUI$V_QUEUE_AVAILABLE[1]	Queue is processing work but is capable of processing additional work.
QUI$V_QUEUE_BUSY[1]	Queue cannot process additional jobs because of work in progress.
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_DISABLED[1]	Queue is not capable of being started or submitted to.
QUI$V_QUEUE_IDLE[1]	Queue contains no job requests capable of being processed.

[1]Bit describes the current state of the queue. Only one of these bits can be set at any time.

Symbolic Name	Description
QUI$V_QUEUE_LOWERCASE	Queue is associated with a printer that can print both uppercase and lowercase characters.
QUI$V_QUEUE_PAUSED[1]	Execution of all current jobs in the queue is temporarily halted.
QUI$V_QUEUE_PAUSING[1]	Queue is temporarily halting execution.
QUI$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[1]	Queue is restarting after pausing.
QUI$V_QUEUE_SERVER	Queue processing is directed to a server symbiont.
QUI$V_QUEUE_STALLED[1]	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$V_QUEUE_STARTING[1]	Queue is starting.
QUI$V_QUEUE_STOP_PENDING	Queue will be stopped when work currently in progress has completed.
QUI$V_QUEUE_STOPPED[1]	Queue is stopped.
QUI$V_QUEUE_STOPPING[1]	Queue is stopping.
QUI$V_QUEUE_UNAVAILABLE	Physical device to which queue is assigned is not available.

[1]Bit describes the current state of the queue. Only one of these bits can be set at any time.

(Valid for QUI$_DISPLAY_ENTRY, QUI$_DISPLAY_QUEUE function codes)

QUI$_REQUEUE_QUEUE_NAME
When you specify QUI$_REQUEUE_QUEUE_NAME, $GETQUI returns, as a character string, the name of the queue to which the specified job is reassigned. This item code only has a value if the QUI$V_JOB_ABORTING bit is set in the QUI$_JOB_STATUS longword, and the job is going to be requeued to another queue. Because a queue name can include up to 31 characters, the buffer length 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$_DISPLAY_ENTRY, QUI$_DISPLAY_JOB function codes)

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 node on which the specified execution queue or queue manager 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$_DISPLAY_QUEUE, QUI$_DISPLAY_MANAGER function codes)

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$V_SEARCH_ALL_JOBS	QUI$_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$_DISPLAY_JOB QUI$_DISPLAY_QUEUE	Selects executing jobs, or queues with executing jobs.
QUI$V_SEARCH_FREEZE_CONTEXT	QUI$_DISPLAY_CHARACTERISTIC QUI$_DISPLAY_ENTRY QUI$_DISPLAY_FILE QUI$_DISPLAY_FORM QUI$_DISPLAY_JOB QUI$_DISPLAY_QUEUE QUI$_DISPLAY_MANAGER	Does not advance wildcard context on completion of this service call.
QUI$V_SEARCH_GENERIC	QUI$_DISPLAY_ENTRY QUI$_DISPLAY_QUEUE	Selects generic queues.
QUI$V_SEARCH_HOLDING_JOBS	QUI$_DISPLAY_ENTRY QUI$_DISPLAY_JOB QUI$_DISPLAY_QUEUE	Selects jobs on unconditional hold, or queues with jobs on unconditional hold.
QUI$V_SEARCH_PENDING_JOBS	QUI$_DISPLAY_ENTRY QUI$_DISPLAY_JOB QUI$_DISPLAY_QUEUE	Selects pending jobs, or queues with pending jobs.
QUI$V_SEARCH_PRINTER	QUI$_DISPLAY_ENTRY QUI$_DISPLAY_QUEUE	Selects printer queues.
QUI$V_SEARCH_RETAINED_JOBS	QUI$_DISPLAY_ENTRY QUI$_DISPLAY_JOB QUI$_DISPLAY_QUEUE	Selects jobs being retained, or queues with jobs being retained.
QUI$V_SEARCH_SERVER	QUI$_DISPLAY_ENTRY QUI$_DISPLAY_QUEUE	Selects server queues.

Symbolic Name	Function Code	Description
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.
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$_DISPLAY_JOB QUI$_DISPLAY_QUEUE	Selects jobs on hold until a specified time, or queues with 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$_SEARCH_JOB_NAME
QUI$_SEARCH_JOB_NAME is an input value item code that specifies a 1- to 39-character string that $GETQUI uses to restrict its search for a job or jobs. $GETQUI searches for job names that match the job name input value for the given user name. Wildcard characters are acceptable.

(Valid for QUI$_DISPLAY_ENTRY function code)

QUI$_SEARCH_NAME
QUI$_SEARCH_NAME is an input value item code that 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$_DISPLAY_MANAGER, QUI$_DISPLAY_QUEUE, QUI$_TRANSLATE_QUEUE function codes)

QUI$_SEARCH_NUMBER
QUI$_SEARCH_NUMBER is an input value item code, that 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 that 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, in pages (on VAX systems) or pagelets (on Alpha systems), 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 queues.

(Valid for QUI$_DISPLAY_ENTRY, QUI$_DISPLAY_JOB, QUI$_DISPLAY_QUEUE function codes)

QUI$_WSEXTENT
When you specify QUI$_WSEXTENT, $GETQUI returns, in pages (on VAX systems) or pagelets (on Alpha systems), the working set extent 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 queues.

(Valid for QUI$_DISPLAY_ENTRY, QUI$_DISPLAY_JOB, QUI$_DISPLAY_QUEUE function codes)

QUI$_WSQUOTA
When you specify QUI$_WSQUOTA, $GETQUI returns, in pages (on VAX systems) or pagelets (on Alpha systems), 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 queues.

(Valid for QUI$_DISPLAY_ENTRY, QUI$_DISPLAY_JOB, QUI$_DISPLAY_QUEUE function codes)

Description

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 queue manager and job controller processes. See the Description section of the $SNDJBC service for a discussion of the different types of jobs and queues.

The $GETQUI service completes asynchronously; that is, it returns to the caller after queuing the request, without waiting for the operation to complete. For synchronous completion, use the Get Queue Information and Wait ($GETQUIW) service. The $GETQUIW service is identical to $GETQUI in every way except that $GETQUIW returns to the caller after the operation has completed.

You can 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_MANAGER	Queue manager
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 queue manager 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 any number of GQC blocks per process.

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 that you identify by name.
- A specific queue manager that you identify by name.
- A specific batch or print job that you identify by job entry number or by name.
- 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$_DISPLAY_FORM function code. You need to specify either the name of the characteristic or form in the QUI$_SEARCH_NAME item code or the number of the characteristic or form in the QUI$_SEARCH_NUMBER item code. The name string you specify cannot include either of the wildcard characters (* or %). You can specify both the QUI$_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 cannot include the wildcard characters (* or %).

To obtain information about a specific queue manager, specify the QUI$_DISPLAY_MANAGER function code and provide the name of the queue manager in the QUI$_SEARCH_NAME item code. The name string you specify cannot include the wildcard characters (* or %).

To obtain information about a specific batch or print job, 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, forms, queues, jobs, or queue managers contained in the queue database.

You can have several streams of operations open at one time. To use a stream, specify a unique longword value for the **context** argument for every call associated with that stream. If you do not specify the **context** argument, then context #0 will be used.

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 queues, use 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 queue managers, specify the QUI$_DISPLAY_MANAGER function code and specify a name in the QUI$_SEARCH_NAME item code that includes one or more wildcard characters (* or %).

To set up a wildcard search for jobs, 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$_SEARCH_NAME item code. Whether or not you specify a wildcard name in the QUI$_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 causes the GQC to be released:

- $GETQUI returns a JBC$_NOMORExxx or JBC$_NOSUCHxxx condition value on a call to display characteristic, form, queue, queue manager, or entry information, where *xxx* refers to CHAR, FORM, QUE, QMGR, or ENT.
- You explicitly cancel the wildcard operation by specifying the QUI$_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.

Note that the context number specified in the **context** argument must remain the same for each level of nesting.

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 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 you either change the context by making another call to $GETQUI that specifies the QUI$_DISPLAY_QUEUE function code or until 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$_SEARCH_FREEZE_CONTEXT option of the QUI$_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.

Required Access or Privileges

The caller must have manage (M) access to the queue, read (R) access to the job, or SYSPRV or OPER privilege to obtain job and file information.

If the caller does not have the privilege required 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

Required Quota

AST limit quota must be sufficient.

Related Services

$ALLOC, $ASSIGN, $BRKTHRU, $BRKTHRUW, $CANCEL, $CREMBX, $DALLOC, $DASSGN, $DELMBX, $DEVICE_SCAN, $DISMOU, $GETDVI, $GETDVIW, $GETMSG, $GETQUIW, $INIT_VOL, $MOUNT, $PUTMSG, $QIO, $QIOW, $SNDERR, $SNDJBC, $SNDJBCW, $SNDOPR, $TRNLNM

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$_BADCONTEXT	Context does not exist or must be called from a more privileged mode.
SS$_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$_MBFULL	The job controller mailbox is full.
SS$_IVLOGNAM	The device name string has a length of 0 or has more than 63 characters.
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$_NOMOREQMGR	No more queue managers are defined, which indicates the termination of a QUI$_DISPLAY_MANAGER wildcard operation.
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$_NOSUCHENT	There is no job with the specified entry number, or there is no job for the specified user or current user name.
JBC$_NOSUCHFILE	The specified file does not exist.
JBC$_NOSUCHFORM	The specified form does not exist.
JBC$_NOSUCHJOB	The specified job does not exist.
JBC$_NOSUCHQMGR	The specified queue manager does not exist.
JBC$_NOSUCHQUE	The specified queue does not exist.

Examples

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$M_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
```

This 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.

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
```

This 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.

$GETQUIW
Get Queue Information and Wait

Returns information about queues and jobs initiated from those queues. The $SNDJBC service is the major interface to the Job Controller, which is the queue and accounting manager. For a discussion of the different types of job and queue, see the Description section of $SNDJBC.

The $GETQUIW service completes synchronously; that is, it returns to the caller with the requested information. For asynchronous completion, use the Get Queue Information ($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 more information about $GETQUIW, refer to the description of $GETQUI in this manual.

For additional information about system service completion, refer to the Synchronize ($SYNCH) service.

Format

SYS$GETQUIW [efn] ,func [,context] [,itmlst] [,iosb] [,astadr] [,astprm]

$GET_REGION_INFO (Alpha Only)
Get Information About a Specified Virtual Region

On Alpha systems, gets information about a specified virtual region.

This service accepts 64-bit addresses.

Format

SYS$GET_REGION_INFO function_code ,region_id_64 ,start_va_64 ,buffer_length ,buffer_address_64 ,return_length_64

Arguments

function_code

OpenVMS usage:	function code
type:	longword (unsigned)
access:	read only
mechanism:	by value

Function code specifying how the information you are requesting should be looked up. All function codes return region summary information in the return buffer in the format of the Region Summary Buffer. The Region Summary Buffer format is shown in Table SYS2–1. If less buffer space is specified than the length of the Region Summary Buffer, only the amount of information requested is returned. If more buffer space is specified than the length of the Region Summary Buffer, the service will fill in the buffer. The return length will reflect the amount of useful information written to the buffer, the size of the Region Summary Buffer.

The file VADEF.H in SYS$STARLET_C.TLB and the $VADEF macro in STARLET.MLB define a symbolic name for each function code. The following function codes are defined:

Symbolic Name	Description
VA$_REGSUM_BY_ID	Return the region summary information for the region whose ID is specified in the **region_id_64** argument.
VA$_REGSUM_BY_VA	Return the region summary information for the region that contains the virtual address specified in the **start_va_64** argument.

Symbolic Name	Description
VA$_NEXT_REGSUM_BY_VA	Return the region summary information for the region containing the starting address. If the starting address is not in a region, return the region summary information for the next region with a starting address higher than the specified address. **Note:** For the VA$_NEXT_REGSUM_BY_VA function, OpenVMS checks for a **start_va_64** argument in the inaccessible address range in **P2** space. If it finds one, OpenVMS adjusts the address to account for the discontinuity. For more information about the layout of the 64-bit virtual address space, see the *OpenVMS Alpha Guide to 64-Bit Addressing*. This function code can be used for wildcard operations. See the description of the **start_va_64** argument for information on how to program a wildcard operation on regions.

region_id

OpenVMS usage:	region identifier
type:	quadword (unsigned)
access:	read only
mechanism:	by 32-bit or 64-bit reference

The region ID associated with the region about which information is requested. This argument is read only if the function code VA$_REGSUM_BY_ID is specified.

The file VADEF.H in SYS$STARLET_C.TLB and the $VADEF macro in STARLET.MLB define a symbolic name for each of the three default regions in P0, P1, and P2 space. The following region IDs are defined:

Symbol	Region
VA$C_P0	Program region
VA$C_P1	Control region
VA$C_P2	64-bit program region

Other region IDs, as returned by the $CREATE_REGION_64 service, can be specified.

start_va_64

OpenVMS usage:	input address
type:	quadword address
access:	read only
mechanism:	by value

Virtual address associated with region about which information is requested. This argument is read only if the **function_code** argument is VA$_REGSUM_BY_VA or VA$_NEXT_REGSUM_BY_VA.

If the **function_code** argument is VA$_REGSUM_BY_VA, this argument is a virtual address within the region about which you are requesting information.

To perform a wildcard search on all regions, specify VA$_NEXT_REGSUM_BY_VA as the function code and begin with the **start_va_64** argument specified as -1. For subsequent calls, specify **start_va_64** as the sum of the previous region's start address and length. Call the $GET_REGION_INFO service in a loop until the condition SS$_NOMOREREG is returned.

Note

Before performing the lookup function, OpenVMS sign-extends the 64-bit starting address so that it represents a properly formed virtual address for the CPU.

buffer_length
OpenVMS usage: longword_unsigned
type: longword (unsigned)
access: read only
mechanism: by value

Length of the buffer into which information is returned.

buffer_address_64
OpenVMS usage: varying_arg
type: unspecified
access: write only
mechanism: by 32-bit or 64-bit reference

The 32-bit or 64-bit virtual address of a quadword-aligned buffer into which to return information if the **buffer_length** argument is non-zero.

This argument is ignored if the **buffer_length** argument is zero.

Table SYS2–1 Region Summary Buffer Format

Field name	Meaning	Field Size (Bytes)	Field Offset (Decimal)
VA$L_FLAGS	Flags used when region was created	4	8
VA$L_REGION_PROTECT	Create and owner mode of region	4	12
VA$Q_REGION_ID	Region identifier	8	0
VA$PQ_START_VA	Starting (lowest) virtual address of region	8	16
VA$Q_REGION_SIZE	Total length of region	8	24
VA$PQ_FIRST_FREE_VA	First free virtual address in region	8	32
VA$C_REGSUM_LENGTH	Length of Region Summary Buffer	constant	40

The file VADEF.H in SYS$STARLET_C.TLB and the $VADEF MACRO in STARLET.MLB define the REGSUM structure.

return_length_64

OpenVMS usage:	longword_unsigned
type:	longword (unsigned)
access:	write only
mechanism:	by 32-bit or 64-bit reference

The 32-bit or 64-bit virtual address of a naturally aligned longword into which the service returns the length of the information in bytes.

Description

The Get Information About a Specified Virtual Region service is a kernel mode service that can be called from any mode. This service gets the requested information about the specified region or the next region in a wildcard search. If the returned value of this service is not a successful condition value, a value *cannot* be returned in the memory locations pointed to by the **buffer_address_64** or **return_length_64** arguments.

Required Privileges

None

Required Quota

None.

Related Services

$CREATE_REGION_64, $DELETE_REGION_64

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The **buffer_address_64** argument or the **return_length_64** argument cannot be written by the caller.
SS$_BADPARAM	Unrecognized function code.
SS$_IVREGID	Invalid region ID specified in conjunction with the VA$_REGSUM_BY_ID function code.
SS$_NOMOREREG	No region at a higher address than specified in the **start_va_64** argument, which was specified in conjunction with the wildcard function code VA$_NEXT_REGSUM_BY_VA.
SS$_PAGNOTINREG	The value specified in the **start_va_64** argument is not within a region and was specified in conjunction with the function code VA$_REGSUM_BY_VA.

$GETSYI
Get Systemwide Information

Returns information about the local system or about other systems in a VMScluster system. The $GETSYI service completes asynchronously; for synchronous completion, use the Get Systemwide Information and Wait ($GETSYIW) service.

For additional information about system service completion, refer to the Synchronize ($SYNCH) service.

Format

SYS$GETSYI [efn] ,[csidadr] ,[nodename] ,itmlst [,iosb] [,astadr] [,astprm]

Arguments

efn

OpenVMS 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

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

VMScluster system identification of the 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 VMScluster system identification of a node. You can obtain this information by using the DCL command SHOW CLUSTER. The value of the cluster system identification for a node is not permanent; a new value is assigned to a node whenever it joins or rejoins the cluster.

You can also specify a 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 node. If you specify neither argument, $GETSYI returns information about the local node. However, for wildcard operations, you must use the **csidadr** argument.

If you specify **csidadr** as –1, $GETSYI assumes a wildcard operation and returns the requested information for each node in the cluster, one node per call. In this case, the program should test for the condition value SS$_NOMORENODE

after each call to $GETSYI and should stop calling $GETSYI when SS$_NOMORENODE is returned.

nodename

OpenVMS usage: process_name
type: character-coded text string
access: read only
mechanism: by descriptor–fixed-length string descriptor

Name of the node about which $GETSYI is to return information. The **nodename** argument is the address of a character string descriptor pointing to this name string.

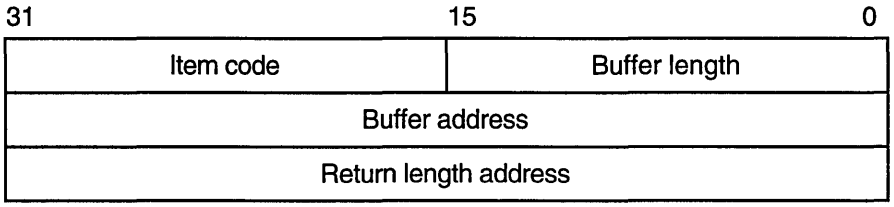
The node name string must contain from 1 to 15 characters and must correspond exactly to the node name; no trailing blanks or abbreviations are permitted.

You can also specify a node to $GETSYI by using the **csidadr** argument. See the description of **csidadr**.

itmlst

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

Item list specifying which information is to be returned about the 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–5186A–GE

The following table defines the item descriptor fields.

Descriptor Field	Definition
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 the buffer length field is too small, $GETSYI truncates the data.

Descriptor Field	Definition
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 Item Codes section.
Buffer address	A longword containing the user-supplied address of the buffer into 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.

See the Item Codes section for a description of the various $GETSYI item codes.

iosb

OpenVMS 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 0 upon request initiation. Upon request completion, a condition value is returned to the first longword; the second longword is reserved for future use.

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

OpenVMS usage: ast_procedure
type: procedure value
access: call without stack unwinding
mechanism: by reference

AST service routine to be executed when $GETSYI completes. The **astadr** argument is the address of this routine.

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

astprm

OpenVMS 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.

Item Codes

SYI$_ACTIVECPU_CNT
When you specify SYI$_ACTIVECPU_CNT, $GETSYI returns a count of the CPUs actively participating in the current boot of the symmetric multiprocessing (SMP) system. The $GETSYI service returns this information for the local node only.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

SYI$_ARCHFLAG
When you specify SYI$_ARCHFLAG, $GETSYI returns the architecture flags for the system.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

SYI$_ARCH_NAME
When you specify SYI$_ARCH_NAME, $GETSYI returns, as a character string, the name of the CPU architecture on which the process is executing. Currently, either of two strings is returned: "Alpha" for Alpha or "VAX" for VAX.

Because this name can include up to 15 characters, the buffer length field in the item descriptor should specify 15 (bytes).

SYI$_ARCH_TYPE
When you specify SYI$_ARCH_TYPE, $GETSYI returns the type of CPU architecture on which the process is executing. SYI$_ARCH_TYPE returns 1 on VAX or 2 on Alpha.

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. The $GETSYI service returns this information for the local node only.

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 node was booted. The $GETSYI service returns this information only for the local 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 the value 0 if they are not. The $GETSYI service returns this information only for the local 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 VMScluster system. The cluster determines this value by selecting the highest number from all of the following: each node's system parameter EXPECTED_VOTES, the sum of the votes currently in the cluster, 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 VMScluster system to boot.

The cluster management software assigns this system identification to the node. You can 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$_CLUSTER_FTIME
When you specify SYI$_CLUSTER_FTIME, $GETSYI returns the time when the founding node is booted. The founding node is the first node in the VMScluster system 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 VMScluster system. 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 VMScluster system.

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 VMScluster system. Each node's quorum value is derived from its system 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 VMScluster system. The number of votes held by any one node is determined by that node's system 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 CPU-specific 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).

VAX

SYI$_CPU

On VAX systems, when you specify SYI$_CPU, $GETSYI returns the CPU processor type, as represented in the processor's system identification (SID) register.

For example, the integer 1 represents a VAX–11/780 system and the integer 6 represents a VAX 8530, VAX 8550, VAX 8700, or VAX 8800 system.

The $GETSYI service returns this information only for the local 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$_SID_TYP730
VAX–11/750	PR$_SID_TYP750
VAX–11/780, 785	PR$_SID_TYP780
VAXstation II, II/GPX, and MicroVAX II	PR$_SID_TYPUV2
VAXstation 2000/MicroVAX 2000	PR$_SID_TYP410
VAX 8200, 8250, 8300, 8350	PR$_SID_TYP8SS
VAX 8530, 8550, 8810 (8700), and 8820-N (8800)	PR$_SID_TYP8NN
VAX 8600, 8650	PR$_SID_TYP790
VAX 8820, 8830, 8840	PR$_SID_TYP8PS
VAXft 3000 Model 310	PR$_SID_TYP520
VAXstation, MicroVAX 3100 series	PR$_SID_TYP420

Processor	Symbol
MicroVAX 3300, 3400, 3500, 3600, 3800, 3900	PR$_SID_TYP650
VAXstation 3520, 3540	PR$_SID_TYP60
VAX 4000-300	PR$_SID_TYP670
VAX 6000-200, 6000-300 series	PR$_SID_TYP9CC
VAX 6000-400 series	PR$_SID_TYP9RR
VAX 9000-200, 9000-400 series	PR$_SID_TYP9AQ♦

Alpha

On Alpha systems, when you specify SYI$_CPU, $GETSYI returns PR$_SID_TYP_NOTAVAX.♦

For information about extended processor type codes, see the description for the SYI$_XCPU item code.

Alpha

SYI$_CPUTYPE
On Alpha systems, when you specify SYI$_CPUTYPE, $GETSYI returns the processor type, as stored in the hardware restart parameter block (HWRPB). The value of 2 represents a DECchip 21064 processor.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).♦

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 the value 0 if they are not. The $GETSYI service returns this information only for the local node.

Because this number is a Boolean value (1 or 0), the buffer length field in the item descriptor should specify 1 (byte).

SYI$_DECNET_FULLNAME
When you specify SYI$_DECNET_FULLNAME, $GETSYI returns, as a character string, the DECnet for OpenVMS full name of the node.

Because the DECnet for OpenVMS full name of a node can contain up to 255 characters, the buffer length field in the item descriptor should specify 255 (bytes).

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 node.

Because this number is a Boolean value (1 or 0), the buffer length field in the item descriptor should specify 1 (byte).

Alpha

SYI$_DEF_PRIO_MAX
On Alpha systems, when you specify SYI$_DEF_PRIO_MAX, $GETSYI returns the maximum priority for the default scheduling policy.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).♦

Alpha

SYI$_DEF_PRIO_MIN
On Alpha systems, when you specify SYI$_DEF_PRIO_MIN, $GETSYI returns the minimum priority for the default scheduling policy.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).♦

SYI$_ERLBUFFERPAGES
When you specify SYI$_ERLBUFFERPAGES, $GETSYI returns the number of pages (on VAX systems) or pagelets (on Alpha systems) in an error log buffer.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

SYI$_ERRORLOGBUFFERS
When you specify SYI$_ERRORLOGBUFFERS, $GETSYI returns the number of system pages (on VAX systems) or pagelets (on Alpha systems) 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 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 system 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 system 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 the value 0 if they are not. The $GETSYI service returns this information only for the local node.

Because this number is a Boolean value (1 or 0), the buffer length field in the item descriptor should specify 1 (byte).

Alpha

SYI$_GH_RSRVPGCNT
On Alpha systems, when you specify SYI$_GH_RSRVPGCNT, $GETSYI returns the number of pages covered by granularity hints to reserve for use by the Install utility after system startup has completed.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).♦

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 the value 0 if they are not. The $GETSYI service returns this information only for the local 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 model type of the node.

Alpha

An integer greater than 1023 indicates an Alpha node.♦

VAX

An integer less than or equal to 1023 indicates a VAX node.♦

The $ALPHADEF and $VAXDEF macros in SYS$LIBRARY:STARLET define the model type integers. See the tables under the SYI$_HW_NAME item code for the VAX and Alpha model processor names and the corresponding model types.

Because SYI$_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 or Alpha model name string of the node. The model name is a character string that describes the model of the node (such as VAX 8800, MicroVAX II). The model name usually corresponds to the nameplate that appears on the outside of the CPU cabinet.

Because SYI$_HW_NAME can include up to 31 characters, the buffer length field in the item descriptor should specify 31 (bytes).

Alpha

The following table lists the Alpha model processor names and the corresponding model types.

Alpha Model Processor Name	Alpha Model Type
DEC 3000 400	ALPHA$K_A3000_400W
DEC 3000 400S	ALPHA$K_A3000_400S
DEC 3000 500	ALPHA$K_A3000_500W
DEC 3000 500S	ALPHA$K_A3000_500S
DEC 4000 610	ALPHA$K_A4000_610
DEC 4000 620	ALPHA$K_A4000_620
DEC 4000 630	ALPHA$K_A4000_630
DEC 4000 640	ALPHA$K_A4000_640
DEC 7000 Model 610	ALPHA$K_A7000_610
DEC 7000 Model 620	ALPHA$K_A7000_620

Alpha Model Processor Name	Alpha Model Type
DEC 7000 Model 630	ALPHA$K_A7000_630
DEC 7000 Model 640	ALPHA$K_A7000_640
DEC 10000 Model 610	ALPHA$K_A10000_610
DEC 10000 Model 620	ALPHA$K_A10000_620
DEC 10000 Model 630	ALPHA$K_A10000_630
DEC 10000 Model 640	ALPHA$K_A10000_640♦

VAX

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 II	VAX$K_VUV2
VAXstation II	VAX$K_VWS2
VAXstation II/GPX	VAX$K_VWSD
VAXstation 2000	VAX$K_VWS2000
MicroVAX 2000	VAX$K_VUV2000
VAXstation 2000/GPX	VAX$K_VWSD2000
VAX 8200	VAX$K_V8200
VAX 8250	VAX$K_V8250
VAX 8300	VAX$K_V8300
VAX 8350	VAX$K_V8350
VAX 8530	VAX$K_V8500
VAX 8550	VAX$K_V8550
VAX 8600	VAX$K_V8600
VAX 8650	VAX$K_V8650
VAX 8810 (8700)	VAX$K_V8700
VAX 8820-N (8800)	VAX$K_V8800
VAX 8820, 8830, or 8840 with one CPU enabled	VAX$K_V8810
VAX 8820	VAX$K_V8820
VAX 8830	VAX$K_V8830
VAX 8840	VAX$K_V8840
VAXft 3000 Model 310	VAX$K_V520FT
VAXstation 3520	VAX$K_V3520L
VAXstation 3540	VAX$K_V3540L
VAX 4000-300 timeshare	VAX$K_V670

VAX Model Processor Name	VAX Model Type
VAX 4000-300 server	VAX$K_V670_S
VAX 6000-210 timeshare	VAX$K_V6210_T
VAX 6000-220 timeshare	VAX$K_V6220_T
VAX 6000-230 timeshare	VAX$K_V6230_T
VAX 6000-240 timeshare	VAX$K_V6240_T
VAX 6000-250 timeshare	VAX$K_V6250_T
VAX 6000-260 timeshare	VAX$K_V6260_T
VAX 6000-210 server	VAX$K_V6210_S
VAX 6000-220 server	VAX$K_V6220_S
VAX 6000-310 timeshare	VAX$K_V6310_T
VAX 6000-320 timeshare	VAX$K_V6320_T
VAX 6000-330 timeshare	VAX$K_V6330_T
VAX 6000-340 timeshare	VAX$K_V6340_T
VAX 6000-350 timeshare	VAX$K_V6350_T
VAX 6000-360 timeshare	VAX$K_V6360_T
VAX 6000-310 server	VAX$K_V6310_S
VAX 6000-320 server	VAX$K_V6320_S
VAX 6000-410 timeshare	VAX$K_V9RR10_T
VAX 6000-420 timeshare	VAX$K_V9RR20_T
VAX 6000-430 timeshare	VAX$K_V9RR30_T
VAX 6000-440 timeshare	VAX$K_V9RR40_T
VAX 6000-450 timeshare	VAX$K_V9RR50_T
VAX 6000-460 timeshare	VAX$K_V9RR60_T
VAX 6000-410 server	VAX$K_V9RR10_S
VAX 6000-420 server	VAX$K_V9RR20_S
VAX 9000-210	VAX$K_V9AR10
VAX 9000-410	VAX$K_V9AQ10
VAX 9000-420	VAX$K_V9AQ20
VAX 9000-430	VAX$K_V9AQ30
VAX 9000-440	VAX$K_V9AQ40♦

Alpha

SYI$_ITB_ENTRIES
On Alpha systems, when you specify SYI$_ITB_ENTRIES, $GETSYI returns the number of instruction stream translation buffer entries that support granularity hints to be allocated for resident code.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).♦

SYI$_MEMSIZE
When you specify SYI$_MEMSIZE, $GETSYI returns the total number of pages of physical memory in the system configuration.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

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 VMScluster system ID (CSID) of the 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 VMScluster system. This number is determined by the system 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 CPU type. The $VAXDEF and $ALPHADEF macros define the CPU types.

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 for OpenVMS number of the node.

Because the DECnet for OpenVMS 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 system 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 VAX system, an Alpha 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 VMScluster management software assigns this system identification to the node. You can 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 system 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 buffer specified in the item list.

Because this name can include 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 page files. The $GETSYI service returns this information only for the local 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 page files. The $GETSYI service returns this information only for the local node.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

SYI$_PAGE_SIZE
When you specify SYI$_PAGE_SIZE, $GETSYI returns the number of CPU-specific bytes per page in the system.

VAX

On VAX systems, when you specify SYI$_PAGE_SIZE, $GETSYI always returns 512.♦

Alpha

On Alpha systems, CPU page size varies from system to system.♦

On Alpha and VAX systems, because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

Alpha

SYI$_PROCESS_SPACE_LIMIT
On Alpha systems, this item code returns the 64-bit virtual address succeeding the last available process private address. The value returned is the upper bound on the process private address space. The value returned is the same for every process on the system.

Because this number is a quadword, the buffer length field in the item descriptor should specify 8 (bytes).♦

Alpha

SYI$_PSXFIFO_PRIO_MAX
On Alpha systems, when you specify SYI$_PSXFIFO_PRIO_MAX, $GETSYI returns the maximum priority for the POSIX FIFO scheduling policy.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).♦

Alpha

SYI$_PSXFIFO_PRIO_MIN
On Alpha systems, when you specify SYI$_PSXFIFO_PRIO_MIN, $GETSYI returns the minimum priority for the POSIX FIFO scheduling policy.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).♦

Alpha

SYI$_PSXRR_PRIO_MAX
On Alpha systems, when you specify SYI$_PSXRR_PRIO_MAX, $GETSYI returns the maximum priority for the POSIX round-robin scheduling policy.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).♦

Alpha

SYI$_PSXRR_PRIO_MIN
On Alpha systems, when you specify SYI$_PSXRR_PRIO_MIN, $GETSYI returns the minimum priority for the POSIX round-robin scheduling policy.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).♦

Alpha

SYI$_PT_BASE
On Alpha systems, when you specify SYI$_PT_BASE, $GETSYI returns the 64-bit virtual address of the base of the page tables. The value returned is the same for every process on the system.

Because this number is a quadword, the buffer length field in the item descriptor should specify 8 (bytes).♦

SYI$_REAL_CPUTYPE
When you specify SYI$_REAL_CPUTYPE, $GETSYI returns the actual CPU type of the primary CPU of the system.

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 node; if the value is 0, the SCS is not currently loaded.

Alpha

SYI$_SHARED_VA_PTES
On Alpha systems, when you specify SYI$_SHARED_VA_PTES, $GETSYI returns the 64-bit virtual address of the PTE that marks the boundary between process-private PTEs and system-shared PTEs. The value returned is the same for every process on the system.

Because this number is a quadword, the buffer length field in the item descriptor should specify 8 (bytes).♦

SYI$_SID
When you specify SYI$_SID, $GETSYI returns the contents of the system identification register of the node. The $GETSYI service returns this information only for the local node.

Alpha

On Alpha systems, SYI$_SID returns a value in which all fields are 0 except the CPU-type field, which always contains the value 256.♦

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 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 node.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

SYI$_SYSTEM_RIGHTS
When you specify SYI$_SYSTEM_RIGHTS, $GETSYI returns the system rights list as an array of quadword identifiers. Each entry consists of a longword identifier value and the following longword identifier attributes.

Bit Position	Meaning When Set
KGB$V_DYNAMIC	Allows holders of the identifier to remove it from or add it to the process rights list using the DCL command SET RIGHTS_LIST.
KGB$V_NOACCESS	Makes any access rights of the identifier null and void. This attribute is intended as a modifier for a resource identifier or the Subsystem attribute.
KGB$V_RESOURCE	Allows holders of an identifier to charge disk space to the identifier. It is used only for file objects.

Bit Position	Meaning When Set
KGB$V_SUBSYSTEM	Allows holders of the identifier to create and maintain protected subsystems by assigning the Subsystem ACE to the application images in the subsystem.

Allocate a buffer that is sufficient to hold the system rights list, because $GETSYI returns only as much of the list as will fit in the buffer.

Alpha

SYI$_SYSTYP
On Alpha systems, when you specify SYI$_SYSTYP, $GETSYI returns the name of the family or system hardware platform. For example, the integer 2 represents a DEC 4000 processor, the integer 3 represents a DEC 7000 or DEC 10000 processor, and the integer 4 represents a DEC 3000 processor.♦

SYI$_VERSION
When you specify SYI$_VERSION, $GETSYI returns, as a character string, the software version number of the OpenVMS operating system running on the node. The $GETSYI service returns this information only for the local node.

Because the version number is 8-byte blank-filled, the buffer length field in the item descriptor should specify 8 (bytes).

SYI$_VECTOR_EMULATOR
When you specify SYI$_VECTOR_EMULATOR, $GETSYI returns a byte, the low-order bit of which, when set, indicates the presence of the Vector Instruction Emulator facility (VVIEF) in the system.

SYI$_VP_MASK
When you specify SYI$_VP_MASK, $GETSYI returns a longword mask, the bits of which, when set, indicate which processors in the system have vector coprocessors.

SYI$_VP_NUMBER
When you specify SYI$_VP_NUMBER, $GETSYI returns an unsigned longword containing the number of vector processors in the system.

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 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).

VAX

On VAX systems, 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 3300, 3400, 3500, 3600, 3800, 3900 series	PR$_XSID_CV_650
	VAX 6000-200, 6000-300 series	PR$_XSID_CV_9CC
	VAXstation 3520, 3540	PR$_XSID_CV_60
	VAXstation 3100 series	PR$_XSID_CV_420
	VAXft 3000 Model 310	PR$_XSID_CV_520
PR$_SID_TYP8NN	VAX 8530	PRS$_XSID_N8500
	VAX 8550	PRS$_XSID_N8550
	VAX 8810 (8700)	PRS$_XSID_N8700
	VAX 8820-N (8800)	PRS$_XSID_N8800
PR$_SID_TYPRV	VAX 4000-300	PR$_XSID_RV_670
	VAX 6000-400 series	PR$_XSID_RV_9RR♦

SYI$_XSID
When you specify SYI$_XSID, $GETSYI returns processor-specific information. For the MicroVAX II system, this information is the contents of the system type register of the node. The system type register contains the full extended information used in determining the extended system type codes. For other processors, the data returned by SYI$_XSID is 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 system parameter named *xxxx* for the node. The $GETSYI service returns this information only for the local node.

The buffer must specify a longword into which $GETSYI writes the value of the specified system parameter. For a list and description of all system parameters, refer to the *OpenVMS System Manager's Manual*.

Description

The Get Systemwide Information service returns information about the local system or about other systems in a VMScluster.

Required Access or Privileges

None

Required Quota

This service uses the process's AST limit quota (ASTLM).

Related Services

$ALLOC, $ASSIGN, $BRKTHRU, $BRKTHRUW, $CANCEL, $CREMBX, $DALLOC, $DASSGN, $DELMBX, $DEVICE_SCAN, $DISMOU, $GETDVI, $GETDVIW, $GETMSG, $GETQUI, $GETQUIW, $INIT_VOL, $MOUNT, $PUTMSG, $QIO, $QIOW, $SNDERR, $SNDJBC, $SNDJBCW, $SNDOPR

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_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$_BADPARAM	The item list contains an invalid item code.
SS$_EXASTLM	The process has exceeded its AST limit quota.
SS$_NOMORENODE	You requested a wildcard operation, and $GETSYI has returned information about all available nodes.
SS$_NOSUCHNODE	The specified node does not exist or is not currently a member of the VMScluster system.

Condition Values Returned in the I/O Status Block

Same as those returned in R0.

Example

```
! 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
```

This Fortran program demonstrates how to use the $GETSYIW service to obtain the operating system version number string and the system's node name.

$GETSYIW
Get Systemwide Information and Wait

Returns information about the local system or about other systems in a cluster.

The $GETSYIW service completes synchronously; that is, it returns to the caller with the requested information. For asynchronous completion, use the Get Systemwide Information ($GETSYI) service; $GETSYI returns to the caller after queuing the information request, without waiting for the information to be returned. In all other respects, these services are identical; refer to the documentation about $GETSYI for information about the $GETSYIW service.

For additional information about system service completion, refer to the Synchronize ($SYNCH) service.

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

Returns the current system time in a 64-bit format.

On Alpha systems, this service accepts 64-bit addresses.

Format

SYS$GETTIM timadr

Argument

timadr

OpenVMS usage:	date_time
type:	quadword (unsigned)
access:	write only
mechanism:	by 32-bit or 64-bit reference (Alpha) by 32-bit reference (VAX)

The 32-bit or 64-bit address (on Alpha systems) or the 32-bit address (on VAX systems) of a quadword to receive the current time in 64-bit format.

Description

The Get Time service returns the current system time in 64-bit format. The time is returned in 100-nanosecond units from the system base time.

Alpha

On Alpha systems, the frequency at which system time is updated varies, depending on the clock frequency of the Alpha processor.♦

VAX

On VAX systems, system time is updated every 10 milliseconds.♦

Required Access or Privileges

None

Required Quota

None

Related Services

$ASCTIM, $BINTIM, $CANTIM, $CANWAK, $NUMTIM, $SCHDWK, $SETIME, $SETIMR

For additional information about the system time, see the *OpenVMS System Manager's Manual*.

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

Returns authorization information about a specified user.

Format

SYS$GETUAI [nullarg] ,[contxt] ,usrnam ,itmlst ,[nullarg] ,[nullarg] ,[nullarg]

Arguments

nullarg

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

Placeholding argument reserved to Digital.

usrnam

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

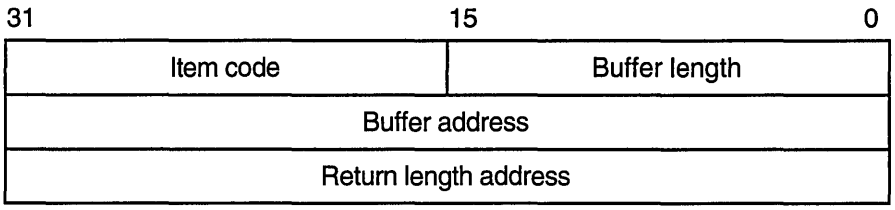
Name of the user about whom $GETUAI returns authorization information. The **usrnam** argument is the address of a descriptor pointing to a character text string containing the user name. The user name string can contain a maximum of 12 alphanumeric characters.

itmlst

OpenVMS 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 value of 0 or by a longword value of 0.

The following diagram depicts the structure of a single item descriptor.



ZK-5186A-GE

The following table defines the item descriptor fields.

Descriptor Field	Definition
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 the buffer length field 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.
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.

The symbolic codes have the following format:

$UAI_code

See the Item Codes section for descriptions of the various $GETUAI item codes.

contxt

OpenVMS usage: longword
type: longword (unsigned)
access: modify
mechanism: by reference

Longword used to maintain authorization file context. The **contxt** argument is the address of a longword to receive a $GETUAI context value. On the initial call, this longword should contain the value –1. On subsequent calls, the value of the **contxt** argument from the previous call should be passed back in.

Item Codes

UAI$_ACCOUNT
When you specify UAI$_ACCOUNT, $GETUAI returns, as a blank-filled 32-character string, the account name of the user.

An account name can include up to 8 characters. Because the account name is a blank-filled string, however, the buffer length field of the item descriptor should specify 32 (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 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, $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 in addition to 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 in addition to 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 in addition to 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 in addition to 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 in pages (on VAX systems) or pagelets (on Alpha systems).

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. For each hour the bit is set to 0, access is allowed. For each hour the bit is set to 1, access is denied.

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. For each hour the bit is set to 0, access is allowed. For each hour the bit is set to 1, access is denied.

The buffer length field in the item descriptor should specify 3 (bytes).

UAI$_ENCRYPT
When you specify UAI$_ENCRYPT, $GETUAI returns one of the values shown in the following table, identifying 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).

Symbolic Name	Description
UAI$C_AD_II	Uses a CRC algorithm and returns a longword hash value. It was used in VAX/VMS releases prior to Version 2.0.
UAI$C_PURDY	Uses a Purdy algorithm over salted input. It expects a blank-padded user name and returns a quadword hash value. This algorithm was used during VAX/VMS Version 2.0 field test.
UAI$C_PURDY_V	Uses the Purdy algorithm over salted input. It expects a variable-length user name and returns a quadword hash value. This algorithm was used in VMS releases prior to Version 5.4.
UAI$C_PURDY_S	Uses the Purdy algorithm over salted input. It expects a variable-length user name and returns a quadword hash value. This is the current algorithm that the operating system uses for all new password changes.

UAI$_ENCRYPT2
When you specify UAI$_ENCRYPT2, $GETUAI returns one of the following values identifying the encryption algorithm for the secondary password:

- UAI$C_AD_II
- UAI$C_PURDY
- UAI$C_PURDY_V
- UAI$C_PURDY_S

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$V_DISFORCE_PWD_CHANGE	User will not be forced to change expired passwords at login.
UAI$V_DISIMAGE	User cannot issue the RUN or MCR commands or use the foreign command mechanism in DCL.
UAI$V_DISMAIL	Announcement of new mail is suppressed.
UAI$V_DISPWDDIC	Automatic checking of user-selected passwords against the system dictionary is disabled.
UAI$V_DISPWDHIS	Automatic checking of user-selected passwords against previously used passwords is disabled.
UAI$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$V_GENPWD	User is required to use generated passwords.
UAI$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$V_PWD2_EXPIRED	Secondary password is expired.
UAI$V_RESTRICTED	User is limited to operating under a restricted account. (See the *Security Guide* for a description of restricted and captive accounts.)

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 OpenVMS RMS file specification, the name of the default login command file.

Because a file specification can include up to 63 characters in addition to 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. For each hour the bit is set to 0, access is allowed. For each hour the bit is set to 1, access is denied.

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. For each hour the bit is set to 0, access is allowed. For each hour the bit is set to 1, access is denied.

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 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, $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. For each hour the bit is set to 0, access is allowed. For each hour the bit is set to 1, access is denied.

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. For each hour the bit is set to 0, access is allowed. For each hour the bit is set to 1, access is denied.

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 in addition to 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 in pages (on VAX systems) or in blocks (on Alpha systems).

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).

A value of –1 indicates that the password is marked as preexpired.

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 delta 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).

A quadword of 0 means that none of the password mechanisms will take effect.

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 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).

A value of –1 indicates that the password could be marked as preexpired.

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). For the format of the UIC, see the *Security Guide*.

UAI$_USER_DATA
When you specify UAI$_USER_DATA, $GETUAI returns up to 255 bytes of information from the user data area of the system user authorization file (SYSUAF).

You can read information written to the user data area from previous versions of the operating system as long as the information written adheres to the guidelines described in the *Security Guide*.

UAI$_WSEXTENT

When you specify UAI$_WSEXTENT, $GETUAI returns the working set extent, in pages (on VAX systems) or pagelets (on Alpha systems), 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, in pages (on VAX systems) or pagelets (on Alpha systems), 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

The Get User Authorization Information service returns authorization information about a specified user.

Required Access or Privileges

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.

  You need read access to the UAF to look up any information other than your own.

Required Quota

None

Related Services

$SETUAI

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$_NOGRPPRV	The user does not have the privileges required to examine the authorization information for other members of the UIC group.
SS$_NOSYSPRV	The user does not have the privileges required to examine the authorization information associated with the user or for users outside of the user's UIC group.
RMS$_RSZ	The UAF record is smaller than required; the caller's SYSUAF is probably corrupt.

This service can also return OpenVMS RMS status codes associated with operations on indexed files. For example, an inquiry about a nonexistent account returns RMS$_RNF, record not found status. For a description of RMS status codes that are returned by this service, refer to the *OpenVMS Record Management Services Reference Manual*.

$GETUTC
Get UTC Time

Returns the current time in 128-bit UTC format.

Format

SYS$GETUTC utcadr

Arguments

utcadr

OpenVMS usage:	coordinated universal time
type:	utc_date_time
access:	write only
mechanism:	by reference

The 128-bit time value to be returned.

Description

The Get UTC Time service returns the current system time in 128-bit UTC format. System time is updated every 10 milliseconds.

Alpha

On Alpha systems, the frequency at which system time is updated varies, depending on the clock frequency of the Alpha processor.♦

Required Access or Privileges

None

Required Quota

None

Related Services

$ASCUTC, $BINUTC, $NUMUTC, $TIMCON

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The argument was not accessible for write in the mode of the caller.

$GET_ALIGN_FAULT_DATA (Alpha Only) Get Alignment Fault Data

On Alpha systems, obtains data from the user image alignment fault buffer if buffered user alignment fault data reporting has been enabled.

This service accepts 64-bit addresses.

Format

SYS$GET_ALIGN_FAULT_DATA buffer ,buffer_size ,return_size

Arguments

buffer
OpenVMS usage: address
type: longword (unsigned)
access: read/write
mechanism: by 32-bit or 64-bit reference

The user buffer in which the alignment fault data is to be stored. The **buffer** is the 32-bit or 64-bit address of this user buffer.

buffer_size
OpenVMS usage: byte count
type: longword (signed)
access: read
mechanism: by value

The size, in bytes, of the buffer specified by the **buffer** argument.

return_size
OpenVMS usage: longword_signed
type: longword (signed)
access: write
mechanism: by 32-bit or 64-bit reference

The amount of data, in bytes, stored in the buffer. The **return_size** argument is the 32-bit or 64-bit address of a naturally aligned longword into which the service returns the size of the buffer. The **return_size** is set to 0 if there is no data in the buffer.

Description

The Get Alignment Fault Data service obtains data from the user image alignment fault buffer if buffered user alignment fault data reporting has been enabled.

When buffered user alignment fault data reporting is enabled, the operating system writes each alignment fault into a user-defined buffer. The user must poll this buffer periodically to read the data.

The user must call the $START_ALIGN_FAULT_REPORT service to enable buffered user alignment fault data reporting.

For more information about buffered user alignment fault data reporting, see the $START_ALIGN_FAULT_REPORT system service.

Required Access or Privileges

None

Required Quota

None

Related Services

$GET_SYS_ALIGN_FAULT_DATA, $INIT_SYS_ALIGN_FAULT_REPORT, $PERM_DIS_ALIGN_FAULT_REPORT, $PERM_REPORT_ALIGN_FAULT, $START_ALIGN_FAULT_REPORT, $STOP_ALIGN_FAULT_REPORT, $STOP_SYS_ALIGN_FAULT_REPORT

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The buffer named in the **buffer** argument is not accessible.
SS$_AFR_NOT_ENABLED	Alignment fault reporting has not been enabled.
SS$_BADPARAM	The buffer size is smaller than the minimum defined by the AFR$K_USER_LENGTH symbol.

$GET_ARITH_EXCEPTION (Alpha Only) Get Arithmetic Exception Information

On Alpha systems, returns information about the exception context for a given arithmetic exception.

Format

SYS$GET_ARITH_EXCEPTION sigarg ,mcharg ,buffer

Arguments

sigarg

OpenVMS usage:	signal array
type:	vector_longword_signed
access:	read only
mechanism:	by reference

Address of the signal array for the given arithmetic exception.

mcharg

OpenVMS usage:	mech array
type:	vector_quadword_unsigned
access:	read only
mechanism:	by reference

Address of the mechanism array for the given arithmetic exception.

buffer

OpenVMS usage:	vector_quadword
type:	vector_quadword_unsigned
access:	write only
mechanism:	by descriptor

Four-quadword buffer to receive additional exception context. The **buffer** argument is the address of a descriptor that points to this buffer.

Description

The Get Arithmetic Exception Information service returns, to the buffer specified by the **buffer** argument, the following information for a given arithmetic exception in an array of quadwords:

- First quadword, the PC of the triggering instruction in the trap shadow
- Second quadword, a copy of the triggering instruction
- Third quadword, the exception summary
- Fourth quadword, the register write mask

Required Access or Privilege

None

Required Quota

None

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The specified buffer cannot be written.

$GET_SECURITY
Get Security Characteristics

Retrieves the security characteristics of an object.

Format

SYS$GET_SECURITY [clsnam] ,[objnam] ,[objhan] ,[flags] ,[itmlst] ,[contxt] ,[acmode]

Arguments

clsnam

OpenVMS usage:	char_string
type:	character-coded text string
access:	read only
mechanism:	by descriptor

Name of the object class. The **clsnam** argument is the address of a descriptor pointing to a string containing the name of the object class. The following is a list of protected object class names:

CAPABILITY
COMMON_EVENT_CLUSTER
DEVICE
FILE
GROUP_GLOBAL_SECTION
LOGICAL_NAME_TABLE
QUEUE
RESOURCE_DOMAIN
SECURITY_CLASS
SYSTEM_GLOBAL_SECTION
VOLUME

objnam

OpenVMS usage:	char_string
type:	character-coded text string
access:	read only
mechanism:	by descriptor

Name of the protected object whose associated security profile is going to be retrieved. The **objnam** argument is the address of a descriptor pointing to a string containing the name of the protected object.

The format of an object name is class specific. The following table lists object names and describes their formats.

Object Class	Object Name Format
CAPABILITY	A character string. Currently, the only capability object is VECTOR.
COMMON_EVENT_CLUSTER	Name of the event flag cluster, as defined in the Associate Common Event Flag Cluster ($ASCEFC) system service.

Object Class	Object Name Format
DEVICE	Standard device specification, described in the *OpenVMS User's Manual.*
FILE	Standard file specification, described in the *OpenVMS User's Manual.*
GROUP_GLOBAL_SECTION	Section name, as defined in the Create and Map Section ($CRMPSC) system service.
LOGICAL_NAME_TABLE	Table name, as defined in the Create Logical Name Table ($CRELNT) system service.
QUEUE	Standard queue name, as described in the Send to Job Controller ($SNDJBC) system service.
RESOURCE_DOMAIN	An identifier or octal string enclosed in brackets.
SECURITY_CLASS	Any class name shown in column 1, or a class name followed by a period (.) and the template name. Use the DCL command SHOW SECURITY to display possible template names.
SYSTEM_GLOBAL_SECTION	Section name, as defined in the Create and Map Section ($CRMPSC) system service.
VOLUME	Volume name or name of the device on which the volume is mounted.

objhan

OpenVMS usage: object_handle
type: longword (unsigned)
access: read only
mechanism: by reference

Data structure identifying the object whose associated characteristics are going to be retrieved. The **objhan** argument is an address of a longword containing the object handle. You can use the **objhan** argument as an alternative to the **objnam** argument; for example, channel number clearly specifies the file open on the channel and can serve as an object handle. The following table shows the format of the object classes.

Object Class	Object Handle Format
COMMON_EVENT_CLUSTER	Event flag number
DEVICE	Channel number
FILE	Channel number
RESOURCE_DOMAIN	Resource domain identifier
VOLUME	Channel number

flags

OpenVMS usage: flags
type: mask_longword
access: read only
mechanism: by value

Mask specifying processing options. The **flags** argument is a longword bit vector wherein a bit, when set, specifies the processing option. The **flags** argument requires the **contxt** argument. The following table describes each flag.

Symbolic Name	Description
OSS$M_RELCTX	Release the context structure at the completion of this request.
OSS$M_WLOCK	Maintain a write lock on the security profile at the completion of this request. $GET_SECURITY ignores the flag if the context has already been established.

These symbolic names are defined in the $OSSDEF macro. You construct the **flags** argument by specifying the symbolic names of each flag.

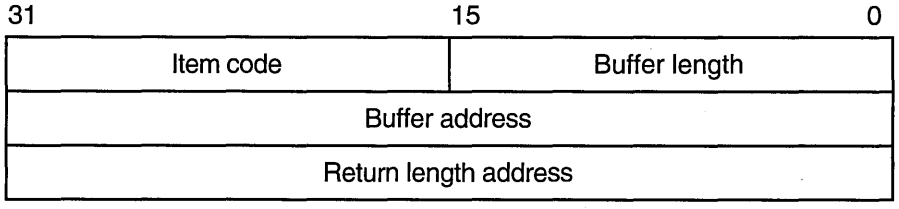
itmlst

OpenVMS 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.

With the item list, the user retrieves the protected object's characteristics. The user defines which security characteristics to retrieve. If this argument is not present, only the **flags** argument is processed. Without the **itmlst** argument, you can *only* manipulate the security profile lock or release **contxt** resources.

The following diagram depicts a single item descriptor.



ZK-5186A-GE

The following table describes the item descriptor fields.

Descriptor Field	Definition
Buffer length	A word containing an integer specifying the length (in bytes) of the buffer in which $GET_SECURITY 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, $GET_SECURITY truncates the data.
Item code	A word containing a symbolic code specifying the item of information that $GET_SECURITY is to return. The $OSSDEF macro defines these codes. A description of each item code is given in the Item Codes section.
Buffer address	A longword containing the address of the buffer in which $GET_SECURITY is to write the information.
Return length address	A longword containing the address of a word in which $GET_SECURITY writes the length (in bytes) of the information it actually returns.

contxt

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

Value used to maintain the processing context when dealing with a single protected object across multiple $GET_SECURITY/$SET_SECURITY calls. Whenever the context value is nonzero, the class name, object name, or object handle arguments are disregarded. An input value of 0 indicates that a new context should be established.

Because an active context block consumes process memory, be sure to release the context block by setting the RELCTX flag when the profile processing is complete. $GET_SECURITY sets the context argument to 0 once the context is released.

acmode

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

Access mode to be used in the object protection check. 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** with the caller's access mode and uses the least privileged mode. The access modes are defined in the system macro $PSLDEF library. Digital recommends that this argument be omitted (passed as zero).

Item Codes

The following table provides a summary of item codes that are valid in an item descriptor in the **itmlst** argument. Complete descriptions of each item code are provided after the table.

Item Identifier	Description
OSS$_ACCESS_NAMES	Returns access bitname translation table for the class.
OSS$_ACCESS_NAMES_LENGTH	Returns the size (in bytes) of the access bitname translation table.
OSS$_ACL_FIND_ENTRY	Locates an access control entry (ACE).
OSS$_ACL_FIND_NEXT	Positions to the next ACE.
OSS$_ACL_FIND_TYPE	Locates an ACE of specified type.
OSS$_ACL_GRANT_ACE	Locates an ACE that either grants or denies access.
OSS$_ACL_LENGTH	Returns the length of the access control list (ACL).
OSS$_ACL_POSITION_BOTTOM	Sets a marker that points to the end of the ACL.
OSS$_ACL_POSITION_TOP	Sets a marker that points to the beginning of the ACL.
OSS$_ACL_READ	Reads the entire ACL.
OSS$_ACL_READ_ENTRY	Reads an ACE.
OSS$_CLASS_NAME	Returns the full object class name.
OSS$_FIRST_TEMPLATE	Returns the name of the first template profile of a Security_Class object.
OSS$_NEXT_OBJECT	Returns the name of the next Security_ Class object.
OSS$_NEXT_TEMPLATE	Returns the name of the next template profile of a Security_Class object.
OSS$_OBJECT_NAME	Returns the name of the object. The FILE class does not return an object name.
OSS$_OWNER	Returns the UIC or general identifier of the object's owner.
OSS$_PROTECTION	Returns the protection code of the object.

OSS$_ACCESS_NAMES

When you specify OSS$_ACCESS_NAMES, $GET_SECURITY returns the access name translation table in the buffer pointed to by the buffer address field of the item descriptor.

The access name translation table is a 32-quadword vector followed by a variable section containing the access names. Each bit in the vector represents a single access type. The contents of the quadword is a string descriptor that corresponds to the ASCII bitname string. Undefined access types have zero-length names. The return length, if present, returns the length of the table.

OSS$_ACCESS_NAMES_LENGTH
When you specify OSS$_ACCESS_NAMES_LENGTH, $GET_SECURITY returns the length of the access name translation table.

OSS$_ACL_FIND_ENTRY
When you specify OSS$_ACL_FIND_ENTRY, $GET_SECURITY locates an ACE pointed to by the buffer address. OSS$_ACL_FIND_ENTRY sets the position within the ACL for succeeding ACL operations; for example, for a deletion or modification of the ACE. If the buffer address is 0, it returns SS$_ACCVIO.

OSS$_ACL_FIND_NEXT
When you specify OSS$_ACL_FIND_NEXT, $GET_SECURITY advances the current position to the next ACE in the ACL.

OSS$_ACL_FIND_TYPE
When you specify OSS$_ACL_FIND_TYPE, $GET_SECURITY returns an ACE of a particular type if there is one in the buffer pointed to by the buffer address. OSS$_ACL_FIND_TYPE sets the position within the ACL for succeeding ACL operations. If the buffer address is 0, it returns SS$_ACCVIO.

OSS$_ACL_GRANT_ACE
When you specify OSS$_ACL_GRANT_ACE, $GET_SECURITY returns the ACE in the object's ACL that grants or denies the user access to that object. OSS$_ ACL_GRANT_ACE returns the ACE found in the buffer pointed to by the buffer address.

OSS$_ACL_LENGTH
When you specify OSS$_ACL_LENGTH, $GET_SECURITY returns the size (in bytes) of the object's ACL. The buffer address field points to a longword that receives the size.

OSS$_ACL_POSITION_BOTTOM
When you specify OSS$_ACL_POSITION_BOTTOM, $GET_SECURITY sets the ACL position to point to the bottom of the ACL.

OSS$_ACL_POSITION_TOP
When you specify OSS$_ACL_POSITION_TOP, $GET_SECURITY sets the ACL position to point to the top of the ACL.

OSS$_ACL_READ
When you specify OSS$_ACL_READ, $GET_SECURITY returns the portion of the object's ACL to the buffer pointed to by the buffer address.

OSS$_ACL_READ_ENTRY
When you specify OSS$_ACL_READ_ENTRY, $GET_SECURITY reads the ACE pointed to by the buffer address.

OSS$_CLASS_NAME
When you specify OSS$_CLASS_NAME, $GET_SECURITY returns the full object class name.

OSS$_FIRST_TEMPLATE
When you specify OSS$_FIRST_TEMPLATE, $GET_SECURITY returns the name of the first template profile for the object named in the **objnam** argument. This item code is valid only for security class objects. If the **clsnam** is not Security_Class, SS$_INVITMCLS is returned.

OSS$_NEXT_OBJECT
When you specify OSS$_NEXT_OBJECT, $GET_SECURITY returns the name of the next object. A return length of 0 indicates the end of the list. This item code is valid only for security class objects. If the **clsnam** is not Security_Class, SS$_INVITMCLS is returned.

OSS$_NEXT_TEMPLATE
When you specify OSS$_NEXT_TEMPLATE, $GET_SECURITY returns the name of the next template. This item code allows you to step through a list of an object's templates. A return length of 0 indicates the end of the list. This item code is valid only for security class objects. If the **clsnam** is not Security_Class, SS$_INVITMCLS is returned.

OSS_OBJECT_NAME
When you specify OSS$_OBJECT_NAME, $GET_SECURITY returns the name of the object.

OSS$_OWNER
When you specify OSS$_OWNER, $GET_SECURITY returns the owner of the object.

OSS$_PROTECTION
When you specify OSS$_PROTECTION, $GET_SECURITY returns the protection code of the object.

Description

The Get Security service returns information about security characteristics of a selected object. Security characteristics include such information as the protection code, the owner, and the access control list (ACL). The security management services, $GET_SECURITY and $SET_SECURITY, maintain a single master copy of a profile for every security object in a VMScluster environment. They also ensure that only one process at a time can modify an object's security profile.

There are different ways of identifying which protected object $GET_SECURITY should process:

- Whenever the **contxt** argument has a nonzero value, $GET_SECURITY uses the context to select the object and ignores the class name, object name, and object handle.
- With some types of objects, such as a file or a device, it is possible to select an object on the basis of its **objhan** and **clsnam** values.
- If neither a nonzero **contxt** argument nor an **objhan** argument is provided, $GET_SECURITY uses an object's class name (**clsnam**) and object name (**objnam**) to select the object.

When you call $GET_SECURITY, the service selects the specified protected object and fetches a local copy of the object's security profile.

The context for a security management operation can be established through either $GET_SECURITY or $SET_SECURITY. Whenever the context is set by one service, the other service can use it, provided the necessary locks are being held. If you intend to modify the profile, you must set the write lock flag (OSS$M_WLOCK) when you establish the context.

There are many situations in which the **contxt** argument is essential. By establishing a context for an ACL operation, for example, a caller can retain an ACL position across calls to $GET_SECURITY so that a set of ACEs can be read and modified sequentially. A security context is released by a call to $SET_SECURITY or $GET_SECURITY that sets the OSS$M_RELCTX flag. Once the context is released, the user-supplied context longword is set to 0.

Required Access or Privileges

Read or control access to the object is required.

Required Quota

None

Related Services

$SET_SECURITY

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The parameter cannot be read and the buffer cannot be written.
SS$_BADPARAM	You specified an invalid object, attribute code, or item size.
SS$_INSFARG	The **clsnam** and **objnam** arguments are not specified, the **clsnam** and **objhan** arguments are not specified, or the **contxt** argument is not specified.
SS$_INVITMCLS	The item code that you specified is not supported for the class.
SS$_NOCLASS	The named security class does not exist.
SS$_OBJLOCKED	The selected object is currently write locked.

$GET_SYS_ALIGN_FAULT_DATA (Alpha Only) Get System Alignment Fault Data

On Alpha systems, obtains data from the system alignment fault buffer if buffered system alignment fault data reporting has been enabled.

This service accepts 64-bit addresses.

Format

SYS$GET_SYS_ALIGN_FAULT_DATA buffer ,buffer_size ,return_size

Arguments

buffer
OpenVMS usage: address
type: longword (unsigned)
access: read/write
mechanism: by 32-bit or 64-bit reference

The user buffer in which the alignment fault data is to be stored. The **buffer** argument is the 32-bit or 64-bit virtual address of this buffer.

buffer_size
OpenVMS usage: byte count
type: longword (signed)
access: read
mechanism: by value

The size, in bytes, of the buffer specified by the **buffer** argument.

return_size
OpenVMS usage: longword_signed
type: longword (signed)
access: write
mechanism: by 32-bit or 64-bit reference

The amount of data, in bytes, stored in the buffer. The **return_size** argument is the 32-bit or 64-bit virtual address of a naturally aligned longword into which the service returns the amount of data, in bytes, stored in the buffer. The **return_size** argument is set to 0 if there is no data in the buffer.

Description

The Get System Alignment Fault Data service obtains data from the system alignment fault buffer if buffered system alignment fault data reporting has been enabled.

When buffered system alignment fault data reporting is enabled, the operating system writes each alignment fault into a system-allocated buffer. The user must poll this buffer periodically to read the data.

The user must call the $INIT_SYS_ALIGN_FAULT_REPORT service to enable buffered system alignment fault data reporting. For more information, see the $INIT_SYS_ALIGN_FAULT_REPORT service.

Required Access or Privileges

CMKRNL privilege is required.

Required Quota

None

Related Services

$GET_ALIGN_FAULT_DATA, $INIT_SYS_ALIGN_FAULT_REPORT, $PERM_DIS_ALIGN_FAULT_REPORT, $PERM_REPORT_ALIGN_FAULT, $START_ALIGN_FAULT_REPORT, $STOP_ALIGN_FAULT_REPORT, $STOP_SYS_ALIGN_FAULT_REPORT

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The buffer named in the **buffer** argument is not accessible.
SS$_AFR_NOT_ENABLED	Alignment fault reporting has not been enabled.
SS$_BADPARAM	The buffer size is smaller than the minimum defined by the AFR$K_VMS_LENGTH or the AFR$K_EXTENDED_LENGTH symbol.

$GOTO_UNWIND (Alpha Only) Unwind Call Stack

On Alpha systems, unwinds the call stack.

Format

SYS$GOTO_UNWIND target_invo ,target_pc ,[new_r0] ,[new_r1]

Arguments

target_invo
OpenVMS usage: invo_handle
type: longword (unsigned)
access: read only
mechanism: by reference

The address of a location that contains a handle for the target invocation.

If you do not specify the **target_invo** argument, or if the handle value is 0, an exit unwind is initiated.

target_pc
OpenVMS usage: address
type: longword (unsigned)
access: read only
mechanism: by reference

The address of a location that contains the address at which execution should continue in the target invocation.

If the **target_pc** argument is omitted or the value is 0, a system-defined target PC is assumed and execution resumes at the location specified at the return address for the call frame of the target procedure invocation.

new_r0
OpenVMS usage: quadword_unsigned
type: quadword (unsigned)
access: read only
mechanism: by reference

The address of a location that contains the value to place in the saved R0 location of the mechanism argument vector. The contents of this location are then loaded into the processor R0 register at the time that execution continues in the target invocation.

If the **new_r0** argument is omitted, the contents of the processor R0 register at the time of the call to $GOTO_UNWIND are used.

new_r1
OpenVMS usage: quadword_unsigned
type: quadword (unsigned)
access: read only
mechanism: by reference

Address of a location that contains the value to place in the saved R1 location of the mechanism argument vector. The contents of the location are then loaded

into the processor R1 register at the time that execution continues in the target invocation.

If the **new_r1** argument is omitted, the contents of the processor R1 register at the time of the call to $GOTO_UNWIND are used.

Description

The Unwind Call Stack service provides the function for a procedure to unwind the call stack.

Required Access or Privileges

None

Required Quota

None

Related Services

$UNWIND

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The specified **target_invo**, **target_pc**, **new_r0**, or **new_r1** argument is not accessible.

$GRANTID
Grant Identifier to Process

Adds the specified identifier record to the rights list of the process or the system.

Format

SYS$GRANTID [pidadr] ,[prcnam] ,[id] ,[name] ,[prvatr]

Arguments

pidadr

OpenVMS 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 a 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

OpenVMS 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

OpenVMS 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.

Use the **id** argument to modify the attributes of the identifier.

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$V_DYNAMIC	Allows holders of the identifier to remove it from or add it to the process rights database using the DCL command SET RIGHTS_LIST.
KGB$V_NOACCESS	Makes any access rights of the identifier null and void. This attribute is intended as a modifier for a resource identifier or the Subsystem attribute.
KGB$V_RESOURCE	Allows holders of an identifier to charge disk space to the identifier. It is used only for file objects.
KGB$V_SUBSYSTEM	Allows holders of the identifier to create and maintain protected subsystems by assigning the Subsystem ACE to the application images in the subsystem.

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

OpenVMS 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. The identifier is granted as it is created. You must specify either **id** or **name**.

prvatr

OpenVMS 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.

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.
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 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. The INSFARG condition value is returned.
Omitted	Specified	Specified identifier value is used.
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 and identifier name is ignored.

Note that a value of 0 in either of the preceding tables indicates that the contents of the address specified by the argument is the value 0. The word *omitted* indicates that the argument was not supplied.

Required Access or Privileges

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.

Required Quota

None

Related Services

$ADD_HOLDER, $ADD_IDENT, $ASCTOID, $CHANGE_ACL, $CHECK_ACCESS, $CHKPRO, $CREATE_RDB, $ERAPAT, $FIND_HELD, $FIND_HOLDER, $FINISH_RDB, $FORMAT_ACL, $FORMAT_AUDIT, $HASH_PASSWORD, $IDTOASC, $MOD_HOLDER, $MOD_IDENT, $MTACCESS, $PARSE_ACL, $REM_HOLDER, $REM_IDENT, $REVOKID

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$_ACCVIO	The **pidadr** argument cannot be read or written; **prcnam** cannot be read; **id** cannot be read or written; the **name** cannot be read; or **prvatr** cannot be written.
SS$_INSFARG	You did not specify either the **id** or the **name** argument.
SS$_INSFMEM	The process dynamic memory is insufficient for opening the rights database.
SS$_IVIDENT	The specified identifier name is invalid; the identifier name is longer than 31 characters, contains an illegal character, or does not contain at least one nonnumeric character.
SS$_IVLOGNAM	You specified an invalid process name.
SS$_NONEXPR	You specified a nonexistent process.
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$_NOSYSNAM	The operation requires SYSNAM privilege.
SS$_RIGHTSFULL	The rights list of the process or system is full.
RMS$_PRV	The user does not have read access to the rights database.

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

$HASH_PASSWORD
Hash Password

Applies the hash algorithm you select to an ASCII password string and returns a quadword hash value that represents the encrypted password.

Format

SYS$HASH_PASSWORD pwd ,alg ,[salt] ,usrnam ,hash

Arguments

pwd

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

ASCII password string to be encrypted. The **pwd** argument is the address of a character string descriptor pointing to the ASCII password. The password string can contain between 1 and 32 characters and use the uppercase characters A through Z, the numbers 0 through 9, the dollar sign ($), and the underscore (_).

alg

OpenVMS usage:	byte_unsigned
type:	byte (unsigned)
access:	read only
mechanism:	by value

Algorithm used to hash the ASCII password string. The **alg** argument is an unsigned byte specifying the hash algorithm. The operating system recognizes the following algorithms.

Symbolic Name	Description
UAI$K_AD_II	Uses a CRC algorithm and returns a longword hash value. This algorithm was used in releases prior to VAX/VMS Version 2.0.
UAI$C_PURDY	Uses a Purdy algorithm over salted input. It expects a blank-padded user name and returns a quadword hash value. This algorithm was used during VAX/VMS Version 2.0 field test.
UAI$C_PURDY_V	Uses the Purdy algorithm over salted input. It expects a variable-length user name and returns a quadword hash value. This algorithm was used in releases prior to VMS Version 5.4.
UAI$K_PURDY_S	Uses the Purdy algorithm over salted input. It expects a variable-length user name and returns a quadword hash value. This algorithm is used to hash all new passwords in VMS Version 5.4 and later.

Symbolic Name	Description
UAI$C_PREFERED_ ALGORITHM[1]	Represents the latest encryption algorithm that the operating system uses to encrypt new passwords. Currently, it equates to UAI$C_PURDY_S. Digital recommends that you use this symbol in source modules because it always equates with the most recent algorithm.

[1] The value of this symbol might be changed in future releases if an additional algorithm is introduced.

Values ranging from 128 to 255 are reserved for customer use; the constant UAI$K_CUST_ALGORITHM defines the start of this range.

You can use the UAI$_ENCRYPT and UAI$_ENCRYPT2 item codes with the $GETUAI system service to retrieve the primary and secondary password hash algorithms for a user.

salt

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

Value used to increase the effectiveness of the hash. The **salt** argument is an unsigned word containing 16 bits of data that is used by the hash algorithms when encrypting a password for the associated user name. The $GETUAI item code UAI$_SALT is used to retrieve the SALT value for a given user. If you do not specify a SALT value, $HASH_PASSWORD uses the value of 0.

usrnam

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

Name of the user associated with the password. The **usrnam** argument is the address of a descriptor pointing to a character text string containing the user name. The current password encryption algorithm (UAI$C_PURDY_S) folds the user name into the ASCII password string to ensure that different users with the same password produce different hash values. This argument must be supplied for all calls to $HASH_PASSWORD but is ignored when using the CRC algorithm (UAI$C_AD_II).

hash

OpenVMS usage:	quadword_unsigned
type:	quadword (unsigned)
access:	write only
mechanism:	by reference

Output hash value representing the encrypted password. The **hash** argument is the address of an unsigned quadword to which $HASH_PASSWORD writes the output of the hash. If you use the UAI$C_AD_II algorithm, the second longword of the hash is always set to 0.

Description

The Hash Password service applies the hash algorithm you select to an ASCII password string and returns a quadword hash value that represents the encrypted password.

Required Access or Privileges

None

Required Quota

None

Related Services

$GETUAI, $SETUAI.

Use $GETUAI to get the values for the **salt** and **alg** arguments. Use $SETUAI to store the resulting hash using the item codes UAI$_PWD and UAI$_PWD2.

For more information, see the appendix on implementing site-specific security policies in the *OpenVMS Programming Concepts Manual*.

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The input or output buffer descriptors cannot be read or written to by the caller.
SS$_BADPARAM	The specified hash algorithm is unknown or invalid.

$HIBER
Hibernate

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.

Format

SYS$HIBER

Arguments

None.

Description

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.

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 of outstanding wakeup requests is maintained.

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()
```

Required Access or Privileges

None

Required Quota

None

Related Services

$CANEXH, $CREPRC, $DCLEXH, $DELPRC, $EXIT, $FORCEX, $GETJPI, $GETJPIW, $PROCESS_SCAN, $RESUME, $SETPRI, $SETPRN, $SETPRV, $SETRWM, $SUSPND, $WAKE

Condition Values Returned

SS$_NORMAL	The service completed successfully.

$IDTOASC
Translate Identifier to Identifier Name

Translates the specified identifier value to its identifier name.

Format

SYS$IDTOASC id ,[namlen] ,[nambuf] ,[resid] ,[attrib] ,[contxt]

Arguments

id
OpenVMS 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 $IDTOASC repeatedly until it returns the status code SS$_NOSUCHID. The identifiers are returned in alphabetical order.

namlen
OpenVMS 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
OpenVMS 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
OpenVMS 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

OpenVMS 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$V_DYNAMIC	Allows holders of the identifier to remove it from or add it to the process rights list using the DCL command SET RIGHTS_LIST.
KGB$V_NAME_HIDDEN	Allows holders of an identifier to have it translated—either from binary to ASCII or vice versa—but prevents unauthorized users from translating the identifier.
KGB$V_NOACCESS	Makes any access rights of the identifier null and void. This attribute is intended as a modifier for a resource identifier or the Subsystem attribute.
KGB$V_RESOURCE	Allows holders of an identifier to charge disk space to the identifier. It is used only for file objects.
KGB$V_SUBSYSTEM	Allows holders of the identifier to create and maintain protected subsystems by assigning the Subsystem ACE to the application images in the subsystem.

contxt

OpenVMS 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 the value 0, 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 can also use it to find all identifiers in the rights database. Owner or read access to the rights database is required. 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 $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.

Required Access or Privileges

None, unless the **id** argument is NAME_HIDDEN, in which case you must hold the identifier or have read access to the rights list.

Required Quota

None

Related Services

$ADD_HOLDER, $ADD_IDENT, $ASCTOID, $CHANGE_ACL, $CHECK_ACCESS, $CHKPRO, $CREATE_RDB, $ERAPAT, $FIND_HELD, $FIND_HOLDER, $FINISH_RDB, $FORMAT_ACL, $FORMAT_AUDIT, $GRANTID, $HASH_PASSWORD, $MOD_HOLDER, $MOD_IDENT, $MTACCESS, $PARSE_ACL, $REM_HOLDER, $REM_IDENT, $REVOKID

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$_NORIGHTSDB	The rights database does not exist.
SS$_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.

Because the rights database is an indexed file that you access with OpenVMS RMS, this service can also return RMS status codes associated with operations on indexed files. For descriptions of these status codes, refer to the *OpenVMS Record Management Services Reference Manual.*

$IEEE_SET_FP_CONTROL (Alpha Only) Set IEEE Floating-Point Control Register

On Alpha systems, modifies the software IEEE floating-point control register and, optionally, returns the previous register value.

The service provides the mechanism to set the specified bits in the IEEE floating-point control register, to clear the specified bits in the register, and to swap the values of the register.

Format

SYS$IEEE_SET_FP_CONTROL [clrmsk] ,[setmsk] ,[prvmsk]

Arguments

clrmsk

OpenVMS usage: mask_quadword
type: quadword (unsigned)
access: read only
mechanism: by reference

Address of a quadword bit mask to be cleared in the IEEE floating-point control register.

The $IEEEDEF macro defines symbols for the floating-point control register. Table SYS2–2 shows the symbols, their corresponding masks, and their meaning.

Table SYS2–2 Format of the IEEE Floating-Point Control Register (Alpha Only)

Symbol	Mask	Meaning
IEEE$M_TRAP_ENABLE_INV	2	Invalid operation
IEEE$M_TRAP_ENABLE_DZE	4	Divide by 0
IEEE$M_TRAP_ENABLE_OVF	8	Overflow
IEEE$M_TRAP_ENABLE_UNF	10	Underflow
IEEE$M_TRAP_ENABLE_INE	20	Inexact
IEEE$M_MAP_UMZ	4000	Underflows are mapped to 0.0
IEEE$M_INHERIT	8000	Inherit FP state on thread create
IEEE$M_STATUS_INV	20000	Invalid operation
IEEE$M_STATUS_DZE	40000	Divide by 0
IEEE$M_STATUS_OVF	80000	Overflow
IEEE$M_STATUS_UNF	100000	Underflow
IEEE$M_STATUS_INE	200000	Inexact

setmsk

OpenVMS usage: mask_quadword
type: quadword (unsigned)
access: read only
mechanism: by reference

Address of a quadword bit mask to be set in the IEEE floating-point control register.

Table SYS2–2 shows the format of the IEEE floating-point control register.

prvmsk

OpenVMS usage:	mask_quadword
type:	quadword (unsigned)
access:	write only
mechanism:	by reference

Address of a quadword to receive the previous value of the IEEE floating-point control register.

Description

The Set IEEE Floating-Point Control Register service updates the IEEE floating-point control register, maintained by the operating system, with the values supplied by the calling program.

The following steps are used to update the register:

1. If the **prvmsk** argument is specified, $IEEE_SET_FP_CONTROL first reads the previous value of the IEEE floating-point control register.
2. If the **clrmsk** argument is specified, $IEEE_SET_FP_CONTROL then clears the specified bit masks in the **clrmsk** argument.
3. If the **setmsk** argument is specified, $IEEE_SET_FP_CONTROL then sets the specified bit masks in the **setmsk** argument.

A program can swap the IEEE floating-point control register (that is, save the old value and specify a new value) by specifying the following:

- The **clrmsk** argument with the address of a quadword of all 1s
- The **setmsk** argument with the address of a quadword that holds the new register value
- The **prvmsk** argument with the address of a quadword to save the old register value

Required Access or Privilege

None

Required Quota

None

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The specified argument cannot be read or cannot be written.

$INIT_SYS_ALIGN_FAULT_REPORT (Alpha Only) Initialize System Alignment Fault Reporting

On Alpha systems, initializes system process alignment fault reporting.

This service accepts 64-bit addresses.

Format

SYS$INIT_SYS_ALIGN_FAULT_REPORT match_table ,buffer_size ,flags

Arguments

match_table

OpenVMS usage:	address
type:	longword (unsigned)
access:	read
mechanism:	by 32-bit or 64-bit reference

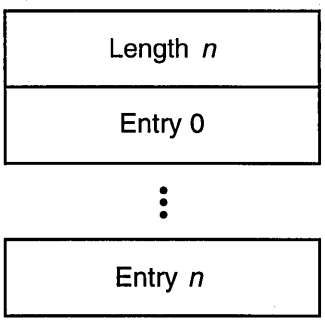
Describes the system fault match table. The **match_table** argument is the 32-bit or 64-bit virtual address of an array of longwords describing the system fault match table. The first longword is the number of match entries; the remaining longwords are the match entries.

The match table is used to restrict the number of alignment faults reported. Each entry in the table is a bit mask divided into three groups: mode bits, program counter (PC) space bits, and virtual address (VA) space bits.

The following table lists the symbols that can be used to define these bits.

Bit Type	Symbols	
Mode bits	AME$M_KERNEL_MODE	Kernel mode
	AME$M_EXEC_MODE	Executive mode
	AME$M_SUPER_MODE	Supervisor mode
	AME$M_USER_MODE	User mode
Program counter bits	AME$M_USER_PC	PC in User space
	AME$M_SYSTEM_PC	PC in System space
Virtual address bits	AME$M_SYSTEM_VA	VA in System space
	AME$M_USER_VA_P0	VA in User P0 space
	AME$M_USER_VA_P1	VA in User P1 space
	AME$M_USER_VA_P2	VA in User P2 space

The following diagram illustrates the data structure of the match table.



ZK-4981A-GE

When an alignment fault occurs, a fault bit mask is created with one bit set in each group. The alignment fault handler then compares this fault bit mask against each entry in the match table. If the fault bit mask is a subset of an entry in the match table, the fault is reported.

buffer_size

OpenVMS usage:	byte count
type:	longword (signed)
access:	read
mechanism:	by value

The number of bytes to allocate, from nonpaged pool, to save the alignment fault data. The buffer you allocate must be sufficient to accommodate one data item of the size specified in the **flags** argument.

flags

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

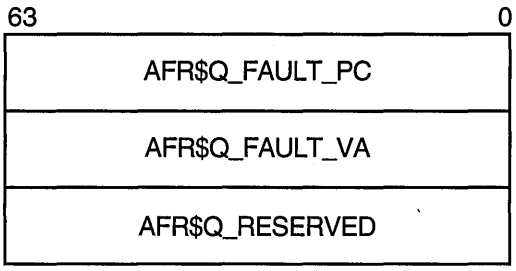
Flag bit mask specifying options for the $GET_SYS_ALIGN_FAULT_DATA operation.

If the **flags** argument is 0, data items of size AFR$K_VMS_LENGTH will be returned. If the **flags** argument is AFR$M_USER_INFO, the user name and image name are added to each data item and they are returned in a buffer of length AFR$K_EXTENDED_LENGTH. If the user name and image name are not available, an empty string is returned in the data item.

Description

The Initialize System Alignment Fault Reporting service initializes system alignment fault reporting.

System alignment faults must be written to a buffer. The following diagram illustrates the format in which system alignment fault data is saved in the buffer.



ZK-4982A-GE

Only one user on a system can initialize system alignment fault reporting at any time. Subsequent calls will return SS$_AFR_ENABLED.

System alignment fault reporting is disabled when the program that called the service completes.

Required Access or Privileges

CMKRNL privilege is required.

Required Quota

None

Related Services

$GET_ALIGN_FAULT_DATA, $GET_SYS_ALIGN_FAULT_DATA, $PERM_DIS_ALIGN_FAULT_REPORT, $PERM_REPORT_ALIGN_FAULT, $START_ALIGN_FAULT_REPORT, $STOP_ALIGN_FAULT_REPORT, $STOP_SYS_ALIGN_FAULT_REPORT

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The match table is not read accessible.
SS$_AFR_ENABLED	The service was already called.
SS$_BADPARAM	The **buffer_size** argument is less than the minimum size required. If the **flags** argument is 0, AFR$K_VMS_LENGTH + 32 is required. If the **flags** argument is 1, AFR$K_EXTENDED_LENGTH + 32 is required.
SS$_NOPRIV	The caller does not have CMKRNL privilege.

$INIT_VOL
Initialize Volume

Formats a disk or magnetic tape volume and writes a label on the volume. At the end of initialization, the disk is empty except for the system files containing the structure information. All former contents of the volume are lost.

Format

SYS$INIT_VOL devnam, volnam [,itmlst]

Arguments

devnam

OpenVMS usage:	char_string
type:	character string
access:	read only
mechanism:	by descriptor

Name of the device on which the volume is physically mounted. The descriptor must point to the device name, a character string of 1 to 64 characters. The device name can be a physical device name or a logical name; if it is a logical name, it must translate to a physical name.

The device does not have to be currently allocated; however, allocating the device before initializing it is recommended.

volnam

OpenVMS usage:	char_string
type:	character string
access:	read only
mechanism:	by descriptor

Identification to be encoded on the volume. The descriptor must point to the volume name, a character string of 1 to 12 characters. For a disk volume name, you can specify a maximum of 12 ANSI characters; for a magnetic tape volume name, you can specify a maximum of 6 ANSI "a" characters. Any valid ANSI "a" characters can be used; these include numbers, uppercase letters, and any one of the following nonalphanumeric characters:

! " % ' ( ) * + , - . / : ; < = >

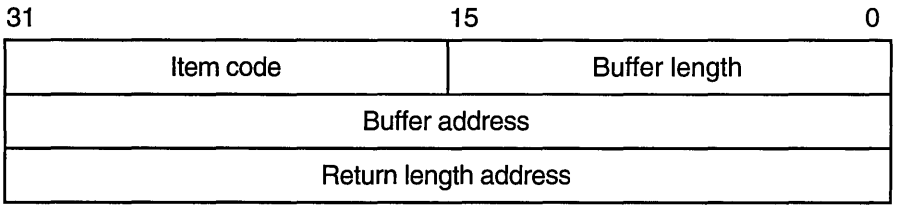
Digital strongly recommends that a disk volume name consist of only alphanumeric characters, dollar signs ($), underscores (_), and hyphens (-).

itmlst

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

Item list specifying options that can be used when initializing the volume. The **itmlst** argument is the address of a list of item descriptors, each of which describes one option. The list of item descriptors is terminated by a longword of 0.

The following diagram depicts the format of a single item descriptor.



ZK-5186A-GE

The following table defines the item descriptor fields.

Descriptor Field	Definition	
Buffer length	A word specifying the length (in bytes) of the buffer that supplies the information $INIT_VOL needs to process the specific item code. The length of the buffer needed depends upon the item code specified in the item descriptor.	
Item code	A word containing an option for the initialize operation. These codes are defined by the $INITDEF macro. There are three types of item codes:	
	Boolean item code	Boolean item codes specify a true or false value. The form INIT$_*code* specifies a true value and the form INIT$_NO_*code* specifies a false value. For Boolean item codes, the buffer length and buffer address fields of the item descriptor must be 0.
	Symbolic value item code	Symbolic value item codes specify one of a specified range of possible choices. The buffer length and buffer address fields of the item descriptor must be 0.
	Input value item code	Input value item codes specify a value to be used by $INIT_VOL. The buffer length and buffer address fields of the item descriptor must be nonzero.
Buffer address	A longword containing the address of the buffer that supplies information to $INIT_VOL.	

Descriptor Field	Definition
Return length address	This field is not used.

Item Codes

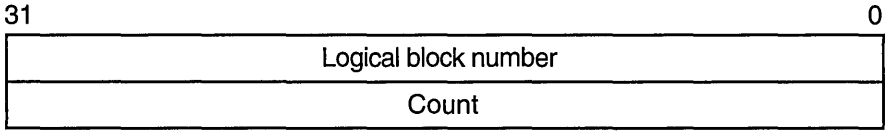
INIT$_ACCESSED
An input item code that specifies the number of directories allowed in system space on the volume.

You must specify an integer between 0 and 255 in the input buffer. The default value is 3.

The INIT$_ACCESSED item code applies only to Files–11 On-Disk Structure Level 1 disks.

INIT$_BADBLOCKS_LBN
An input item code that enables $INIT_VOL to mark bad blocks on the volume; no data is written to those faulty areas. INIT$_BADBLOCKS_LBN specifies faulty areas on the volume by logical block number and block count.

The buffer from which $INIT_VOL reads the option information contains an array of quadwords containing information in the following format.



ZK–1590A–GE

The following table describes the information to be specified for INIT$_ BADBLOCKS_LBN.

Field	Symbol Name	Description
Logical block number	INIT$L_BADBLOCKS_LBN	Specifies the logical block number of the first block to be marked as allocated.
Count	INIT$L_BADBLOCKS_ COUNT	Specifies the number of blocks to be allocated. This range begins with the first block, as specified in INIT$L_ BADBLOCKS_LBN.

For example, if the input buffer contains the values 5 and 3, INIT_VOL starts at logical block number 5 and allocates 3 blocks.

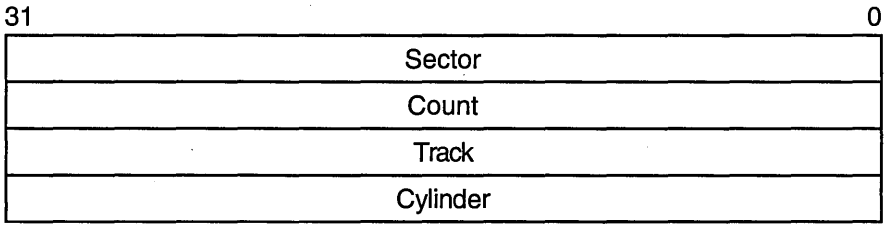
The number of entries in the buffer is determined by the buffer length field in the item descriptor.

All media supplied by Digital and supported on the operating system, except disks and TU58 cartridges, are factory formatted and contain bad block data. The Bad Block Locator utility (BAD) or the diagnostic formatter EVRAC can be used to refresh the bad block data or to construct it for the disks and TU58 cartridges. The INIT$_BADBLOCKS_LBN item code is necessary only to enter bad blocks that are not identified in the volume's bad block data. For more information, see the *OpenVMS Bad Block Locator Utility Manual*.

The INIT$_BADBLOCKS_LBN item code applies only to disks.

INIT$_BADBLOCKS_SEC
An input item code that specifies faulty areas on the volume by sector, track, cylinder, and block count. $INIT_VOL marks the bad blocks as allocated; no data is written to them.

The input buffer must contain an array of octawords containing information in the following format.



ZK-1591A-GE

The following table describes the information to be specified for INIT$_ BADBLOCKS_SEC.

Field	Symbol Name	Description
Sector	INIT$L_BADBLOCKS_ SECTOR	Specifies the sector number of the first block to be marked as allocated.
Count	INIT$L_BADBLOCKS_ COUNT	Specifies the number of blocks to be allocated.
Track	INIT$L_BADBLOCKS_ TRACK	Specifies the track number of the first block to be marked as allocated.
Cylinder	INIT$L_BADBLOCKS_ CYLINDER	Specifies the cylinder number of the first block to be marked as allocated.

For example, if the input buffer contains the values 12, 3, 1, and 2, INIT_VOL starts at sector 12, track 1, cylinder 2, and allocates 3 blocks.

The number of entries in the buffer is determined by the buffer length field in the item descriptor.

All media supplied by Digital and supported on the operating system, except disks and TU58 cartridges, are factory formatted and contain bad block data. The Bad Block Locator utility (BAD) or the diagnostic formatter EVRAC can be used to refresh the bad block data or to construct it for the disks and TU58 cartridges. The INIT$_BADBLOCKS_SEC item code is necessary only to enter bad blocks that are not identified in the volume's bad block data. For more information, see the *OpenVMS Bad Block Locator Utility Manual*.

The INIT$_BADBLOCKS_SEC item code applies only to disks.

INIT$_CLUSTERSIZE
An input item code that specifies the minimum allocation unit in blocks. The input buffer must contain a longword value. The maximum size that can be

specified for a volume is one-hundredth the size of the volume; the minimum size is calculated with the following formula:

$$\frac{\textit{volume size in blocks}}{255 * 4096}$$

The INIT$_CLUSTERSIZE item code applies only to Files–11 On-Disk Structure Level 2 disks (for Files–11 On-Disk Structure Level 1 disks, the cluster size is 1). For Files–11 On-Disk Structure Level 2 disks, the cluster size default depends on the disk capacity.

- Disks that are 50,000 blocks or larger have a default cluster size of 3.
- Disks smaller than 50,000 blocks have a default value of 1.

INIT$_COMPACTION
INIT$_NO_COMPACTION—Default
A Boolean item code that specifies whether data compaction should be performed when writing the volume.

The INIT$_COMPACTION item code applies only to TA90 drives.

INIT$_DENSITY
A symbolic item code that specifies the density value for magnetic tapes and diskettes.

For magnetic tape volumes, the INIT$_DENSITY item code specifies the density in bytes per inch (bpi) at which the magnetic tape is written. Possible symbolic values for tapes are as follows:

- INIT$K_DENSITY_800_BPI
- INIT$K_DENSITY_1600_BPI
- INIT$K_DENSITY_6250_BPI

The specified density value must be supported by the drive. If you do not specify a density item code for a blank magnetic tape, the system uses a default density of the highest value allowed by the tape drive. If the drive allows 6250, 1600, and 800 bpi operation, the default density is 6250. If the drive allows only 1600 and 800 bpi operation, the default density is 1600. If you do not specify a density item code for a magnetic tape that has been previously written, the system uses the previously set volume density.

For diskettes, the INIT$_DENSITY item code specifies how the diskette is to be formatted. Possible symbolic values for diskettes are as follows:

- INIT$K_DENSITY_SINGLE_DISK
- INIT$K_DENSITY_DOUBLE_DISK
- INIT$K_DENSITY_DD_DISK
- INIT$K_DENSITY_HD_DISK

For disk volumes that are to be initialized on RX02, RX23, or RX33 diskette drives, the following values specify how the disk is to be formatted:

- INIT$K_DENSITY_SINGLE_DISK
- INIT$K_DENSITY_DOUBLE_DISK
- INIT$K_DENSITY_DD_DISK

- INIT$K_DENSITY_HD_DISK

Diskettes are initialized as follows:

- RX23 diskettes—DD or HD density
- RX33 diskettes—double density only
- RX02 dual-density diskette drives—single or double density

If you do not specify a density item code for a disk, the system leaves the volume at the density at which it was last formatted. RX02 disks purchased from Digital are formatted in single density.

Note

Disks formatted in double density cannot be read or written by the console block storage device (an RX01 drive) of a VAX–11/780 processor until they have been reformatted in single density.

INIT$_DIRECTORIES
An input item code that specifies the number of entries to preallocate for user directories. The input buffer must contain a longword value in the range of 16 to 16000. The default value is 16.

The INIT$_DIRECTORIES item code applies only to disks.

INIT$_ERASE
INIT$_NO_ERASE—Default
A Boolean item code that specifies whether deleted data should be physically destroyed by performing the data security erase (DSE) operation on the volume before initializing it. The INIT$_ERASE item code applies to the following devices:

- ODS–2 disk volumes
- ANSI magnetic tape volumes on magnetic tape devices that support the hardware erase function, for example, TU78 and MSCP magnetic tapes

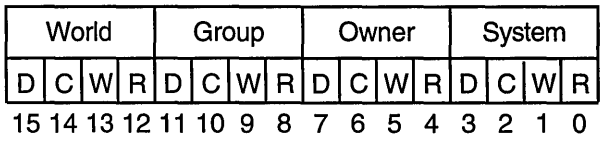
For disk devices, this item code sets the ERASE volume attribute, causing each file on the volume to be erased when it is deleted.

INIT$_EXTENSION
An input item code that specifies, by the number of blocks, the default extension size for all files on the volume. The extension default is used when a file increases to a size greater than its initial default allocation during an update. For Files–11 On-Disk Structure Level 2 disks, the buffer must contain a longword value in the range 0 to 65535. For Files–11 On-Disk Structure Level 1 disks, the input buffer must contain a longword value in the range of 0 to 255. The default value is 5 for both Structure Level 1 and Structure Level 2 disks.

The default extension set by this item code is used only if the following conditions are in effect:

- No default extension for the file has been set
- No default extension for the process has been set using the SET RMS command

INIT$_FPROT
An input item code that specifies the default protection applied to all files on the volume. The input buffer must contain a longword protection mask that contains four 4-bit fields. Each field grants or denies read, write, create, and delete access to a category of users. Cleared bits grant access; set bits deny access. The following diagram depicts the structure of the protection mask on systems.



ZK–5893A–GE

The INIT$_FPROT item code applies only to Files–11 On-Disk Structure Level 1 disks and is ignored if it is used on an OpenVMS system. OpenVMS systems use the default file extension set by the DCL command SET PROTECTION /DEFAULT.

INIT$_HEADERS
An input item code that specifies the number of file headers to be allocated for the index file. The input buffer must contain a longword value within the range of 16 to the value set by the INIT$_MAXFILES item code. The default value is 16.

The INIT$_HEADERS item code applies only to disks.

INIT$_HIGHWATER—Default
INIT$_NO_HIGHWATER
A Boolean item code that sets the file highwater mark (FHM) volume attribute, which guarantees that users cannot read data that they have not written.

INIT$_NO_HIGHWATER disables FHM for a volume.

The INIT$_HIGHWATER and INIT$_NO_HIGHWATER item codes apply only to Files–11 On-Disk Structure Level 2 disks.

INIT$_HOMEBLOCKS
Specifies where the volume's homeblock and spare copy of the homeblock are placed on disk. This item code applies only to Files–11 ODS–2 volumes. It can have the following values:

- INIT$K_HOMEBLOCKS_GEOMETRY

  Causes the homeblocks to be placed at separate locations on disk, to protect against failure of a disk block. Placement depends on the reported geometry of the disk.

- INIT$K_HOMEBLOCKS_FIXED

  Causes the homeblocks to be placed at separate fixed locations on the disk; this is the default. Placement is independent of the reported geometry of the disk. This caters for disks that report different geometries according to the type of controller to which they are attached.

- INIT$K_HOMEBLOCKS_CONTIGUOUS

  Causes the homeblocks to be placed contiguously at the start of the disk. This allows container file systems to maximize the amount of contiguous space on the disk, when used with the INIT$_INDEX_BEGINNING item code.

INIT$_INDEX_BEGINNING
A symbolic item code that places the index file for the volume's directory structure at the beginning of the volume. By default, the index is placed in the middle of the volume.

This item code applies only to disks.

INIT$_INDEX_BLOCK
An input item code that specifies the location of the index file for the volume's directory structure by logical block number. The input buffer must contain a longword value specifying the logical block number of the first block of the index file. By default, the index is placed in the middle of the volume.

The INIT$_INDEX_BLOCK item code applies only to disks.

INIT$_INDEX_END
A symbolic item code that places the index file for the volume's directory structure at the end of the volume. The default is to place the index in the middle of the volume.

This item code applies only to disks.

INIT$_INDEX_MIDDLE
A symbolic item code that places the index file for the volume's directory structure in the middle of the volume. This is the default location for the index.

This item code applies only to disks.

INIT$_LABEL_ACCESS
An input item code that specifies the character to be written in the volume accessibility field of the ANSI volume label VOL1 on an ANSI magnetic tape. Any valid ANSI "a" characters can be used; these include numbers, uppercase letters, and any one of the following nonalphanumeric characters:

! " % ' ( ) * + , - . / : ; < = >

By default, the operating system provides a routine SYS$MTACCESS that checks this field in the following manner:

- If the magnetic tape was created on a version of the operating system that conforms to Version 3 of ANSI, this item code is used to override any character except an ASCII space.
- If the magnetic tape conforms to an ANSI standard that is later than Version 3, this item code is used to override any character except an ASCII 1 character.

INIT$_LABEL_VOLO
An input item code that specifies the text that is written in the owner identifier field of the ANSI volume label VOL1 on an ANSI magnetic tape. The owner identifier field can contain up to 14 valid ANSI "a" characters.

INIT$_MAXFILES
An input item code that restricts the maximum number of files that the volume can contain. The input buffer must contain a longword value between 0 and a value determined by the following calculation:

$$\frac{volume\ size\ in\ blocks}{cluster\ factor + 1}$$

Once initialized, the maximum number of files can be increased only by reinitializing the volume.

The default maximum number of files is calculated as follows:

$$\frac{volume\ size\ in\ blocks}{(cluster\ factor + 1) * 2}$$

The INIT$_MAXFILES item code applies only to disks.

INIT$_OVR_ACCESS
INIT$_NO_OVR_ACCESS—Default
A Boolean item code that specifies whether to override any character in the accessibility field of the ANSI volume label VOL1 on an ANSI magnetic tape. For more information, see the *OpenVMS System Manager's Manual.*

To specify INIT$_OVR_ACCESS, the caller must either own the volume or have VOLPRO privilege.

INIT$_OVR_EXP
INIT$_NO_OVR_EXP—Default
A Boolean item code that specifies whether the caller writes to a magnetic tape that has not yet reached its expiration date. This item code applies only to the magnetic tapes that were created before VAX/VMS Version 4.0 and that use the D% format in the volume owner identifier field.

To specify INIT$_OVR_EXP, the caller must either own the volume or have VOLPRO privilege.

INIT$_OVR_VOLO
INIT$_NO_OVR_VOLO—Default
A Boolean item code that allows the caller to override processing of the owner identifier field of the ANSI volume label VOL1 on an ANSI magnetic tape.

To specify INIT$_OVR_VOLO, the caller must either own the volume or have VOLPRO privilege.

INIT$_OWNER
An input item code that specifies the UIC that will own the volume. The input buffer must contain a longword value, which is the UIC. The default is the UIC of the caller.

For magnetic tapes, no UIC is written unless protection on the magnetic tape is specified. If the INIT$_VPROT item code is specified but the INIT$_OWNER item code is not specified, the UIC of the caller is assigned ownership of the volume.

INIT$_READCHECK
INIT$_NO_READCHECK—Default
A Boolean item code that specifies whether data checking should be performed for all read operations on the volume. For more information about data checking, see the *OpenVMS I/O User's Reference Manual.*

The INIT$_READCHECK item code applies only to disks.

INIT$_SIZE
An input item code that specifies the number of blocks allocated for a RAM disk with a device type of DT$_RAM_DISK. The input buffer must contain a longword value.

INIT$_STRUCTURE_LEVEL_1
INIT$_STRUCTURE_LEVEL_2—Default
Symbolic item codes that specify whether the volume should be formatted in Files–11 On-Disk Structure Level 1 or Structure Level 2. Structure Level 1 is incompatible with the following item codes:

- INIT$_READCHECK
- INIT$_WRITECHECK
- INIT$_CLUSTERSIZE

The default protection for a Structure Level 1 disk is full access to system, owner, and group users, and read access to all other users.

The INIT$_STRUCTURE_LEVEL_1 item code applies only to disks.

INIT$_USER_NAME
An input item code that specifies the user name that is associated with the volume. The input buffer must contain a character string from 1 to 12 alphanumeric characters, which is the user name. The default is the user name of the caller.

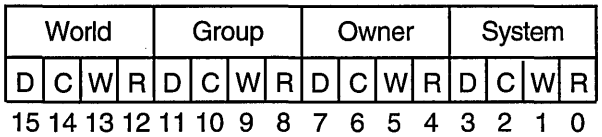
INIT$_VERIFIED
INIT$_NO_VERIFIED
A Boolean item code that indicates whether the disk contains bad block data. INIT$_NO_VERIFIED indicates that any bad block data on the disk should be ignored. For disks with 4096 blocks or more, the default is INIT$_VERIFIED.

INIT$_NO_VERIFIED is the default for the following:

- Disks with fewer than 4096 blocks
- DIGITAL Storage Architecture (DSA) devices
- Disks that are not last-track devices

The INIT$_VERIFIED item codes apply only to disks.

INIT$_VPROT
An input item code that specifies the protection assigned to the volume. The input buffer must contain a longword protection mask that contains four 4-bit fields. Each field grants or denies read, write, create, and delete access to a category of users. Cleared bits grant access; set bits deny access. The following diagram depicts the structure of the protection mask.



ZK-5893A-GE

The default is the default protection of the caller.

For magnetic tape, the protection code is written to a specific volume label. The system applies only read and write access restrictions; execute and delete access are ignored. Moreover, the system and the owner are always given read and write access to magnetic tapes, regardless of the protection mask specified.

When you specify a protection mask for a disk volume, access type E (execute) indicates create access.

For Files–11 On-Disk Structure Level 2 volumes, an initial security profile is created from the VOLUME.DEFAULT profile, with the owner and protection as currently defined for INITIALIZE.

You can use the $SET_SECURITY service to modify the security profile after the volume is initialized and mounted.

The caller needs read, write, or control access to the device.

INIT$_WINDOW
The INIT$_WINDOW item code specifies the number of mapping pointers to be allocated for file windows. The input buffer must contain a longword value in the range 7 to 80. The default is 7.

When a file is opened, the file system uses the mapping pointers to access the data in the file.

The INIT$_WINDOW item code applies only to disks.

INIT$_WRITECHECK
INIT$_NO_WRITECHECK—Default
A Boolean item code that specifies whether data checking should be performed for all read operations on the volume. For more information about data checking, see the *OpenVMS I/O User's Reference Manual.*

The INIT$_WRITECHECK item code applies only to disks.

Description

The Initialize Volume system service formats a disk or magnetic tape volume and writes a label on the volume. At the end of initialization, the disk is empty except for the system files containing the structure information. All former contents of the volume are lost.

A blank magnetic tape can sometimes cause unrecoverable errors when it is read. $INIT_VOL attempts to read the volume unless the following three conditions are in effect:

- INIT$_OVR_ACCESS Boolean item code is specified.
- INIT$_OVR_EXP Boolean item code is specified.
- Caller has VOLPRO privilege.

  If the caller has VOLPRO privilege, $INIT_VOL initializes a disk without reading the ownership information. Otherwise, the ownership of the volume is checked.

A blank disk or a diskette with an incorrect format can sometimes cause a fatal drive error. Such a diskette can be initialized successfully by specifying the INIT$_DENSITY item code to format the diskette.

Required Access or Privileges

To initialize a particular volume, the caller must either have volume protection (VOLPRO) privilege or the volume must be one of the following:

- Blank disk or magnetic tape; that is, a volume that has never been written
- Disk that is owned by the caller's UIC or by the UIC [0,0]
- Magnetic tape that allows write access to the caller's UIC or that was not protected when it was initialized

Required Quota

None

Related Services

$ALLOC, $ASSIGN, $BRKTHRU, $BRKTHRUW, $CANCEL, $CREMBX, $DALLOC, $DASSGN, $DELMBX, $DEVICE_SCAN, $DISMOU, $GETDVI, $GETDVIW, $GETMSG, $GETQUI, $GETQUIW, $MOUNT, $PUTMSG, $QIO, $QIOW, $SET_SECURITY, $SNDERR, $SNDJBC, $SNDJBCW, $SNDOPR

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$_BADPARAM	A buffer length of 0 was specified with a nonzero item code or an illegal item code was specified.
SS$_IVSSRQ	A concurrent call to SYS$INIT_VOL is already active for the process.
SS$_NOPRIV	The caller does not have sufficient privilege to initialize the volume.
SS$_NOSUCHDEV	The specified device does not exist on the host system.

The $INIT_VOL service can also return the following condition values, which are specific to the Initialize Volume utility:

INIT$_ALLOCFAIL	Index file allocation failure.
INIT$_BADACCESSED	Value for INIT$_ACCESSED item code out of range.
INIT$_BADBLOCKS	Invalid syntax in bad block list.
INIT$_BADCLUSTER	Value for INIT$_CLUSTER_SIZE item code out of range.
INIT$_BADDENS	Invalid value for INIT$_DENSITY item code.
INIT$_BADDIRECTORIES	Value for INIT$_DIRECTORIES item code out of range.
INIT$_BADEXTENSION	Value for INIT$_EXTENSION item code out of range.
INIT$_BADHEADERS	Value for INIT$_HEADER item code out of range.
INIT$_BADMAXFILES	Value for INIT$_MAXFILES item code out of range.
INIT$_BADOWNID	Invalid value for owner ID.
INIT$_BADRANGE	Bad block address not on volume.
INIT$_BADVOL1	Bad VOL1 ANSI label.
INIT$_BADVOLACC	Invalid value for INIT$_LABEL_ACCESS item code.
INIT$_BADVOLLBL	Invalid value for ANSI tape volume label.

INIT$_BADWINDOWS	Value for INIT$_WINDOWS item code out of range.
INIT$_BLKZERO	Block 0 is bad—volume not bootable.
INIT$_CLUSTER	Unsuitable cluster factor.
INIT$_CONFQUAL	Conflicting options were specified.
INIT$_DIAGPACK	Disk is a diagnostic pack.
INIT$_ERASEFAIL	Volume not completely erased.
INIT$_FACTBAD	Cannot read factory bad block data.
INIT$_ILLOPT	Item codes not appropriate for the device were specified.
INIT$_INDEX	Invalid index file position.
INIT$_LARGECNT	Disk too large to be supported.
INIT$_MAXBAD	Bad block table overflow.
INIT$_MTLBLLONG	Magnetic tape label specified is longer than 6 characters.
INIT$_MTLBLNONA	Magnetic tape label specified contains non-ANSI "a" characters.
INIT$_NOBADDATA	Bad block data not found on volume.
INIT$_NONLOCAL	Device is not a local device.
INIT$_NOTRAN	Logical name cannot be translated.
INIT$_NOTSTRUC1	Options not available with Files–11 On-Disk Structure Level 1.
INIT$_UNKDEV	Unknown device type.

$IO_CLEANUP (Alpha Only) Clean Up Fast I/O

On Alpha systems, returns all resources allocated by $IO_SETUP.

This service accepts 64-bit addresses.

Format

SYS$IO_CLEANUP fandle

Arguments

fandle

OpenVMS usage:	fandle
type:	64-bit integer (unsigned)
access:	read only
mechanism:	by value

A fandle, passed by value, returned by a previous call to $IO_SETUP.

Description

The Clean Up Fast I/O system service returns various internal resources allocated by the $IO_SETUP system service. Buffer objects passed to $IO_SETUP cannot be deleted until every $IO_SETUP call has had a corresponding $IO_CLEANUP call.

Image rundown executes any required $IO_CLEANUP operations on behalf of the process.

Required Privileges

None

Required Quota

None

Related Services

$IO_PERFORM(W), $IO_SETUP

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_BADFANDLE	Argument was not a valid **fandle**.
SS$_BUSY	The fandle cannot be cleaned up because an I/O is in progress. Reissue the call to $IO_CLEANUP after the I/O has finished.

$IO_PERFORM (Alpha Only) Perform Fast I/O

On Alpha systems, starts the Fast I/O operation. The $IO_PERFORM service completes asynchronously. For synchronous completion, use the Perform Fast I/O and Wait ($IO_PERFORMW) service.

This service accepts 64-bit addresses.

Format

SYS$IO_PERFORM fandle ,chan ,iosadr ,bufadr ,buflen ,porint

Arguments

fandle

OpenVMS usage:	fandle
type:	64-bit integer (unsigned)
access:	read only
mechanism:	by value

A fandle returned by a previous call to $IO_SETUP.

chan

OpenVMS usage:	channel
type:	word (unsigned)
access:	read
mechanism:	by value

Software I/O channel number.

iosadr

OpenVMS usage:	address
type:	address
access:	read only
mechanism:	by value

Address of the I/O Status Area (IOSA). This value cannot be 0; that is, an IOSA is required. The **iosadr** must be aligned to a quadword boundary.

bufadr

OpenVMS usage:	char_string
type:	address
access:	read only
mechanism:	by value

The process buffer address. Must be aligned on a 512-byte boundary.

buflen

OpenVMS usage:	byte count
type:	64-bit integer
access:	read only
mechanism:	by value

The byte count for the I/O. The **buflen** argument must be a multiple of 512 bytes. Drivers have further limitations on the maximum size of an I/O request.

porint

OpenVMS usage:	address
type:	pointer or integer
access:	read only
mechanism:	by value

A hardware integer passed unchanged to the driver. For disk devices, this is the media address for the transfer; that is, the virtual block number (VBN) for virtual I/O functions or the logical block number (LBN) for logical I/O functions. This argument is ignored for tape devices.

For drivers with complex parameters, **porint** would be the address of a descriptor or buffer specific to the device and function and would be documented with the driver.

Description

The Perform Fast I/O system service initiates an I/O operation on the channel number specified by the **chan** argument. The bytes specified by the **buflen** argument are transferred between the location (**porint**) on the device driver and the user's buffer starting at the process buffer address (**bufadr**). The byte count is read or written according to the function code previously specfied in the $IO_SETUP call associated with the **fandle** argument.

Upon completion, the I/O status is written to the IOSA starting at the location specified by **iosadr**, and an AST is delivered to the **astadr** address supplied in the $IO_SETUP call associated with **fandle**. The IOSA address is passed to the AST as the AST parameter.

Required Privileges

None

Required Quota

None

Related Services

$IO_CLEANUP, $IO_SETUP, $IO_PERFORMW

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_BADBUFADR	The data buffer does not reside within the bounds of the data buffer object for the fandle.
SS$_BADIOSADR	The IOSA does not reside within the bounds of the IOSA buffer object for this fandle.
SS$_FANDLEBUSY	The operation using this fandle is already in progress.
SS$_IVCHAN	An invalid channel number was specified; that is, a channel number of 0 or a number larger than the number of channels available.
SS$_UNALIGNED	The buffer specified by **bufadr** or **iosadr** is not properly aligned.

SS$_WRONGACMODE	The request is invalid because the **fandle** was created from a more privileged access mode, or the channel was assigned from a more privileged access mode.

Condition Values Returned in the I/O Status Block

The *OpenVMS I/O User's Reference Manual* lists these device-specific condition values for each device.

$IO_PERFORMW (Alpha Only) Perform Fast I/O and Wait

On Alpha systems, starts a Fast I/O operation. The $IO_PERFORMW service completes synchronously; that is, it returns to the caller after performing the Fast I/O operation.

In all other respects, $IO_PERFORMW is identical to $IO_PERFORM. For all other information about the IO_PERFORMW service, refer to the description of $IO_PERFORM in this manual.

Format

SYS$IO_PERFORMW fandle ,chan ,iosadr ,bufadr ,buflen ,porint

$IO_SETUP (Alpha Only) Set Up Fast I/O

On Alpha systems, allocates resources for Fast I/O.

This service accepts 64-bit addresses.

Format

SYS$IO_SETUP func ,bufobj ,iosobj ,astadr ,flags ,return_fandle

Arguments

func

OpenVMS usage:	function_code
type:	longword
access:	read only
mechanism:	by value

I/O function code. Must be one of the following:

- IO$_READVBLK
- IO$_WRITEVBLK
- IO$_READLBLK
- IO$_WRITELBLK

Various function modifiers are supported, depending on the device and driver. Disk drivers support IO$M_NOVCACHE and IO$M_DATACHECK. Some tape devices support IO$M_REVERSE. Illegal modifiers are detected by the $IO_PERFORM(W) service.

bufobj

OpenVMS usage:	buffer object
type:	vector longword (unsigned)
access:	read only
mechanism:	by 32-bit or 64-bit reference

Handle describing the buffer object that contains the user's buffer. This identifier cannot be 0.

iosobj

OpenVMS usage:	object handle
type:	vector longword (unsigned)
access:	read only
mechanism:	by 32-bit or 64-bit reference

Buffer object handle describing the buffer object that contains the I/O Status Area (IOSA). This may or may not be the same identifier as the **bufobj** argument. This identifier cannot be 0.

astadr

OpenVMS usage:	ast_procedure
type:	procedure value
access:	read only
mechanism:	by 32-bit or 64-bit reference

Completion AST routine address (0, if none). There is no **AST parameter** argument. When the AST routine is called, the AST parameter will be the address of the IOSA for the operation. Applications can store data in the IOSA at offset IOSA$IH_CONTEXT.

flags

OpenVMS usage: mask_longword
type: 64-bit integer (unsigned)
access: read only
mechanism: by value

Flag mask. The **flags** argument is a bit vector in which each bit corresponds to a flag. Flags are defined in the module IOSADEF. The following table describes the flags that are valid for the $IO_SETUP service:

Flag	Description
FIO$M_EXPEDITE	This is a *high priority I/O*; that is, it is to be given preferential treatment by the I/O subsystem. Use of this bit requires ALTPRI or PHY_IO privilege.
FIO$M_AST_NOFLOAT	The AST procedure does not use, or call any procedure that uses, any floating-point registers. This is a performance option. If set, AST delivery will neither save nor restore floating-point registers. Caution: Use of floating-point registers when FIO$M_AST_NOFLOAT has been specified can cause unpredictable, difficult to detect, error conditions.

All other bits in the **flags** argument are reserved for future use by Digital and should be specified as 0.

return_fandle

OpenVMS usage: fandle
type: 64-bit integer (unsigned)
access: write only
mechanism: by 32-bit or 64-bit reference

Address of an aligned quadword to receive the fandle for this I/O operation.

Description

The Set Up Fast I/O system service allocates and initializes a number of internal objects based on the parameters supplied. Because these objects are then ready for use when a subsequent $IO_PERFORM or $IO_PERFORMW is issued, the I/O operation will require less CPU time and fewer multiprocessor steps.

Required Privileges

If you use the **flags** argument FIO$M_EXPEDITE, a process must have ALTPRI or PHY_IO privilege.

Required Quota

Byte count.

Related Services

$IO_CLEANUP, $IO_PERFORM(W)

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The fandle does not have 8 bytes of writability, or the two buffer objects do not have 8 bytes of readability each.
SS$_INSFMEM	There is no pool available from which to create a fandle vector, or the fandle vector is already full and an attempted expansion failed.
SS$_ILLIOFUNC	The function code is not valid.
SS$_ILLMODIFIER	The I/O function modifier is not permitted.
SS$_UNALIGNED	The I/O Status Area (IOSA) or data buffer is not aligned on a quadword boundary.

$LCKPAG
Lock Pages in Memory

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]

Arguments

inadr

OpenVMS 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 byte-within-page bits are ignored.

VAX

On VAX systems, if the starting and ending virtual addresses are the same, a single page is locked.♦

retadr

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

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

OpenVMS 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 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.

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.

Alpha

On Alpha systems, if you are attempting to lock executable code, you should issue multiple $LCKPAG calls: one to lock the code pages and others to lock the linkage section references into these pages. ♦

Required Access or Privileges

The calling process must have PSWAPM privilege to lock pages into memory.

Required Quota

None

Related Services

You can unlock pages locked in memory with the Unlock Pages from Memory ($ULKPAG) service. Locked pages are automatically unlocked at image exit.

For more information, see the chapter on memory management in the *OpenVMS Programming Concepts Manual*.

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$_ACCVIO	The input array cannot be read; the output array cannot be written; the page in the specified range is inaccessible or nonexistent; or an attempt to lock pages was made by a caller whose access mode is less privileged than the access mode associated with the pages.
SS$_LCKPAGFUL	The system-defined maximum limit on the number of pages that can be locked in memory has been reached.
SS$_LDWSETFUL	The locked working set is full. If any more pages are locked, not enough dynamic pages will be available to continue execution.

SS$_NOPRIV	The process does not have the privilege to lock pages in memory.
SS$_PAGOWNVIO	The pages could not be locked because the access mode associated with the call to $LCKPAG was less privileged than the access mode associated with the pages that were to be locked.

$LCKPAG_64 (Alpha Only) Lock Pages in Memory

On Alpha systems, locks a 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.

This service accepts 64-bit addresses.

Format

SYS$LCKPAG_64 start_va_64 ,length_64 ,acmode ,return_va_64 ,return_length_64

Arguments

start_va_64

OpenVMS usage:	address
type:	quadword address
access:	read only
mechanism:	by value

The starting virtual address of the pages to be locked. The specified virtual address will be rounded down to a CPU-specific page boundary.

length_64

OpenVMS usage:	byte count
type:	quadword (unsigned)
access:	read only
mechanism:	by value

Length of the virtual address space to be locked. The specified length will be rounded up to a CPU-specific page boundary so that it includes all CPU-specific pages in the requested range.

acmode

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

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

The $PSLDEF macro in STARLET.MLB and the file PSLDEF.H in SYS$STARLET_C.TLB define the following symbols and their values for the four access modes:

Value	Symbolic Name	Access Mode
0	PSL$C_KERNEL	Kernel
1	PSL$C_EXEC	Executive
2	PSL$C_SUPER	Supervisor

Value	Symbolic Name	Access Mode
3	PSL$C_USER	User

The most privileged access mode used is the access mode of the caller. For the $LCKPAG_64 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.

return_va_64
OpenVMS usage: address
type: quadword address
access: write only
mechanism: by 32-bit or 64-bit reference

The lowest process virtual address of the pages locked in memory. The **return_va_64** argument is the 32-bit or 64-bit virtual address of a naturally aligned quadword into which the service returns the virtual address.

return_length_64
OpenVMS usage: byte count
type: quadword (unsigned)
access: write only
mechanism: by 32-bit or 64-bit reference

The 32-bit or 64-bit virtual address of a naturally aligned quadword into which the service returns the length of the virtual address range locked in bytes.

Description

The Lock Pages in Memory service locks a 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.

If the condition value SS$_ACCVIO is returned by this service, a value *cannot* be returned in the memory locations pointed to by the **return_va_64** and **return_length_64** arguments. If a condition value other than SS$_ACCVIO is returned, the returned address and returned length indicate the pages that were successfully locked before the error occurred. If no pages were locked, the **return_va_64** argument will contain the value -1, and a value *cannot* be returned in the memory location pointed to by the **return_length_64** argument.

Required Privileges

A process must have PSWAPM privilege to call the $LCKPAG_64 service.

Required Quota

None.

Related Services

$LCKPAG, $ULKPAG, $ULKPAG_64

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$_ACCVIO	The **return_va_64** argument or the **return_length_64** argument cannot be written by the caller, or an attempt was made to lock pages by a caller whose access mode is less privileged than the access mode associated with the pages.
SS$_LCKPAGFUL	The system-defined maximum limit on the number of pages that can be locked in memory has been reached.
SS$_LKWSETFUL	The locked working set is full. If any more pages are locked, not enough dynamic pages will be available to continue execution.
SS$_NOPSWAPM	The process does not have the privilege to lock pages in memory.
SS$_PAGOWNVIO	The pages could not be locked because the access mode associated with the call to $LCKPAG_64 was less privileged than the access mode associated with the pages that were to be locked.

$LKWSET
Lock Pages in Working Set

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]

Arguments

inadr

OpenVMS 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 byte-within-page bits are ignored.

VAX

On VAX systems, if the starting and ending virtual addresses are the same, a single page is locked.♦

retadr

OpenVMS 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

OpenVMS 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

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.

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 the value –1.

Global pages with write access cannot be locked into the working set.

Alpha

On Alpha systems, if you are attempting to lock executable code, you should issue multiple $LKWSET calls: one to lock the code pages and others to lock the linkage section references into these pages.♦

Required Access or Privileges

None

Required Quota

None

Related Services

You can unlock pages locked in the working set with the Unlock Page from Working Set ($ULWSET) service.

For more information, see the chapter on memory management in the *OpenVMS Programming Concepts Manual*.

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$_ACCVIO	The input address array cannot be read; the output address array cannot be written; a page in the specified range is inaccessible or nonexistent; or an attempt was made to lock pages by a caller whose access mode is less privileged than the access mode associated with the pages.
SS$_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.

$LKWSET_64 (Alpha Only) Lock Pages in Working Set

On Alpha systems, locks a range of virtual addresses in the working set; if the pages are not already in the working set, the service brings them in and locks them. A page locked in the working set does not become a candidate for replacement.

This service accepts 64-bit addresses.

Format

SYS$LKWSET_64 start_va_64 ,length_64 ,acmode ,return_va_64 ,return_length_64

Arguments

start_va_64
OpenVMS usage: address
type: quadword address
access: read only
mechanism: by value

The starting virtual address of the pages to be locked in the working set. The specified virtual address will be rounded down to a CPU-specific page boundary.

length_64
OpenVMS usage: byte count
type: quadword (unsigned)
access: read only
mechanism: by value

Length of the virtual address space to be locked in the working set. The specified length will be rounded up to a CPU-specific page boundary so that it includes all CPU-specific pages in the requested range.

acmode
OpenVMS usage: access_mode
type: longword (unsigned)
access: read only
mechanism: by value

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

The $PSLDEF macro in STARLET.MLB and the file PSLDEF.H in SYS$STARLET_C.TLB define the following symbols and their values for the four access modes:

Value	Symbolic Name	Access Mode
0	PSL$C_KERNEL	Kernel
1	PSL$C_EXEC	Executive
2	PSL$C_SUPER	Supervisor
3	PSL$C_USER	User

The most privileged access mode used is the access mode of the caller. For the $LKWSET_64 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.

return_va_64

OpenVMS usage:	address
type:	quadword address
access:	write only
mechanism:	by 32-bit or 64-bit reference

The lowest process virtual address of the pages locked in the working set. The **return_va_64** argument is the 32-bit or 64-bit virtual address of a naturally aligned quadword into which the service returns the virtual address.

return_length_64

OpenVMS usage:	byte count
type:	quadword (unsigned)
access:	write only
mechanism:	by 32-bit or 64-bit reference

The length of the virtual address range locked in the working set. The **return_length_64** argument is the 32-bit or 64-bit virtual address of a naturally aligned quadword into which the service returns the length of the virtual address range in bytes.

Description

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.

If the condition value SS$_ACCVIO is returned by this service, a value *cannot* be returned in the memory locations pointed to by the **return_va_64** and **return_length_64** arguments. If a condition value other than SS$_ACCVIO is returned, the returned address and returned length indicate the pages that were successfully locked before the error occurred. If no pages were locked, the **return_va_64** argument will contain the value -1, and a value *cannot* be returned in the memory location pointed to by the **return_length_64** argument.

Global pages with write access cannot be locked into the working set.

Required Privileges

None.

Required Quota

None.

Related Services

$LKWSET, $ULWSET, $ULWSET_64

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$_ACCVIO	The **return_va_64** or **return_length_64** argument cannot be written by the caller, or an attempt was made to lock pages by a caller whose access mode is less privileged than the access mode associated with the pages.
SS$_LKWSETFUL	The locked working set is full. If any more pages are locked, not enough dynamic pages will be available to continue execution.
SS$_NOPRIV	No privilege; global pages with write access cannot be locked into the working set.
SS$_PAGNOTINREG	A page in the specified range is not within the specified region.
SS$_PAGOWNVIO	The pages could not be locked because the access mode associated with the call to $LKWSET_64 was less privileged than the access mode associated with the pages that were to be locked.

$MGBLSC
Map Global Section

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]

Arguments

inadr
OpenVMS 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 to specify which pages are to be mapped; the low-order byte-within-page bits are ignored for this purpose.

The interpretation of the **inadr** argument depends on the setting of SEC$M_EXPREG in the **flags** argument and on whether you are using an Alpha or a VAX system. The two system types are discussed separately in this section.

Alpha

On Alpha systems, if you do *not* set the SEC$M_EXPREG flag, the **inadr** argument specifies the starting and ending virtual addresses of the region to be mapped. Addresses in system space are not allowed. The addresses must be aligned on CPU-specific pages; no rounding to CPU-specific pages occurs. The lower address of the **inadr** argument must be on a CPU-specific page boundary and the higher address of the **inadr** argument must be 1 less than a CPU-specific boundary, thus forming a range from lowest to highest address bytes. You can use the SYI$_PAGE_SIZE item code in the $GETSYI service to set the **inadr** argument to the proper values.

If, on the other hand, you *do* set the SEC$M_EXPREG flag, indicating that the mapping should take place using the first available space in a particular region, the **inadr** argument is used only to indicate the desired region: the program region (P0) or the control region (P1).

Caution

Mapping into the P1 region is generally discouraged, but, if done, must be executed with extreme care. Since the user stack is mapped in P1, it is possible that references to the user stack may inadvertently read or write the pages mapped with $MGBLSC.

When the SEC$M_EXPREG flag is set, the second **inadr** longword is ignored, while bit 30 (the second most significant bit) of the first **inadr** longword is used to determine the region of choice. If the bit is clear, P0 is chosen; if the bit is set, P1 is chosen. On Alpha systems, bit 31 (the most significant bit) of the first **inadr** longword *must be* 0. To ensure compatibility between VAX and Alpha systems when you choose a region, Digital recommends that you specify, for the first **inadr** longword, any virtual address in the desired region.♦

VAX

On VAX systems, if you do *not* set the SEC$M_EXPREG flag, the **inadr** argument specifies the starting and ending virtual addresses of the region to be mapped. Addresses in system space are not allowed. If the starting and ending virtual addresses are the same, a single page is mapped.

Note

If the SEC$M_EXPREG flag is not set, Digital recommends that the **inadr** argument always specify the entire virtual address range, from starting byte address to ending byte address. This ensures compatibility between VAX and Alpha systems.

If, on the other hand, you *do* set the SEC$M_EXPREG flag, indicating that the mapping should take place using the first available space in a particular region, the **inadr** argument is used only to indicate the desired region: the program region (P0) or the control region (P1).

Caution

Mapping into the P1 region is generally discouraged, but, if done, must be executed with extreme care. Since the user stack is mapped in P1, it is possible that references to the user stack may inadvertently read or write the pages mapped with $MGBLSC.

When the SEC$M_EXPREG flag is set, the second **inadr** longword is ignored, while bit 30 (the second most significant bit) of the first **inadr** longword is used to determine the region of choice. If the bit is clear, P0 is chosen; if the bit is set, P1 is chosen. On VAX systems, bit 31 (the most significant bit) of the first **inadr** longword *is ignored.* To ensure compatibility between VAX and Alpha systems when you choose a region, Digital recommends that you specify, for the first **inadr** longword, any virtual address in the desired region.♦

retadr

OpenVMS 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.

Alpha

On Alpha systems, the **retadr** argument returns the starting and ending addresses of the *usable* range of addresses. This may differ from the total amount mapped. The **retadr** argument is required when the **relpag** argument is specified. If the section being mapped does not completely fill the last page used to map the section, the **retadr** argument indicates the highest address that actually maps the section. If the **relpag** argument is used to specify an offset into the section, the **retadr** argument reflects the offset.♦

acmode

OpenVMS 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

OpenVMS 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 Option	Description
SEC$M_WRT	Map the section with read/write access. By default, the section is mapped with read-only access. If SEC$M_WRT is specified and the section is not copy-on-reference, write access is required.
SEC$M_SYSGBL	Map a system global section. By default, the section is a group global section.
SEC$M_EXPREG	Map the section into the first available virtual address range. By default, the section is mapped into the range specified by the **inadr** argument. See the **inadr** argument description for a complete explanation of how to set the SEC$M_EXPREG flag.

gsdnam

OpenVMS 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, the operating system 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

OpenVMS 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 two 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 the value 0 (the default), the version number and match control fields default to the value 0.

relpag

OpenVMS 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.

Alpha

On Alpha systems, the **relpag** argument is interpreted as an index into the section file, measured in pagelets for a file-backed section or CPU-specific pages for a PFN-mapped section.♦

On Alpha and VAX systems, if you do not specify **relpag** or specify it as the value 0 (the default), the global section is mapped beginning with the first virtual block in a file-backed section or the first CPU-specific page in a PFN-mapped section.

Description

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. 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.

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 the value –1.

Required Access or Privileges

Read access is required. If the SEC$M_WRT flag is specified, write access is required.

Required Quota

The working set 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).

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.

Related Services

$ADJSTK, $ADJWSL, $CRETVA, $CRMPSC, $DELTVA, $DGBLSC, $EXPREG, $LCKPAG, $LKWSET, $PURGWS, $SETPRT, $SETSTK, $SETSWM, $ULKPAG, $ULWSET, $UPDSEC, $UPDSECW

For more information, see the chapter on memory management in the *OpenVMS Programming Concepts Manual.*

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$_INSFWSL	The working set limit of the process is not large enough to accommodate the increased virtual address space.
SS$_IVLOGNAM	The global section name has a length of 0 or has more than 43 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$_PAGOWNVIO	A page in the specified input address range is owned by a more privileged access mode.
SS$_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.

$MGBLSC_64 (Alpha Only)
Map Global Disk or Page File Section

On Alpha systems, establishes a correspondence between pages in the virtual address space of the process and the pages occupied by a global disk file or page file section.

This service accepts 64-bit addresses.

Format

SYS$MGBLSC_64 gs_name_64 ,ident_64 ,region_id_64 ,section_offset_64 ,length_64 ,acmode ,flags ,return_va_64 ,return_length_64 [,start_va_64]

Arguments

gs_name_64
OpenVMS usage: section_name
type: character-coded text string
access: read only
mechanism: by 32-bit or 64-bit descriptor–fixed-length string descriptor

Name of the global section. The **gs_name_64** argument is the 32-bit or 64-bit virtual address of a naturally aligned 32-bit or 64-bit string descriptor pointing to this name string.

ident_64
OpenVMS usage: section_id
type: quadword (unsigned)
access: read only
mechanism: by 32-bit or 64-bit reference

Identification value specifying the version number of a global section. The **ident_64** argument is a quadword containing three fields. The **ident_64** argument is the 32-bit or 64-bit virtual address of a naturally aligned quadword that contains the identification value.

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

Value	Symbolic 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 the **ident_64** argument as 0, the version number and match control fields default to 0.

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.

region_id_64

OpenVMS usage:	region identifier
type:	quadword (unsigned)
access:	read only
mechanism:	by 32-bit or 64-bit reference

The region ID associated with the region to map the private page frame section. The file VADEF.H in SYS$STARLET_C.TLB and the $VADEF macro in STARLET.MLB define a symbolic name for each of the three default regions in P0, P1, and P2 space. The following region IDs are defined:

Symbol	Region
VA$C_P0	Program region
VA$C_P1	Control region
VA$C_P2	64-bit program region

Other region IDs, as returned by the $CREATE_REGION_64 service, can be specified.

section_offset_64

OpenVMS usage:	byte offset
type:	quadword (unsigned)
access:	read only
mechanism:	by value

Offset into the global section at which to start mapping into the process's virtual address space.

If a map to a global section is being requested, the **section_offset_64** argument specifies an even multiple of disk blocks. If a map to a global page file section is being requested, the **section_offset_64** argument specifies an even multiple of CPU-specific pages. If zero is specified, the global section is mapped beginning with the first virtual block of the disk file section.

length_64

OpenVMS usage:	byte count
type:	quadword (unsigned)
access:	read only
mechanism:	by value

Length of the desired mapping of the global disk file section in bytes.

If a map to a global section is being requested, the **length_64** argument specifies an even multiple of disk blocks. If a map to a global page file section is being requested, the **length_64** argument specifies an even multiple of CPU-specific pages. If zero is specified, the global section is mapped beginning with the first virtual block of the disk file section. If zero is specified, the size of the disk file is used.

Always refer to the **return_length_64** argument to determine the resulting length of the global disk file section.

acmode

OpenVMS 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 $PSLDEF macro in STARLET.MLB and the file PSLDEF.H in SYS$STARLET_C.TLB define the following symbols and their values for the four access modes:

Value	Symbolic Name	Access Mode
0	PSL$C_KERNEL	Kernel
1	PSL$C_EXEC	Executive
2	PSL$C_SUPER	Supervisor
3	PSL$C_USER	User

The most privileged access mode used is the access mode of the caller. Address space cannot be created within a region that has a create mode associated with it that is more privileged than the caller's mode. The condition value SS$_IVACMODE is returned if the caller is less privileged than the create mode for the region.

flags

OpenVMS 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 in which each bit corresponds to a flag. The $SECDEF macro and the SECDEF.H file define 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 that is valid for the $MGBLSC_64 service.

Flag	Description
SEC$M_GBL	Pages form a global section. By default, this flag is always present in this service and cannot be disabled.
SEC$M_EXPREG	Map the section into the first available space at the current end of the specified region. If this flag is specified, the **start_va_64** argument is not used.
SEC$M_SYSGBL	Map a system global section. By default, the section is a group global section.

Flag	Description
SEC$M_WRT	Map the section with read/write access. By default, the section is mapped with read-only access. If SEC$M_WRT is specified and the section is not copy-on-reference, write access to the section is required.

All other bits in the flags argument are reserved for future use by Digital and should be specified as 0. The condition value SS$_IVSECFLG is returned if any undefined bits are set or if an attempt is made to use the SEC$M_PAGFIL flag, which applies only to the creation of a page-file backed section.

return_va_64
OpenVMS usage: address
type: quadword address
access: write only
mechanism: by 32-bit or 64-bit reference

The process virtual address into which the global disk or page file section was mapped. The **return_va_64** argument is the 32-bit or 64-bit virtual address of a naturally aligned quadword into which the service returns the virtual address.

Upon successful completion of this service, if the **section_offset_64** argument was specified, the virtual address returned in the **return_va_64** argument reflects the offset into the global section mapped such that the virtual address returned cannot be aligned on a CPU-specific page boundary. The virtual address returned will always be on an even virtual disk block boundary.

return_length_64
OpenVMS usage: byte count
type: quadword (unsigned)
access: write only
mechanism: by 32-bit or 64-bit reference

The length of the usable virtual address range mapped. The **return_length_64** argument is the 32-bit or 64-bit virtual address of a naturally aligned quadword into which the service returns the length of the virtual address range in bytes.

Upon successful completion of this service, the value in the **return_length_64** argument might differ from the total amount of virtual address space mapped. The value in the **return_va_64** argument plus the value in the **return_length_64** argument indicates the address of the first byte beyond the end of the mapping of the global disk file section.

If the value in the **section_offset_64** argument plus the value in the **length_64** argument did not specify to map the entire global section, this byte can be located at an even virtual disk block boundary within the last page of the mapping.

If the section being mapped does not completely fill the last page used to represent the global disk file section, this byte can be mapped into your address space; however, it is not backed up by the disk file.

start_va_64
OpenVMS usage: address
type: quadword address
access: read only
mechanism: by value

The starting virtual address to which to map the global section. The specified virtual address must be a CPU-specific page-aligned address. If the flag SEC$M_EXPREG is specified, the **start_va_64** argument must not be specified or must be specified as 0. If SEC$M_EXPREG is set and the **start_va_64** argument is non-zero, the condition value SS$_IVSECFLG is returned.

Always refer to the **return_va_64** and **return_length_64** arguments to determine the usable range of virtual addresses mapped.

Description

The Map Global Disk or Page File Section service establishes a correspondence between pages in the virtual address space of the process and pages occupied by a global disk or page file section. This service adds pages to the virtual address space of the process. The section is mapped from low address to high address.

If the condition value SS$_ACCVIO is returned by this service, a value *cannot* be returned in the memory locations pointed to by the **return_va_64** and **return_length_64** arguments. If a condition value other than SS$_ACCVIO is returned, the returned address and returned length indicate the pages that were successfully mapped before the error occurred. If no pages were mapped, the **return_va_64** argument will contain the value -1, and a value *cannot* be returned in the memory location pointed to by the **return_length_64** argument.

Required Privileges

Read access is required. If the SEC$M_WRT flag is specified and the section is not a copy-on-reference section, write access is required.

Required Quota

The working set quota (WSQUOTA) of the process must be sufficient to accommodate the increased length of the process page table required by the increase in virtual address space.

If the section pages are copy-on-reference, the process must also have sufficient paging file quota (PGFLQUOTA).

Related Services

$CREATE_GFILE, $CREATE_GPFILE, $CREATE_REGION_64, $CRMPSC_GFILE_64, $CRMPSC_GPFILE_64, $DELETE_REGION_64, $DELTVA_64, $LCKPAG_64, $LKWSET_64, $MGBLSC, $MGBLSC_GPFN_64, $PURGE_WS, $ULKPAG_64, $ULWSET_64, $UPDSEC_64, $UPDSEC_64W

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The **gs_name_64** argument cannot be read by the caller, or the **return_va_64** argument or the **return_length_64** argument cannot be written by the caller.
SS$_EXPGFLQUOTA	The process exceeded its paging file quota, creating copy-on-reference pages.
SS$_GBLSEC_MISMATCH	Global section type mismatch. The specified global section was found; however, it is not a global disk or page file section.

SS$_INSFWSL	The working set limit of the process is not large enough to accommodate the increased virtual address space.
SS$_IVACMODE	The caller's mode is less privileged than the create mode associated with the region.
SS$_IVLOGNAM	The specified global section name has a length of 0 or has more than 43 characters.
SS$_IVREGID	An invalid region ID was specified.
SS$_IVSECFLG	An invalid flag, a reserved flag, or an invalid combination of flags was specified.
SS$_IVSECIDCTL	The match control field of the global section identification is invalid.
SS$_LEN_NOTBLKMULT	The **length_64** argument is not a multiple of virtual disk blocks if a map to a global section was requested (SEC$M_PAGFIL is clear in the flags argument).
SS$_LEN_NOTPAGMULT	The **length_64** argument is not a multiple of CPU-specific pages if a map to a global page file section was requested (SEC$M_PAGFIL is set in the flags argument).
SS$_NOSUCHSEC	The specified global section does not exist.
SS$_NOWRTACC	The specified global section is not copy-on-reference and does not allow write access.
SS$_OFF_NOTPAGALGN	The **section_offset_64** argument is not CPU-specific page aligned if a map to a global page file section was requested (SEC$M_PAGFIL is set in the flags argument).
SS$_OFFSET_TOO_BIG	The **section_offset_64** argument specified is beyond the logical end-of-file.
SS$_PAGNOTINREG	A page in the specified range is not within the specified region.
SS$_PROTVIO	The file protection mask specified when the global section was created prohibits the type of access requested by the caller.
SS$_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$_REGISFULL	The specified virtual region is full; no space is available in the region for the pages created to contain the mapped global section.
SS$_TOOMANYLNAM	The logical name translation of the **gs_name_64** argument exceeded the allowed depth of 10.
SS$_VA_NOTPAGALGN	The **start_va_64** argument is not CPU-specific page-aligned.

$MGBLSC_GPFN_64 (Alpha Only) Map Global Page Frame Section

On Alpha systems, establishes a correspondence between pages in the virtual address space of the process and the pages occupied by a global page frame section.

This service accepts 64-bit addresses.

Format

SYS$MGBLSC_GPFN_64 gs_name_64 ,ident_64 ,region_id_64 ,relative_page ,page_count ,acmode ,flags ,return_va_64 ,return_length_64 [,start_va_64]

Arguments

gs_name_64

OpenVMS usage: section_name
type: character-coded text string
access: read only
mechanism: by 32-bit or 64-bit descriptor–fixed-length string descriptor

Name of the global section. The **gs_name** argument is the 32-bit or 64-bit virtual address of a naturally aligned 32-bit or 64-bit descriptor pointing to this name string.

ident_64

OpenVMS usage: section_id
type: quadword (unsigned)
access: read only
mechanism: by 32-bit or 64-bit reference

Identification value specifying the version number of a global section. The **ident_64** argument is a quadword containing three fields. The **ident_64** argument is the 32-bit or 64-bit virtual address of a naturally aligned quadword that contains the identification value.

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

Value	Symbolic 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 the **ident_64** argument as 0, the version number and match control fields default to 0.

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.

region_id_64

OpenVMS usage:	region identifier
type:	quadword (unsigned)
access:	read only
mechanism:	by 32-bit or 64-bit reference

The region ID associated with the region to map the private page frame section. The file VADEF.H in SYS$STARLET_C.TLB and the $VADEF macro in STARLET.MLB define a symbolic name for each of the three default regions in P0, P1, and P2 space. The following region IDs are defined:

Symbol	Region
VA$C_P0	Program region
VA$C_P1	Control region
VA$C_P2	64-bit program region

Other region IDs, as returned by the $CREATE_REGION_64 service, can be specified.

relative_page

OpenVMS usage:	CPU-specific page count
type:	longword (unsigned)
access:	read only
mechanism:	by value

Relative CPU-specific page number within the global section to start mapping.

page_count

OpenVMS usage:	CPU-specific page count
type:	longword (unsigned)
access:	read only
mechanism:	by value

Length of mapping in CPU-specific pages. If zero is specified, the global page frame section is mapped to the end of the section.

acmode

OpenVMS 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. Address space cannot be created within a region that has a create mode associated with it that is more privileged than the caller's mode. The condition value SS$_IVACMODE is returned if the caller is less privileged than the create mode for the region.

flags

OpenVMS 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 in which each bit corresponds to a flag. The $SECDEF macro and the SECDEF.H file define 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 that is valid for the $MGBLSC_GPFN_64 service:

Flag	Description
SEC$M_GBL	Pages form a global section. By default, this flag is always present in this service and cannot be disabled.
SEC$M_EXPREG	Map the section into the first available space at the current end of the specified region. If this flag is specified, the **start_va_64** argument is not used.
SEC$M_PERM	Pages are permanent. By default, this flag is always present in this service and cannot be disabled.
SEC$M_PFNMAP	Pages form a page frame section. By default, this flag is always present in this service and cannot be disabled.
SEC$M_PAGFIL	Pages form a global page-file section. SEC$M_PAGFIL also implies SEC$M_WRT and SEC$M_DZRO.
SEC$M_SYSGBL	Map a system global section. By default, the section is a group global section.
SEC$M_WRT	Map the section with read/write access. By default, the section is mapped with read-only access. If SEC$M_WRT is specified, write access is required.

All other bits in the **flags** argument are reserved for future use by Digital and should be specified as 0. The condition value SS$_IVSECFLG is returned if any undefined bits are set or if an illegal combination of flags is set.

return_va_64

OpenVMS usage: address
type: quadword address
access: write only
mechanism: by 32-bit or 64-bit reference

The lowest process virtual address into which the global page frame section was mapped. The **return_va_64** argument is the 32-bit or 64-bit virtual address of a naturally aligned quadword that contains the virtual address.

return_length_64

OpenVMS usage:	byte count
type:	quadword (unsigned)
access:	write only
mechanism:	by 32-bit or 64-bit reference

The 32-bit or 64-bit virtual address of a naturally aligned quadword into which the $MGBLSC_GPFN_64 service returns the length of the virtual address range in bytes.

start_va_64

OpenVMS usage:	address
type:	quadword address
access:	read only
mechanism:	by value

The starting virtual address to map the global section. The specified virtual address must be a CPU-specified page-aligned address. If the flag SEC$M_EXPREG is specified, the **start_va_64** argument must not be specified or must be specified as 0. If SEC$M_EXPREG is set and the **start_va_64** argument is non-zero, the condition value SS$_IVSECFLG is returned.

Always refer to the **return_va_64** and **return_length_64** arguments to determine the range of virtual addresses mapped.

Description

The Map Global Page Frame Section service establishes a correspondence between pages in the virtual address space of the process and pages occupied by a global page frame section. It adds pages to the virtual address space of the process.

Pages mapped to a global page frame section 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 can result in a machine check if they are in I/O space.

If the condition value SS$_ACCVIO is returned by this service, a value *cannot* be returned in the memory locations pointed to by the **return_va_64** and **return_length_64** arguments.

If a condition value other than SS$_ACCVIO is returned, the returned address and returned length indicate the pages that were successfully mapped before the error occurred. If no pages were mapped, the **return_va_64** argument will contain the value -1, and a value *cannot* be returned in the memory location pointed to by the **return_length_64** argument.

Required Privileges

Read access is required. If the SEC$M_WRT flag is specified, write access is required.

Required Quota

The working set quota (WSQUOTA) of the process must be sufficient to accommodate the increased length of the process page table required by the increase in virtual address space.

The page file quota (PAGFLQUOTA) of the process must be sufficient to accommodate the increased number of process page tables required by the increase in virtual address space. (Note that this service can return the SS$_EXPGFLQUOTA.)

Related Services

$CREATE_GPFN, $CREATE_REGION_64, $CRMPSC_GPFN_64, $DELETE_REGION_64, $DELTVA_64, $MGBLSC, $MGBLSC_64

Condition Values Returned

SS$_NORMAL	The service completed successfully
SS$_ACCVIO	The **gs_name_64** argument cannot be read by the caller, or the **return_va_64** or **return_length_64** argument cannot be written by the caller.
SS$_GBLSEC_MISMATCH	Global section type mismatch. The specified global section was found; however, it is not a global page frame section.
SS$_ILLRELPAG	The specified relative page argument is either larger than the highest page number within the section or is not a valid 32-bit physical page frame number.
SS$_INSFWSL	The working set limit of the process is not large enough to accommodate the increased virtual address space.
SS$_IVACMODE	The caller's mode is less privileged than the create mode associated with the region.
SS$_IVLOGNAM	The specified global section name has a length of 0 or has more than 43 characters.
SS$_IVREGID	Invalid region ID specified.
SS$_IVSECFLG	An invalid flag, a reserved flag, or an invalid combination of flags was specified.
SS$_IVSECIDCTL	The match control field of the global section identification is invalid.
SS$_NOSUCHSEC	The specified global section does not exist.
SS$_NOWRTACC	The specified global section is not copy-on-reference and does not allow write access.
SS$_PROTVIO	The file protection mask specified when the global section was created prohibits the type of access requested by the caller.
SS$_PAGNOTINREG	A page in the specified range is not within the specified region.
SS$_PAGOWNVIO	A page in the specified range already exists and cannot be deleted because it is owned by a more privileged access mode than that of the caller.
SS$_REGISFULL	The specified virtual region is full; no space is available in the region for the pages created to contain the mapped global section.

SS$_TOOMANYLNAM	The logical name translation of the **gs_name_64** argument exceeded the allowed depth of 10.
SS$_VA_NOTPAGALGN	The **start_va_64** argument is not CPU-specific page-aligned.

$MOD_HOLDER
Modify Holder Record in Rights Database

Modifies the specified holder record of the target identifier in the rights database.

Format

SYS$MOD_HOLDER id ,holder ,[set_attrib] ,[clr_attrib]

Arguments

id

OpenVMS 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

OpenVMS 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 0 in the second longword.

set_attrib

OpenVMS 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$V_DYNAMIC	Allows holders of the identifier to remove it from or add it to the process rights list by using the DCL command SET RIGHTS_LIST.
KGB$V_NOACCESS	Makes any access rights of the identifier null and void. This attribute is intended as a modifier for a resource identifier or the Subsystem attribute.
KGB$V_RESOURCE	Allows the holder to charge resources, such as disk blocks, to the identifier.
KGB$V_SUBSYSTEM	Allows holders of the identifier to create and maintain protected subsystems by assigning the Subsystem ACE to the application images in the subsystem.

clr_attrib

OpenVMS 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$V_DYNAMIC	Allows holders of the identifier to remove it from or add it to the process rights list by using the DCL command SET RIGHTS_LIST.
KGB$V_NOACCESS	Makes any access rights of the identifier null and void. This attribute is intended as a modifier for a resource identifier or the Subsystem attribute.
KGB$V_RESOURCE	Allows the holder to charge resources, such as disk blocks, to the identifier.
KGB$V_SUBSYSTEM	Allows holders of the identifier to create and maintain protected subsystems by assigning the Subsystem ACE to the application images in the subsystem.

Description

The Modify Holder Record in Rights Database service modifies the specified holder record in the rights database. Identifier attributes can be added or removed.

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.

Required Access or Privileges

Write access to the rights database is required.

Required Quota

None

Related Services

$ADD_HOLDER, $ADD_IDENT, $ASCTOID, $CHANGE_ACL, $CREATE_RDB, $FIND_HELD, $FIND_HOLDER, $FINISH_RDB, $GRANTID, $IDTOASC, $MOD_IDENT, $REM_HOLDER, $REM_IDENT, $REVOKID

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 OpenVMS RMS, this service can also return RMS status codes associated with operations on indexed files. For descriptions of these status codes, refer to the *OpenVMS Record Management Services Reference Manual*.

$MOD_IDENT
Modify Identifier in Rights Database

Modifies the specified identifier record in the rights database.

Format

SYS$MOD_IDENT id ,[set_attrib] ,[clr_attrib] ,[new_name] ,[new_value]

Arguments

id
OpenVMS 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
OpenVMS 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$V_DYNAMIC	Allows holders of the identifier to remove it from or add it to the process rights list by using the DCL command SET_ RIGHTS_LIST.
KGB$V_HOLDER_HIDDEN	Prevents someone from getting a list of users who hold an identifier, unless they own the identifier themselves.
KGB$V_NAME_HIDDEN	Allows holders of an identifier to have it translated—either from binary to ASCII or vice versa—but prevents unauthorized users from translating the identifier.

Bit Position	Meaning When Set
KGB$V_NOACCESS	Makes any access rights of the identifier null and void. This attribute is intended as a modifier for a resource identifier or the Subsystem attribute.
KGB$V_RESOURCE	Allows holders of an identifier to charge disk space to the identifier. It is used only for file objects.
KGB$V_SUBSYSTEM	Allows holders of the identifier to create and maintain protected subsystems by assigning the Subsystem ACE to the application images in the subsystem.

clr_attrib

OpenVMS 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$V_DYNAMIC	Allows holders of the identifier to remove it from or add it to the process rights list by using the DCL command SET_ RIGHTS_LIST.
KGB$V_HOLDER_HIDDEN	Prevents someone from getting a list of users who hold an identifier, unless they own the identifier themselves.
KGB$V_NAME_HIDDEN	Allows holders of an identifier to have it translated—either from binary to ASCII or vice versa—but prevents unauthorized users from translating the identifier.
KGB$V_NOACCESS	Makes any access rights of the identifier null and void. This attribute is intended as a modifier for a resource identifier or the Subsystem attribute.
KGB$V_RESOURCE	Allows holders of an identifier to charge disk space to the identifier. It is used only for file objects.

Bit Position	Meaning When Set
KGB$V_SUBSYSTEM	Allows holders of the identifier to create and maintain protected subsystems by assigning the Subsystem ACE to the application images in the subsystem.

new_name

OpenVMS 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 one nonnumeric character. Any lowercase characters specified are automatically converted to uppercase.

new_value

OpenVMS 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. Identifier attributes can be added or removed. The identifier name or value can be changed. 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.

Required Access or Privileges

Write access to the rights database is required.

Required Quota

None

Related Services

$ADD_HOLDER, $ADD_IDENT, $ASCTOID, $CREATE_RDB, $FIND_HELD, $FIND_HOLDER, $FINISH_RDB, $GRANTID, $IDTOASC, $MOD_HOLDER, $REM_HOLDER, $REM_IDENT, $REVOKID

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_NOSUCHID	The specified identifier does not exist in the rights database.
SS$_BADPARAM	The specified attributes contain invalid attribute flags.
SS$_DUPIDENT	The specified identifier value already exists.
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 OpenVMS RMS, this service can also return RMS status codes associated with operations on indexed files. For descriptions of these status codes, refer to the *OpenVMS Record Management Services Reference Manual.*

$MOUNT
Mount Volume

Mounts a tape, disk volume, or volume set and specifies options for the mount operation.

Format

SYS$MOUNT itmlst

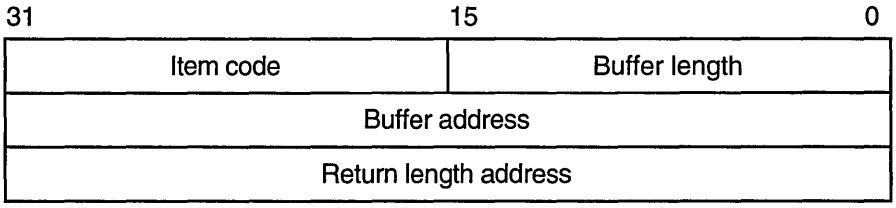
Argument

itmlst
OpenVMS 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 value of 0.

The following diagram depicts the format of a single item descriptor.



ZK-5186A-GE

The following table defines the item descriptor fields.

Descriptor Field	Definition
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 the 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.

Descriptor Field	Definition
Buffer address	A longword containing the address of the buffer that supplies information to $MOUNT.
Return length address	This field is not used.

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$_BLOCKSIZE item code specifies the default block size for tape volumes. The buffer must contain a longword integer value in the range 20 to 65,532 bytes for OpenVMS RMS operations or 10 to 65,534 bytes for operations that do not use RMS. The MNT$_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 can 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 can 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 2-longword bit vector wherein each bit specifies an option for the mount operation. The buffer must contain a quadword, which is the bit vector.

The $MNTDEF macro defines symbolic names for each option (bit) in the bit vector. You construct the bit vector by specifying the symbolic names for the desired options in a logical OR operation. In the first longword you logically OR the MNT$M_ mask bits, and in the second longword you logically OR the MNT2$M_ mask bits. The following table describes the symbolic names for each option. The MNT2$M_ options are at the end of the table.

Option	Description
MNT$M_CLUSTER	The volume is to be mounted for clusterwide access; that is, every VMScluster node can access the volume. $MOUNT mounts the volume first on the caller's node and then on every other node in the existing cluster. Only system or group volumes can be mounted clusterwide. If you do not specify MNT$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 cluster. MNT$M_CLUSTER applies only to disks.
MNT$M_FOREIGN	The volume is to be mounted as a foreign volume; a foreign volume is not Files–11 structured. If you specify MNT$M_FOREIGN, the following item codes can each appear in the item list only once: the caller must either own the volume or have VOLPRO privilege.
MNT$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.

Option	Description
MNT$M_INCLUDE	Automatically reconstructs a shadow set to the state it was in before the shadow set was dissolved (due to dismounting or system failure). Use this option to mount a shadow set or a volume set of shadow sets. You must specify the exact name of the original virtual unit and the device name of at least one of the shadow set members. The shadowing software reads the shadow set membership information from the named device to determine the membership of the original shadow set. You can include the MNT$M_INCLUDE option in executable images to have a shadow set reconstructed. Using MNT$M_INCLUDE prevents your having to manually reinstate shadow sets after they have been dismounted. If you do not select this option, $MOUNT does not automatically reconstruct the former shadow set.
MNT$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. 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$M_MESSAGE	Messages will be sent to the caller's SYS$OUTPUT device.

Option	Description
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. MNT$M_MULTI_VOL can only be used with the MNT$M_FOREIGN option. Digital recommends the use of this qualifier only when it is not possible to alter the utility to explicitly perform MOUNT and DISMOUNT operations on each reel in the set.
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_NOAUTO	Automatic volume labeling (AVL) and automatic volume recognition (AVR) are to be disabled. If MNT$M_NOAUTO is specified, the operator must enter commands from the console to process each additional volume in a volume set. When a volume is finished processing, the operator specifies the drive on which the next volume is loaded and the label name of the next volume. You might want to use MNT$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$M_NOCACHE	All caching associated with the volume is turned off. Specifying MNT$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_NOCOPY	Disables full copy operations on all physical devices being mounted or added to a shadow set. This option provides you with the opportunity to confirm the states of all of the devices or members of a shadow set before proceeding with any full copy operation. This prevents any accidental loss of data that could occur if an unintended device is added to the shadow set. If you do not select this option, $MOUNT automatically overwrites the data on shadow set members that are not current. When you select this option, a $MOUNT operation fails if any of the specified potential shadow set members require full copy operations.
MNT$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.

Option	Description
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$M_NOHDR3 applies only to tapes.
MNT$M_NOLABEL	The volume is to be mounted as a foreign volume; a foreign volume is not Files–11 structured. If you specify MNT$M_NOLABEL, the following item codes can each appear in the item list only once: MNT$_DEVNAM, MNT$_VOLNAM, and MNT$_LOGNAM. To specify MNT$M_NOLABEL, the caller must either own the volume or have VOLPRO privilege.
MNT$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_NOREBUILD	The volume to be mounted should be returned to active use immediately, without performing a rebuild operation. This flag defers the disk rebuild operation, so that the volume to be mounted is returned to active use immediately. A rebuild operation can consume a considerable amount of time, depending on the number of files on the volume and on the number of different file owners (if quotas are in use). The volume can be rebuilt later with the DCL command SET VOLUME/REBUILD to recover the free space; for more information, see the *OpenVMS DCL Dictionary*. If a disk volume is improperly dismounted, for example, during a system failure, it must be rebuilt to recover any caching limits that were enabled on the volume at the time of the dismount. By default, $MOUNT attempts to rebuild. When mounting a volume set, you must mount all members of the set to reclaim all available free space. MNT$M_NOREBUILD applies only to disks.
MNT$M_NOUNLOAD	The volume to be mounted is not to be unloaded when it is dismounted. Specifying MNT$M_NOUNLOAD causes the volume to remain loaded when it is dismounted unless the dismount explicitly requests that the volume be unloaded.
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.

Option	Description
MNT$M_OVR_ACCESS	If the installation allows, this option overrides any character in the accessibility field of the volume. The necessity of this option is defined by the installation. That is, each installation has the option of specifying a routine that the magnetic tape file system will use to process this field. By default, the operating system provides a routine that checks this field in the following manner: • If the magnetic tape was created on a version of the operating system that conforms to Version 3 of ANSI, then you must use this option to override any character other than an ASCII space. • If a protection is specified and that magnetic tape conforms to an ANSI standard that is higher than Version 3, then you must use this option to override any character other than an ASCII 1. To specify MNT$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 can 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$M_OVR_LOCK	The software write lock that occurs when a volume has a corrupted storage bit mask can be overridden.
MNT$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_OVR_SHAMEM	Allows you to mount former shadow set members outside of the shadow set. If you do not specify this option, $MOUNT automatically mounts the volume write-locked to prevent accidental deletion of data. To specify this option, you must either own the volume or have VOLPRO privilege. When you use this option, the shadow set generation number is erased from the volume. If you then remount the volume in the former shadow set, $MOUNT considers it an unrelated volume and marks it for a full copy operation.

Option	Description
MNT$M_OVR_VOLO	The volume label's owner identifier field is not to be processed. $MOUNT reads volume owner and protection information from the volume owner field of the volume labels. The operating system requires that you specify MNT$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 OpenVMS; (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$M_READCHECK	Read checks are to be performed following all read operations.
MNT$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, $MOUNT will deallocate the device so that other users can access the volume.
MNT$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$M_TAPE_DATA_WRITE	Enables the tape controller's write cache for this device. Enabling the write cache improves data throughput for write operations. By default, the tape controller's write cache is disabled for the device. This option applies only to tape systems that support a write cache.
MNT$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.
MNT2$M_CDROM	Mounts a volume assuming the media to be ISO 9660 (or High Sierra) formatted.
MNT2$M_COMPACTION	Enables data compaction for those magnetic tapes that support data compaction (TA90, TA91, and others).
MNT2$M_DISKQ	Controls whether quotas are to be enforced on the specified disk volume.
MNT2$_DSI	Enables XAR permissions Owner and Group for XARs containing Digital System Identifiers (DSI). For more information, see the *OpenVMS Record Management Services Reference Manual.*

Option	Description
MNT2$_INCLUDE	Automatically reconstructs a former shadow set to the way it was before the shadow set was dissolved. Applicable only if you have the volume shadowing option. For more information, see *Volume Shadowing for OpenVMS*.
MNT2$M_ NOCOMPACTION	Forces the density to no compaction for those magnetic tapes that support data compaction (TA90, TA91, and others).
MNT2$_OVR_LIMITED_ SEARCH	For disk type devices that do not provide for bad-block revectoring, it is possible that the Files–11 homeblock has been placed numerous I/Os from the start of the volume. To decrease the failover time when accessing media which does not contain a valid Files–11 homeblock, a limited-search algorithm was implemented. This switch overrides the limited-search algorithm so that the entire volume will be searched for a valid Files–11 homeblock.
MNT2$M_OVR_NOFE	This bit mask is set to override those SCSI devices that do not support forced error functionality. By overriding those SCSI devices not supporting forced error capabilities, MNT2$M_OVR_NOFE enables those devices to be mounted. Otherwise, the shadowing code would report to $MOUNT that the device does not support forced error, and the device would not be mounted.
MNT2$_OVR_SECURITY	Enables you to continue mounting a volume if an error is returned because the volume has an invalid SECURITY.SYS file. You must have the VOLPRO privilege or own the volume to use this keyword.
MNT2$M_SUBSYSTEM	Enables the processing of protected subsystem identifiers on the volume. By default, subsystem identifiers are ignored on all but the system disk. Requires SECURITY privilege.
MNT2$M_XAR	Enables enforcement of the extended record attribute (XAR) access controls. For more information about XAR, see the *OpenVMS System Manager's Manual*.

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 On-Disk 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 On-Disk 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.
filespec	Specifies that $MOUNT create a new process to execute the ACP image with the file specification *filespec*. Wildcard characters are not allowed in the file specification. The file must be in the disk and directory specified by the logical name SYS$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$_SHAMEM item code specifies the name of a physical device to be mounted into a shadow set. The MNT$_SHAMEM descriptor is a 1- to 64-character string containing the device name. The string can be a physical device name or a logical name; if it is a logical name, it must translate to a physical device name. An item list must contain at least one item descriptor specifying

a member; this item descriptor must appear after the MNT$_SHANAM item descriptor.

Volume Shadowing for OpenVMS automatically performs a copy or a merge operation, if necessary, when it mounts the disk into the shadow set.

MNT$_SHAMEM_COPY
The MNT$_SHAMEM_COPY item code specifies the name of a device that will be the target of a copy operation when it is mounted into a shadow set. The buffer is a 1- to 64-character string containing the device name. The string can be a physical device name or a logical name; if it is a logical name, it must translate to a physical device name. The device will become a member of the shadow set represented by the virtual unit name specified in the MNT$_SHANAM item descriptor.

Volume Shadowing for OpenVMS automatically performs a copy operation on the device specified with the MNT$_SHAMEM_COPY item code.

Caution

When adding a device with the MNT$_SHAMEM_COPY item code, specify only those members that require a copy operation.

MNT$_SHAMEM_MGCOPY
The MNT$_SHAMEM_MGCOPY item code specifies the name of a device that will be a merge member of the shadow set. The device will merge with members of the shadow set represented by the virtual unit specified in the MNT$_SHANAM item descriptor. The buffer is a 1- to 64-character string containing the device name. The string can contain a physical device name or a logical name; if it is a logical name, it must translate to a physical device name.

Volume Shadowing for OpenVMS automatically performs a merge operation on the device specified with the MNT$_SHAMEM_MGCOPY item code. Therefore, you should use MNT$_SHAMEM_MGCOPY when the information on the disks is correct except for possible data inconsistencies.

MNT$_SHANAM
The MNT$_SHANAM item code specifies the name of the virtual unit to be mounted. The buffer is a 1- to 64-character string containing the device name. The virtual unit name may be a logical name; if it is a logical name, it must translate to a virtual unit name.

Because every shadow set is represented by a virtual unit, you must include at least one MNT$_SHANAM item descriptor in the item list that you pass to $MOUNT to create and mount the shadow set. If you are mounting a volume set containing more than one shadow set, you must include one MNT$_SHANAM item descriptor for each virtual unit included in the volume set.

The relative position of the item descriptors in the item list determines the membership of the shadow set. That is, it indicates which members should be bound to a specific virtual unit to form the shadow set. You must first specify the virtual unit by using the MNT$_SHANAM item code. Then, you can specify any number of members that are to be represented by that virtual unit by using one of the following item codes: MNT$_SHAMEM, MNT$_SHAMEM_COPY, or MNT$_SHAMEM_MGCOPY. If you specify one shadow set and want to specify a second, specify a second virtual unit item descriptor. The members you

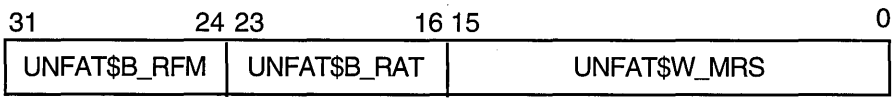
specify subsequently are bound to the shadow set represented by the virtual unit specified in the second virtual unit item descriptor.

MNT$_UCS
The MNT$_UCS item code specifies a descriptor containing a Universal Character Sequence (UCS) defined by ISO 2022 and used when mounting an ISO 9660 CD–ROM. For more information, see the *OpenVMS System Manager's Manual*.

MNT$_UNDEFINED_FAT
The MNT$_UNDEFINED_FAT item code specifies the default file attributes to be used for the records on ISO 9660 media for which no record format has been specified.

The buffer contains a 32-bit structure that defines a file's record format, record attributes, and maximum record size.

The following diagram depicts the structure of the Undefined File Attributes buffer.



ZK–6644A–GE

The following table defines the buffer fields.

Buffer Field	Definition
UNFAT$W_MRS	Maximum record size; specifies the maximum record size for all records in a file: 0 to 32767. Applies only to FIXED or STREAM formats.
UNFAT$B_RAT	Record attributes; specifies the attributes for all records in a file: NONE, CR, FTN, PRN, NOBKS. Applies only to non-STREAM record formats.
UNFAT$B_RFM	Record format; specifies the format for all records in a file: FIXED, VARIABLE, STREAM, STREAM_LF, STREAM_CR, LSB_VARIABLE, or MST_VARIABLE.

MNT$_VOLNAM
The MNT$_VOLNAM item code specifies the name of the volume to be mounted on the device. The number of characters allowed in a volume name depends on the type of device, as follows:

Device Type	Number of Characters in Label
Magnetic tape	0–6
Files–11 disk	1–12
ISO 9660 disk	1–32

The operating system requires disk volume labels to be unique in the first 12 characters within a given domain.

The MNT$_VOLNAM item code can 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 can 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.

An ISO 9660 volume set name can be from 1 to 128 characters in length.

Volume set names must be unique in the first 12 characters. In addition, if the first 12 characters of the volume set name are the same as the first 12 characters of any volume label, a lock manager deadlock will occur. To avoid this problem, you must override either the volume label (by using the MNT$_VOLNAM item code) or the volume set name (by using the MNT$_VOLSET item code).

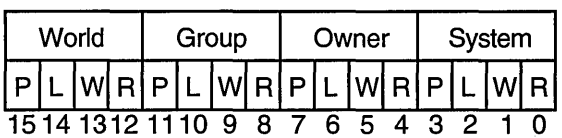
When you specify MNT$_VOLSET, volumes specified by the MNT$_VOLNAM item code are bound into a new volume set or added to an existing volume set, depending on whether the name specified by MNT$_VOLSET is a new or already existing name.

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 category of users. Cleared bits grant access; set bits deny access. The following diagram depicts the structure of the protection mask.



ZK–1715–GE

If you do not specify MNT$_VPROT or specify it as the value 0, the volume receives the protection that it was assigned when it was initialized. To specify MNT$_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

The Mount Volume service mounts a tape, disk volume, or volume set and specifies options for the mount operation.

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 can 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.

Required Access or Privileges

To mount a volume on a device, you must have read, write, or control access to that device.

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.

SECURITY privilege is required to enable protected subsystems.

Required Quota

None

Related Services

$ALLOC, $ASSIGN, $BRKTHRU, $BRKTHRUW, $CANCEL, $CREMBX, $DALLOC, $DASSGN, $DELMBX, $DEVICE_SCAN, $DISMOU, $GETDVI, $GETDVIW, $GETMSG, $GETQUI, $GETQUIW, $INIT_VOL, $PUTMSG, $QIO, $QIOW, $SNDERR, $SNDJBC, $SNDJBCW, $SNDOPR

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$_BADPARAM	A buffer length of 0 was specified with a nonzero item code; an illegal item code was specified; or no device was specified.
SS$_NOGRPNAM	The caller does not have GRPNAM privilege.
SS$_NOHOMEBLK	Files–11 home block not found on volume.
SS$_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.
SS$_NOSYSNAM	The caller does not have SYSNAM privilege.

The $MOUNT service can also return a condition value that is specific to the Mount utility. The symbolic definition macro $MOUNDEF defines these condition values.

$MTACCESS
Magnetic Tape Accessibility

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.

Format

SYS$MTACCESS lblnam ,[uic] ,[std_version] ,[access_char] ,[access_spec] ,type

Arguments

lblnam
OpenVMS 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, the value of this field is 0. The type of label passed is determined by **type**.

uic
OpenVMS 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
OpenVMS 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
OpenVMS 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
OpenVMS 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

OpenVMS 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 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 ANSI labeled magnetic tapes. The installation can override the default routine by providing an MTACCESS.EXE 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 the **access_char** argument 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 protection is not checked. If the field contains an ASCII 1, and the VOL1 Implementation Identifier field contains the system code, R0 is set to SS$_NORMAL. In that case, the 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 accessibility checking and allows the magnetic tape file system to check protection.

The following table summarizes the results of label input check.

Contents of R0	Result
SS$_NORMAL	Check the protection on the magnetic tape.
0	Give the user full access. protection is not checked.
SS$_FILACCERR	Check for explicit override, then check 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 tape file system handles these cases.

The magnetic tape file system calls $MTACCESS to process the accessibility field in the VOL1 and HDR1 labels. After a call to the system service, the magnetic tape file system checks that the installation did not move the magnetic tape. If the magnetic tape was moved, the magnetic tape file system completes the current operation with an SS$_TAPEPOSLOST error. Finally, it processes the remainder of the label according to the status returned by $MTACCESS.

Required Access or Privileges

Because accessibility is an installation-provided routine, the operating system 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.

Required Quota

None

Related Services

$ADD_HOLDER, $ADD_IDENT, $ASCTOID, $CHANGE_ACL, $CHECK_ACCESS, $CHKPRO, $CREATE_RDB, $ERAPAT, $FIND_HELD, $FIND_HOLDER, $FINISH_RDB, $FORMAT_ACL, $FORMAT_AUDIT, $GRANTID, $HASH_PASSWORD, $IDTOASC, $MOD_HOLDER, $MOD_IDENT, $PARSE_ACL, $REM_HOLDER, $REM_IDENT, $REVOKID

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

Converts an absolute or delta time from 64-bit system time format to binary integer date and time values.

Format

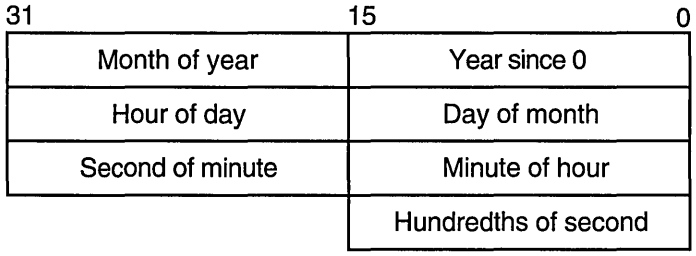
SYS$NUMTIM timbuf ,[timadr]

Arguments

timbuf

OpenVMS 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–GE

If the **timadr** argument specifies a delta time, $NUMTIM returns the value 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

OpenVMS usage:	date_time
type:	quadword
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 the value 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.

$NUMUTC
Convert UTC Time to Numeric Components

Converts an absolute 128-bit binary time into its numeric components. The numeric components are returned in local time.

Format

SYS$NUMUTC timbuf ,[utcadr]

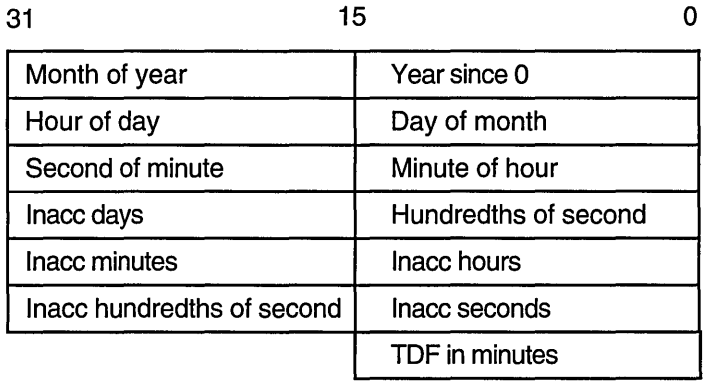
Arguments

timbuf

OpenVMS usage:	vector_word_unsigned
type:	word
access:	write only
mechanism:	by reference

Buffer into which $NUMUTC writes the converted date and time. The **timbuf** argument is the address of a 13-word structure containing time, inaccuracy of time, and time differential factor. The time differential factor encoded in the 128-bit buffer is used to convert the UTC to its numerical components. Negative values in the inaccuracy field indicate an infinite inaccuracy.

The following diagram depicts the fields in this structure.



ZK-4631A

utcadr

OpenVMS usage:	coordinated universal time
type:	utc_date_time
access:	read only
mechanism:	by reference

The 128-bit UTC time value to be converted.

The **utcadr** argument is optional; if it is not used, $NUMUTC will use the current time.

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_INVTIME	The 128-bit UTC time is not valid.

$PARSE_ACL
Parse Access Control List Entry

Parses the specified text string and converts it to the binary representation for an access control entry (ACE).

Format

SYS$PARSE_ACL aclstr ,aclent ,[errpos] ,[accnam] ,[nullarg]

Arguments

aclstr

OpenVMS 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

OpenVMS 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 $FORMAT_ACL.

errpos

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

Number of characters from **aclstr** processed by $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

OpenVMS 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.

You can call LIB$GET_ACCNAM to retrieve the access name table for the class of object whose ACL is to be formatted. If you omit **accnam**, the following names are used.

Bit	Name
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

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

Placeholding argument reserved to Digital.

Description

The Parse Access Control List Entry service parses the specified text string and converts it to the binary representation for an access control entry (ACE).

Required Access or Privileges

None

Required Quota

None

Related Services

$ADD_HOLDER, $ADD_IDENT, $ASCTOID, $CHANGE_ACL, $CHECK_ACCESS, $CHKPRO, $CREATE_RDB, $ERAPAT, $FIND_HELD, $FIND_HOLDER, $FINISH_RDB, $FORMAT_ACL, $FORMAT_AUDIT, $GRANTID, $HASH_PASSWORD, $IDTOASC, $MOD_HOLDER, $MOD_IDENT, $MTACCESS, $REM_HOLDER, $REM_IDENT, $REVOKID

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The string or its descriptor cannot be read by the caller; the buffer descriptor cannot be read by the caller; the buffer cannot be written by the caller; or the buffer is too small to hold the ACL entry.
SS$_IVACL	The format of the access control list entry is not valid.
SS$_NOSUCHID	The specified identifier does not exist in the rights database.

$PERM_DIS_ALIGN_FAULT_REPORT (Alpha Only) Disable Alignment Fault Reporting

On Alpha systems, disables user process alignment fault reporting.

Format

SYS$PERM_DIS_ALIGN_FAULT_REPORT

Description

The Disable Alignment Fault Reporting service disables user process alignment fault reporting.

See the description of the $PERM_REPORT_ALIGN_FAULT service for an example of a program that can be used to enable and disable user process alignment fault reporting.

Required Access or Privileges

None

Required Quota

None

Related Services

$GET_ALIGN_FAULT_DATA, $GET_SYS_ALIGN_FAULT_DATA, $INIT_SYS_ALIGN_FAULT_REPORT, $PERM_REPORT_ALIGN_FAULT, $START_ALIGN_FAULT_REPORT, $STOP_ALIGN_FAULT_REPORT, $STOP_SYS_ALIGN_FAULT_REPORT

Condition Values Returned

SS$_NORMAL	The service completed successfully.

$PERM_REPORT_ALIGN_FAULT (Alpha Only) Report Alignment Fault

On Alpha systems, initializes user process alignment fault reporting.

Format

SYS$PERM_REPORT_ALIGN_FAULT

Description

The Report Alignment Fault service allows the user to permanently enable user process alignment fault reporting for all subsequent images.

This service reports alignment faults only in exception mode. For more information about reporting modes, see the $START_ALIGN_FAULT_REPORT service.

Image alignment fault reporting takes precedence over process alignment fault reporting. That is, if both image and process alignment fault reporting are enabled, faults are reported to the image first.

Required Access or Privileges

None

Required Quota

None

Related Services

$GET_ALIGN_FAULT_DATA, $GET_SYS_ALIGN_FAULT_DATA, $INIT_SYS_ALIGN_FAULT_REPORT, $PERM_DIS_ALIGN_FAULT_REPORT, $START_ALIGN_FAULT_REPORT, $STOP_ALIGN_FAULT_REPORT, $STOP_SYS_ALIGN_FAULT_REPORT

Condition Values Returned

SS$_NORMAL	The service completed successfully.

Example

```
/*****************************************************************/
/*                                                               */
/* SET_ALIGN_REPORT.C                                            */
/*                                                               */
/*  This program can be used to permanently turn on and off      */
/*  alignment fault reporting for a process. After creating the  */
/*  executable, do:                                              */
/*                                                               */
/*        $ align :== $dir:set_align_report.exe                  */
/*        $ align on                                             */
/*        $ run program        ! will generate align faults on screen */
/*        $ align off                                            */
/*        $ run program        ! will not generate align faults  */
/*                                                               */
/*****************************************************************/

#include <stdio>
#include <ctype>
#include <ssdef>
```

```
/*                alignment fault reporting system services          */
extern          sys$perm_report_align_fault(),
                sys$perm_dis_align_fault_report();

main(argc, argv)
  int                 argc;
  char                *argv[];
{
  int                 status;

  /* check arguments */
  if (argc < 2) {
    printf ("Insufficient arguments\n");
    return (40);
  }

  /* check if the argument is on or off */
  if ((strcmp ("ON", argv[1]) == 0) || (strcmp ("on", argv[1]) == 0))
    /* on, turn alignment fault reporting on for this process */
    status = sys$perm_report_align_fault ();

  else if ((strcmp ("OFF", argv[1]) == 0) || (strcmp ("off", argv[1]) == 0))
    /* off, turn alignment fault reporting off for this process */
    status = sys$perm_dis_align_fault_report ();

  else
    return (SS$_BADPARAM);

  /* return status */
  return (status);
}
```

This example shows a program that can be used to enable and disable alignment fault reporting for a process.

$PROCESS_AFFINITY (Alpha Only) Modify Process Affinity

On Alpha systems, allows modification of the CPU affinity set for a specified kernel thread.

This service accepts 64-bit addresses.

Format

SYS$PROCESS_AFFINITY [pidadr] [,prcnam] [,select_mask] [,modify_mask] [,prev_mask] [,flags]

Arguments

pidadr

OpenVMS usage:	process_id
type:	longword (unsigned)
access:	read only
mechanism:	by 32-bit or 64-bit reference

Process identification (PID) of a kernel thread whose affinity mask is to be modified or returned. The **pidadr** argument is the 32-bit or 64-bit address of a longword that contains the PID.

Process selection is made through a combination of the **pidadr** and **prcnam** arguments. If neither are specified or if both have a zero value, the service operations are made to the user affinity mask of the initial thread of the current calling process. The **pidadr** argument takes precedence over the **prcnam** argument in any circumstances where both are supplied in the service call.

prcnam

OpenVMS usage:	process_name
type:	character-coded text string
access:	read only
mechanism:	by 32-bit or 64-bit descriptor–fixed length string descriptor

Process name of the process whose affinity mask is to be modified or returned. The **prcnam** argument is the 32-bit or 64-bit address of a character string descriptor pointing to the process name string. A process can be identified with a 1- to 15-character string. The service operations are made to the user affinity mask of the initial thread of the specified process.

If **pidadr** and **prcnam** are both specified, then **pidadr** is modified or returned and **prcnam** is ignored. If neither argument is specified, then the context of the initial thread of the calling process is modified or returned.

select_mask

OpenVMS usage:	mask_quadword
type:	quadword (unsigned)
access:	read only
mechanism:	by 32-bit or 64-bit reference

Mask specifying which bits of the specified process' affinity mask are to be modified. The **select_mask** argument is the 32-bit or 64-bit address of a quadword bit vector wherein a bit, when set, specifies that the corresponding CPU position in the mask is to be modified.

The individual CPU bits in **select_mask** can be referenced by their symbolic name constants, CAP$M_CPU0 through CAP$M_CPU31. These constants (zero-relative to match system CPU IDs) specify the position in the mask quadword that correspond to the bit name. Multiple CPUs can be selected by ORing together the appropriate bits.

modify_mask

OpenVMS usage:	mask_quadword
type:	quadword (unsigned)
access:	read only
mechanism:	by 32-bit or 64-bit reference

Mask specifying the settings for those explicit affinities selected in the **select_mask** argument. The **modify_mask** argument is the 32-bit or 64-bit address of a quadword bit vector wherein a bit, when set, specifies that the corresponding CPU is to be added to the specified process affinity set; when clear, the corresponding CPU is to be removed from the specified process affinity set.

The bit constants CAP$M_CPU0 through CAP$M_CPU31 can be used to modify the appropriate bit position in the quadword pointed to by **modify_mask**. Multiple CPUs can be added to the affinity set by ORing together the appropriate bits.

To add a specific CPU to the affinity mask set, that bit position must be set in both **select_mask** and **modify_mask**. To remove a specific CPU from the affinity mask set, that bit position must be set in **select_mask** and clear in **modify_mask**.

The constant CAP$K_ALL_CPU_ADD, when specified in **modify_mask**, indicates that all CPUs specified in **select_mask** are to be added to the affinity mask set. The constant CAP$K_ALL_CPU_REMOVE indicates that all CPUs in **select_mask** are to be removed from the affinity mask set.

prev_mask

OpenVMS usage:	mask_quadword
type:	quadword (unsigned)
access:	write only
mechanism:	by 32-bit or 64-bit reference

Previous CPU affinity mask for the specified kernel thread before execution of this call to $PROCESS_AFFINITY. The **prev_mask** argument is the 32-bit or 64-bit address of a quadword into which $PROCESS_AFFINITY writes the previous explicit affinity bit mask.

flags

OpenVMS usage:	mask_quadword
type:	quadword (unsigned)
access:	read only
mechanism:	by 32-bit or 64-bit reference

Options selected for affinity modification. The **flags** argument is a quadword bit vector wherein a bit corresponds to an option. Only the bits specified below are used; the remainder of the quadword bits are reserved and must be 0.

Each option (bit) has a symbolic name, which the $CAPDEF macro defines. The **flags** argument is constructed 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
CAP$M_FLAG_PERMANENT	Indicates whether to modify the permanent process affinities in addition to the current image copy. If CAP$M_FLAG_PERMANENT is set, then both the permanent and current affinities are modified. If the flag bit is clear or **flags** is unspecified, then just the current image process affinities are modified. This bit also determines which of the affinity masks are returned in **prev_mask**. If set, the permanent mask, used to reinitialize the current set at image rundown, is returned. If the bit is clear or the **flags** argument is not specified, the current running mask is returned.
CAP$M_FLAG_CHECK_CPU	Determines whether the kernel thread can be left in a non-runnable state under some circumstances. No operation of this service will allow a transition from a runnable to blocked state; however, if the kernel thread is already at a blocked state, this bit determines whether the result of the operation must leave it runnable. If CAP$M_FLAG_CHECK_CPU is set or **flags** is unspecified, the kernel thread will be checked to ensure it can safely run on one of the CPUs in the active set; otherwise, any valid state operations on kernel threads already in a blocked state will be allowed.
CAP$M_FLAG_CHECK_CPU_ACTIVE	Indicates whether a check is made to verify that all CPUs in the select mask that are about to be selected for affinity binding are in the active set. This does not apply to CPUs that are about to be cleared from the current affinity set. Unlike CAP$M_FLAG_CHECK_CPU where only a single CPU has to be valid for the condition to pass, CAP$M_FLAG_CHECK_CPU_ACTIVE requires that all CPUs in the selected set must pass the criteria.

Description

The Modify Process Affinity system service, based on the arguments **select_mask** and **modify_mask**, adds or removes CPUs from the specified kernel thread's affinity mask sets. If specified, the previous affinity mask is returned in **prev_mask**. With the **modify_mask** argument, multiple CPUs can be added or removed from the process affinity mask set in the same system service call.

Adding a specific CPU to the process affinity mask indicates that the kerenl thread is able to execute only on that CPU or on the others specified in the mask. Affinity scheduling takes effect as soon as the affinity mask becomes nonzero, limiting the CPU selection for the kernel thread to what is specified and available. Thread selection and execution is still subject to standard capability requirements, but only the affinity CPU set is considered when looking for an available site. When the affinity mask is cleared, all CPUs are again considered available and affinity is deactivated.

Either **modify_mask** or **prev_mask**, or both, must be specified as arguments. If **modify_mask** is specified, then **select_mask** must be specified as an argument. If **modify_mask** is not specified, then no modifications are made to the affinity mask for the specified kernel thread. In this case, **select_mask** is ignored. If **prev_mask** is not specified, then no previous mask is returned.

No service changes will be allowed if the specified kernel thread will transition from a runnable to blocked state. The CAP$M_FLAG_CHECK_CPU bit in the **flags** argument requires that the final thread state be runnable regardless of previous state; otherwise, interim changes that maintain a blocked state are allowed if the thread is already in one.

Required Privileges

The caller must have the ALTPRI privilege to call SYS$PROCESS_AFFINITY to modify its own affinity mask. To modify another process' affinity mask, the caller must have:

ALTPRI—To modify any process with a matching UIC
ALTPRI and GROUP—To modify any process in the same UIC group
ALTPRI and WORLD—To modify any process

To call SYS$PROCESS_AFFINITY simply to retrieve the specific process or global mask, the caller need only have the following privileges:

None—To retrieve the state of itself or any process with a matching UIC
GROUP—To retrieve the state of any process in the same UIC group
WORLD—To retrieve the state of any process

Related Services

$CPU_CAPABILITIES
$PROCESS_CAPABILITIES

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_BADPARAM	One of more arguments has an invalid value.
SS$_ACCVIO	The service cannot access the locations specified by one or more arguments.
SS$_NOPRIV	Insufficient privilege for attempted operation.
SS$_NOSUCHTHREAD	The specified kernel thread does not exist.
SS$_NONEXPR	The specified process does not exist, or an invalid process identification was specified.
SS$_IVLOGNAM	The process name string has a length of 0 or has more than 15 characters.

SS$_CPUCAP	No CPU can run the specified process with new affinities.
SS$_INSFARG	Fewer than the required number of arguments were specified or no operation was specified.

$PROCESS_CAPABILITIES (Alpha Only) Modify Process User Capabilities

On Alpha systems, allows modification of the user capability set for a specified kernel thread, or for the global user capability process default.

This service accepts 64-bit addresses.

Format

SYS$PROCESS_CAPABILITIES [pidadr] [,prcnam] [,select_mask] [,modify_mask] [,prev_mask] [,flags]

Arguments

pidadr

OpenVMS usage:	process_id
type:	longword (unsigned)
access:	read only
mechanism:	by 32-bit or 64-bit reference

Process identification (PID) of a kernel thread whose user capability mask is to be modified or returned. The **pidadr** argument is the 32-bit or 64-bit address of a longword that contains the PID.

Process selection is made through a combination of the **pidadr** and **prcnam** arguments. If neither are specified or if both have a zero value, the service operations are made to the user capability mask of the initial thread of the current calling process. The **pidadr** argument takes precedence over the **prcnam** argument where both are supplied in the service call.

If the constant CAP$M_FLAG_DEFAULT_ONLY is specified in **flags**, then the user portion of the default process user capability mask is modified or returned instead, regardless of the values specified in **pidadr**.

prcnam

OpenVMS usage:	process_name
type:	character-coded text string
access:	read only
mechanism:	by 32-bit or 64-bit descriptor–fixed-length string descriptor

Process name of the process whose user capability mask is to be modified or returned. The **prcnam** argument is the 32-bit or 64-bit address of a character string descriptor pointing to the process name string. A process can be identified with a 1- to 15-character string. The service operations are made to the user capability mask of the initial thread of the specified process.

You can use the **prcnam** argument only if the process identified by the descriptor has the same UIC group number as the calling process. To obtain information about processes in other groups, the **pidadr** argument must be used.

If **pidadr** and **prcnam** are both specified, then **prcnam** is ignored. If neither argument is specified, then the context of the initial thread of the calling process is modified or returned.

select_mask

OpenVMS usage:	mask_quadword
type:	quadword (unsigned)
access:	read only
mechanism:	by 32-bit or 64-bit reference

Mask specifying which bits of the specified process's user capability mask are to be modified. The **select_mask** argument is the 32-bit or 64-bit address of a quadword bit vector wherein a bit, when set, specifies that the corresponding user capability is to be modified.

The individual user capability bits in **select_mask** can be referenced by their symbolic bit constant names, CAP$M_USER1 through CAP$M_USER16. These constants (not zero-relative) specify the position in the mask quadword that corresponds to the bit name. Multiple capabilities can be selected by ORing together the appropriate bits.

Alternately, the constant CAP$K_ALL_USER, when specified as the **select_mask** argument, selects all user capabilities.

modify_mask

OpenVMS usage:	mask_quadword
type:	quadword (unsigned)
access:	read only
mechanism:	by 32-bit or 64-bit reference

Mask specifying the settings for those capabilities selected in the **select_mask** argument. The **modify_mask** argument is the 32-bit or 64-bit address of a quadword bit vector wherein a bit, when set, specifies that the corresponding user capability is to be added to the specified kernel thread; when clear, the corresponding user capability is to be removed.

The symbolic bit constants CAP$M_USER1 through CAP$M_USER16 can be used to modify the appropriate bit position in **modify_mask**. Multiple capabilities can be modified by ORing together the appropriate bits.

To add a specific user capability to a kernel thread, that bit position must be set in both **select_mask** and **modify_mask**. To remove a specific user capability from a kernel thread, that bit position must be set in **select_mask** and clear in **modify_mask**.

The symbolic constant CAP$K_ALL_USER_ADD, when specified in **modify_mask**, indicates that all capabilities specified in **select_mask** are to be added to the appropriate capability set. The symbolic constant CAP$K_ALL_USER_REMOVE indicates that all specified capabilities are to be removed from the set.

prev_mask

OpenVMS usage:	mask_quadword
type:	quadword (unsigned)
access:	write only
mechanism:	by 32-bit or 64-bit reference

Previous user capability mask for the specified process or thread before execution of this call to $PROCESS_CAPABILITIES. The **prev_mask** argument is the 32-bit or 64-bit address of a quadword into which $PROCESS_CAPABILITIES writes the previous bit mask. If CAP$M_FLAG_DEFAULT_ONLY is set in the **flags** argument, then **prev_mask** will contain the user portion of the global default capability mask.

flags

OpenVMS usage: mask_quadword
type: quadword (unsigned)
access: read only
mechanism: by 32-bit or 64-bit reference

Options selected for the user capability modification. The **flags** argument is a quadword bit vector wherein a bit corresponds to an option. Only the bits specified below are used; the remainder of the quadword bits are reserved and must be zero.

Each option (bit) has a symbolic name, defined by the $CAPDEF macro. The **flags** argument is constructed 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
CAP$M_FLAG_DEFAULT_ONLY	Indicates that the specified operations are to be performed on the global context cell instead of on a specific kernel thread. This bit supersedes any individual kernel thread specified in **pidadr** or **prcnam**. Specifying this bit constant applies the service operations to the capabilities for all newly created processes.
CAP$M_FLAG_PERMANENT	Indicates whether to modify the permanent user process capabilities in addition to the current image copy. If CAP$M_FLAG_PERMANENT is set, then both the permanent and current user process capabilities are modified. If this bit is clear or **flags** is unspecified, then just the current image process capabilities are modified. This bit also determines which of the capability masks are returned in **prev_mask**. If set, the permanent mask, used to reinitialize the current set at image rundown, is returned. If the bit is clear or the **flags** argument is not specified, the current running mask is returned.

Symbolic Name	Description
CAP$M_FLAG_CHECK_CPU	Determines whether the kernel thread can be left in a non-runnable state under some circumstances. No operation of this service will allow a transition from runnable to blocked state; however, if the kernel thread is already at a blocked state, this bit determines whether the result of the operation must leave it runnable. If CAP$M_FLAG_CHECK_CPU is set or **flags** is unspecified, the kernel thread will be checked to ensure it can safely run on one of the CPUs in the active set; otherwise, any state operations on kernel threads already in a blocked state will be allowed.

Description

The Modify Process User Capabilities system service, based on the arguments **select_mask** and **modify_mask**, adds or removes user capabilities for the specified kernel thread. If specified, the previous capability mask is returned in **prev_mask**. With the **modify_mask** argument, multiple user capabilities for a kernel thread can be added or removed in the same system service call.

Either **modify_mask** or **prev_mask**, or both, must be specified as arguments. If **modify_mask** is specified, then **select_mask** must be specified as an argument. If **modify_mask** is not specified, then no modifications are made to the user capability mask for the specified kernel thread. In this case, **select_mask** is ignored. If **prev_mask** is not specified, then no previous mask is returned.

No service changes will be allowed if the specified kernel thread will transition from a runnable to blocked state. The CAP$M_FLAG_CHECK_CPU bit in the **flags** argument requires that the final thread state be runnable regardless of previous state; otherwise, interim changes that maintain a blocked state are allowed if the thread is already in one.

If the symbolic bit constant CAP$M_FLAG_DEFAULT_ONLY is set in the **flags** argument, the user capability modifications or the mask read requests are made only to the global initialization cell regardless of what process selections values are specified in the **pidadr** and **prcnam** arguments.

Required Access or Privileges

The caller must have the ALTPRI privilege to call SYS$PROCESS_CAPABILITIES to modify its own user capability mask. To modify another process' user capability mask, the caller must have:

ALTPRI—To modify any process with a matching UIC
ALTPRI and GROUP—To modify any process in the same UIC group
ALTPRI and WORLD—To modify any process

To call SYS$PROCESS_CAPABILITIES simply to retrieve the specific process or global mask, the caller need only have the following privileges:

None—To retrieve the state of itself or any process with a matching UIC
GROUP–To retrieve the state of any process in the same UIC group

WORLD—To retrieve the state of any process

Related Services

$CPU_CAPABILITIES

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_BADPARAM	One of more arguments has an invalid value.
SS$_ACCVIO	The service cannot access the locations specified by one or more arguments.
SS$_NOSUCHTHREAD	The specified kernel thread does not exist.
SS$_NONEXPR	The specified process does not exist, or an invalid process identification was specified.
SS$_IVLOGNAM	The process name string has a length of 0 or more than 15 characters.
SS$_NOPRIV	Insufficient privilege for attempted operation.
SS$_CPUCAP	No CPU can run the specified process with new capabilities.
SS$_INSFARG	Fewer than the required number of arguments were specified or no operation was specified.

$PROCESS_SCAN
Process Scan

Creates and initializes a process context that is used by $GETJPI to scan processes on the local system or across the nodes in a VMScluster system.

Format

SYS$PROCESS_SCAN pidctx [,itmlst]

Arguments

pidctx

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

Context value supplied by $PROCESS_SCAN to be used as the **pidadr** argument of $GETJPI. The **pidctx** argument is the address of a longword that is to receive the process context longword. This longword normally contains 0 or a previous context. If it contains a previous context, the old context is deleted. If it contains a value other than 0 or a previous context, the old value is ignored.

itmlst

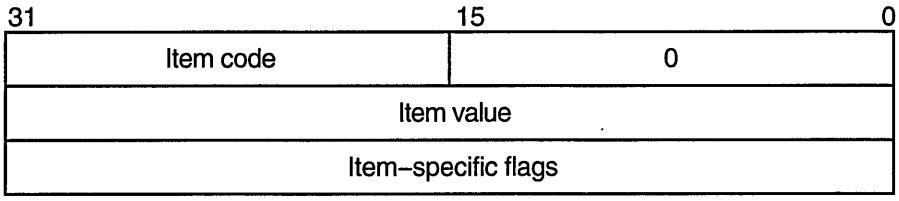
OpenVMS usage:	item_list_3
type:	longword (unsigned)
access:	read only
mechanism:	by reference

Item list specifying selection criteria to be used by the scan or to control the scan.

The **itmlst** argument is the address of a list of item descriptors, each of which describes one selection criterion or control option. Within each selection criterion you can include several item entries. The list of item descriptors is terminated by a longword of 0.

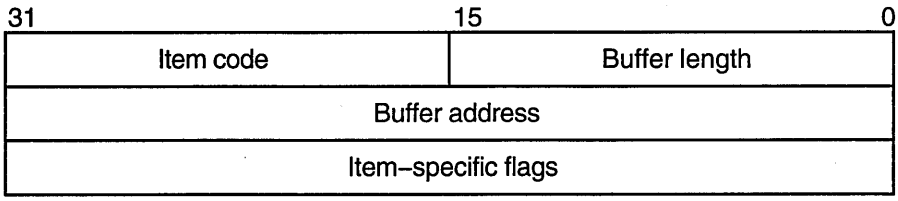
The information in the item list is passed to the item descriptor in one of two ways. If the item descriptor can always hold the actual value of the selection criterion, the value is placed in the second longword of the item descriptor and the buffer length is specified as 0. If the item descriptor points to the actual value of the selection criterion, the address of the value is placed in the second longword of the item descriptor and you must specify the buffer length for the selection criterion. Each item code description specifies whether the information is passed by value or by reference.

The following diagram depicts the format of an item descriptor that passes the selection criterion as a value.



ZK–0949A–GE

The following diagram depicts the format of an item descriptor that passes the selection criterion by reference.



ZK–0948A–GE

The following table defines the item descriptor fields.

Descriptor Field	Definition
Buffer length	Buffer length is specified in a different way for the two types of item descriptors.
	Character string or reference descriptors: A word containing a user-supplied integer specifying the length (in bytes) of the buffer from which $PROCESS_SCAN retrieves a selection criterion. The length of the buffer needed depends on the item code specified in the item descriptor.
	Immediate value descriptors: The length of the buffer is always specified as 0.
Item code	A word containing the selection criterion. These codes are defined by the $PSCANDEF macro. Each item code is described after this list of descriptor fields.

Descriptor Field	Definition
Item value	A longword containing the actual value of the selection criterion. When you specify an item code that is passed by value, $PROCESS_SCAN searches for the actual value contained in the item list. See the description of the buffer address field for information about item codes that are passed by reference.
Buffer address	A longword containing the user-supplied address of the buffer from which $PROCESS_SCAN retrieves information needed by the scan. When you specify an item code that is passed by reference, $PROCESS_SCAN uses the address as a pointer to the actual value. See the description of the item value field for information about item codes that are passed by value.
Item-specific flags	A longword that contains flags to help control selection information. Item-specific flags, for example EQL or NEQ, are used to specify how the value specified in the item descriptor is compared to the process value. These flags are defined by the $PSCANDEF macro. Some flags are common to multiple item codes; other flags are specific to an individual item code. See the description of each item code to determine which flags are used. For item codes that describe bit masks or character strings, these flags control how the bit mask or character string is compared with that in the process. By default, they are compared for equality. For item codes that describe integers, these flags specify an arithmetic comparison of an integer item with the process attribute. For example, a PSCAN$M_GTR selection specifying the value 4 for the item code PSCAN$_PRIB finds only the processes with a base priority above 4. Without one of these flags, the comparison is for equality.

Item Codes

PSCAN$_ACCOUNT
When you specify PSCAN$_ACCOUNT, $GETJPI returns information about processes that match the account field.

If the string supplied in the item descriptor is shorter than the account field, the string is blank-padded for the comparison unless the item-specific flag PSCAN$M_PREFIX_MATCH is present.

Because the information is a character string, the selection value is passed by reference. The length of the buffer is placed in the first word of the item descriptor and the address of the buffer is placed in the second longword.

Although the current length of the account field is 8 bytes, the PSCAN$_ ACCOUNT buffer can be up to 64 bytes in length. If the buffer length is 0 or greater than 64, the SS$_IVBUFLEN error is returned.

PSCAN$_AUTHPRI
When you specify PSCAN$_AUTHPRI, $GETJPI returns information about processes that match the authorized base priority field.

This integer item code is passed by value; the value is placed in the second longword of the item descriptor. The buffer length must be specified as 0.

The flags that can be used with this item code are listed in Table SYS2–3.

PSCAN$_CURPRIV
When you specify PSCAN$_CURPRIV, $GETJPI returns information about processes that match the current privilege field. Privilege bits are defined by the $PRVDEF macro.

Because the bit mask information is too long to be passed by value, the information is passed by reference. The privilege buffer must be exactly 8 bytes, otherwise the SS$_IVBUFLEN error is returned.

The flags that can be used with this item code are listed in Table SYS2–3.

PSCAN$_GETJPI_BUFFER_SIZE
When you specify PSCAN$_GETJPI_BUFFER_SIZE, you determine the size of a buffer to be used by $GETJPI to process multiple requests in a single message. Using this item code can greatly improve the performance of scans on remote nodes, because fewer messages are needed. This item code is ignored during scans on the local node.

This integer item code is passed by value; the value is placed in the second longword of the item descriptor. The buffer length must be specified as 0. The buffer is allocated by $PROCESS_SCAN; you do not have to allocate a buffer.

If you use PSCAN$_GETJPI_BUFFER_SIZE with $PROCESS_SCAN, all calls to $GETJPI using the context established by $PROCESS_SCAN must request the same item code information. Because $GETJPI locates information for more than one process at a time, it is not possible to change the item codes or the length of the buffers used in the $GETJPI item list. $GETJPI checks each call and returns the error SS$_BADPARAM if an attempt is made to change the item list during a buffered process scan. However, the buffer addresses can be changed between $GETJPI calls.

Because the locating and buffering of information by $GETJPI is transparent to a calling program, you are not required to change the way $GETJPI is called when you use this item code.

The $GETJPI buffer uses the process quota BYTLM. If the buffer is too large for the process quota, $GETJPI (not $PROCESS_SCAN) returns the error SS$_ EXBYTLM. If the buffer specified is not large enough to contain the data for at least one process, $GETJPI returns the error SS$_BADPARAM.

No item-specific flags are used with PSCAN$_GETJPI_BUFFER_SIZE.

PSCAN$_GRP
When you specify PSCAN$_GRP, $GETJPI returns information about processes that match the UIC group number.

This integer item code is passed by value; the value is placed in the second longword of the item descriptor. Because the value of the group number is a word, the high-order word of the value is ignored. The buffer length must be specified as 0.

The flags that can be used with this item code are listed in Table SYS2–3.

PSCAN$_HW_MODEL
When you specify PSCAN$_HW_MODEL, $GETJPI returns information about processes that match the specified CPU hardware model number.

The hardware model number is an integer, such as VAX$K_V8840. The VAX$ symbols are defined by the $VAXDEF macro.

This integer item code is passed by value; the value is placed in the second longword of the item descriptor. The buffer length must be specified as 0.

The flags that can be used with this item code are listed in Table SYS2–3.

PSCAN$_HW_NAME
When you specify PSCAN$_HW_NAME, $GETJPI returns information about processes that match the specified CPU hardware name, such as VAX–11/780, VAX 8800, or VAXstation II/GPX.

Because the information is a character string, the selection value is passed by reference. The length of the selection value is placed in the first word of the item descriptor and the address of the buffer is placed in the second longword.

The PSCAN$_HW_NAME buffer can be up to 128 bytes in length. If the buffer length is 0 or greater than 128, the SS$_IVBUFLEN error is returned.

The flags that can be used with this item code are listed in Table SYS2–3.

PSCAN$_JOBPRCCNT
When you specify PSCAN$_JOBPRCCNT, $GETJPI returns information about processes that match the subprocess count for the job (the count of all subprocesses in the job tree).

This integer item code is passed by value; the value is placed in the second longword of the item descriptor. The buffer length must be specified as 0.

The flags that can be used with this item code are listed in Table SYS2–3.

PSCAN$_JOBTYPE
When you specify PSCAN$_JOBTYPE, $GETJPI returns information about processes that match the job type. The job type values include the following.

Value	Description
JPI$K_LOCAL	Local interactive process
JPI$K_DIALUP	Interactive process accessed by a modem line
JPI$K_REMOTE	Interactive process accessed by using SET HOST
JPI$K_BATCH	Batch process
JPI$K_NETWORK	Noninteractive network process
JPI$K_DETACHED	Detached process

These values are defined by the $JPIDEF macro. Note that values checked by PSCAN$_JOBTYPE are similar to PSCAN$_MODE values.

This integer item code is passed by value; the value is placed in the second longword of the item descriptor. The buffer length must be specified as 0.

The flags that can be used with this item code are listed in Table SYS2–3.

PSCAN$_MASTER_PID
When you specify PSCAN$_MASTER_PID, $GETJPI returns information about processes that are descendants of the specified parent process. The master process is the first process created in the job tree. The PSCAN$_OWNER item is similar, but the owner process is the process that created the target process (the owner process might itself be a subprocess). Although all jobs in a job tree must have the same master, they can have different owners.

This integer item code is passed by value; the value is placed in the second longword of the item descriptor. The buffer length must be specified as 0.

The flags that can be used with this item code are listed in Table SYS2–3.

PSCAN$_MEM
When you specify PSCAN$_MEM, $GETJPI returns information about processes that match the UIC member number.

This integer item code is passed by value; the value is placed in the second longword of the item descriptor. Because the value of the member number is a word, the high-order word of the value is ignored. The buffer length must be specified as 0.

The flags that can be used with this item code are listed in Table SYS2–3.

PSCAN$_MODE
When you specify PSCAN$_MODE, $GETJPI returns information about processes that match the specified mode. Mode values include the following.

Value	Description
JPI$K_INTERACTIVE	Interactive process
JPI$K_BATCH	Batch job
JPI$K_NETWORK	Noninteractive network job
JPI$K_OTHER	Detached and other process

These values are defined by the $JPIDEF macro. Note that values checked by PSCAN$_MODE are similar to PSCAN$_JOBTYPE values.

This integer item code is passed by value; the value is placed in the second longword of the item descriptor. The buffer length must be specified as 0.

The flags that can be used with this item code are listed in Table SYS2–3.

PSCAN$_NODE_CSID
When you specify PSCAN$_NODE_CSID, $GETJPI returns information about processes on the specified nodes. To scan all nodes in a VMScluster system, you specify a CSID of 0 and the item-specific flag PSCAN$M_NEQ.

This integer item code is passed by value; the value is placed in the second longword of the item descriptor. The buffer length must be specified as 0.

The flags that can be used with this item code are listed in Table SYS2–3.

PSCAN$_NODENAME
When you specify PSCAN$_NODENAME, $GETJPI returns information about processes that match the specified node names.

To scan all of the nodes in a VMScluster system, specify the node name using an asterisk wildcard (*) and the PSCAN$M_WILDCARD item-specific flag.

Because the information is a character string, the selection value is passed by reference. The length of the selection value is placed in the first word of the item descriptor and the address of the buffer is placed in the second longword.

Although the current length of the node name is 6 bytes, the PSCAN$_ NODENAME buffer can be up to 64 bytes in length. If the buffer length is 0 or greater than 64, the SS$_IVBUFLEN error is returned.

The flags that can be used with this item code are listed in Table SYS2–3.

PSCAN$_OWNER
When you specify PSCAN$_OWNER, $GETJPI returns information about processes that are immediate descendants of the specified process. The PSCAN$_ MASTER_PID item is similar, but the owner process is the process that created the target process (the owner process might itself be a subprocess). Although all jobs in a job tree must have the same master, they can have different owners.

This integer item code is passed by value; the value is placed in the second longword of the item descriptor. The buffer length must be specified as 0.

The flags that can be used with this item code are listed in Table SYS2–3.

PSCAN$_PRCCNT
When you specify PSCAN$_PRCCNT, $GETJPI returns information about processes that match the subprocess count (the count of all immediate descendants of a given process). The PSCAN$_JOBPRCCNT item code is similar, except that JOBPRCCNT is the count of all subprocesses in a job.

This integer item code is passed by value; the value is placed in the second longword of the item descriptor. The buffer length must be specified as 0.

The flags that can be used with this item code are listed in Table SYS2–3.

PSCAN$_PRCNAM
When you specify PSCAN$_PRCNAM, $GETJPI returns information about processes that match the specified process names.

The process name string is blank-padded for the comparison unless the item-specific flag PSCAN$M_PREFIX_MATCH is present.

Because the information is a character string, the selection value is passed by reference. The length of the selection value is placed in the first word of the item descriptor and the address of the buffer is placed in the second longword.

Although the current length of the process name field is 15 bytes, the PSCAN$_ PRCNAM buffer can be up to 64 bytes in length. If the buffer length is 0 or greater than 64, the SS$_IVBUFLEN error is returned.

The flags that can be used with this item code are listed in Table SYS2–3.

PSCAN$_PRI
When you specify PSCAN$_PRI, $GETJPI returns information about processes that match current priority. Note that the current priority of a process can be temporarily increased as a result of system events such as the completion of I/O.

This integer item code is passed by value; the value is placed in the second longword of the item descriptor. The buffer length must be specified as 0.

The flags that can be used with this item code are listed in Table SYS2–3.

PSCAN$_PRIB
When you specify PSCAN$_PRIB, $GETJPI returns information about processes that match base priority.

This integer item code is passed by value; the value is placed in the second longword of the item descriptor. The buffer length must be specified as 0.

The flags that can be used with this item code are listed in Table SYS2–3.

PSCAN$_STATE
When you specify PSCAN$_STATE, $GETJPI returns information about processes that match the specified process state. State values, for example SCH$C_COM and SCH$C_PFW, are defined by the $STATEDEF macro.

This integer item code is passed by value; the value is placed in the second longword of the item descriptor. The buffer length must be specified as 0.

The flags that can be used with this item code are listed in Table SYS2–3.

PSCAN$_STS
When you specify PSCAN$_STS, $GETJPI returns information that matches the current status mask. Without any item-specific flags, the match is for a process mask that is equal to the pattern. Status bits, for example PCB$V_ASTPEN or PCB$V_PSWAPM, are defined by the $PCBDEF macro.

This bit mask item code uses an immediate value descriptor; the selection value is placed in the second longword of the item descriptor. The buffer length must be specified as 0.

The flags that can be used with this item code are listed in Table SYS2–3.

PSCAN$_TERMINAL
When you specify PSCAN$_TERMINAL, $GETJPI returns information that matches the specified terminal names. The terminal name string is blank-padded for the comparison unless the item-specific flag PSCAN$M_PREFIX_MATCH is present.

Because the information is a character string, the selection value is passed by reference. The length of the selection value is placed in the first word of the item descriptor and the address of the buffer is placed in the second longword.

Although the current length of the terminal name field is 8 bytes, the PSCAN$_TERMINAL buffer can be up to 64 bytes in length. If the buffer length is 0 or greater than 64, the SS$_IVBUFLEN error is returned.

The flags that can be used with this item code are listed in Table SYS2–3.

PSCAN$_UIC
When you specify PSCAN$_UIC, $GETJPI returns information about processes that match the UIC identifier. To convert an alphanumeric identifier name to the internal identifier, use the $ASCTOID system service before calling $PROCESS_SCAN.

This integer item code is passed by value; the value is placed in the second longword of the item descriptor. The buffer length must be specified as 0.

The flags that can be used with this item code are listed in Table SYS2–3.

PSCAN$_USERNAME
When you specify PSCAN$_USERNAME, $GETJPI returns information about processes that match the specified user name.

The user name string is blank-padded for the comparison unless the item-specific flag PSCAN$M_PREFIX_MATCH is present.

Because the information is a character string, the selection value is passed by reference. The length of the selection value is placed in the first word of the item descriptor and the address of the buffer is placed in the second longword.

Although the current length of the user name field is 12 bytes, the PSCAN$_USERNAME buffer can be up to 64 bytes in length. If the buffer length is 0 or greater than 64, the SS$_IVBUFLEN error is returned.

The flags that can be used with this item code are listed in Table SYS2–3.

Item-Specific Flags

Table SYS2–3 lists the flags and the item codes that can be used together. The flags are described in the section following the table.

Table SYS2–3 Flags Used with $PROCESS_SCAN

Item-Specific Flag	Description	Common to the Following $PROCESS_SCAN Item Codes
PSCAN$M_BIT_ALL	All bits set in pattern set in target	_CURPRIV
PSCAN$M_BIT_ANY	Any bit set in pattern set in target	_STS
PSCAN$M_CASE_BLIND	Match without regard to case of letters	_ACCOUNT
PSCAN$M_EQL	Match value exactly (the default)	All except _BUFFER_SIZE
PSCAN$M_GEQ	Match if value is greater than or equal to	_AUTHPRI
PSCAN$M_GTR	Match if value is greater than	_GRP
PSCAN$M_LEQ	Match if value is less than or equal to	_JOBPRCCNT
PSCAN$M_LSS	Match if value is less than	_PRI _PRIB
PSCAN$M_NEQ	Match if value is not equal	All except _BUFFER_SIZE
PSCAN$M_OR	Match this value or the next value	All except _BUFFER_SIZE
PSCAN$M_PREFIX_MATCH	Match on leading substring	_HW_NAME

(continued on next page)

Table SYS2–3 (Cont.) Flags Used with $PROCESS_SCAN

Item-Specific Flag	Description	Common to the Following $PROCESS_SCAN Item Codes
PSCAN$M_ WILDCARD	Match a wildcard pattern	_NODENAME _PRCNAM _TERMINAL _USERNAME

PSCAN$M_BIT_ALL
If the PSCAN$M_BIT_ALL flag is used, all bits set in the pattern mask specified by the item descriptor must also be set in the process mask. Other bits in the process mask can also be set.

For item codes that describe bit masks, such as privilege masks and status words, this flag controls how the pattern bit mask specified by the item descriptor is compared with that in the process. By default, the bit masks are compared for equality.

The PSCAN$M_BIT_ALL flag is used only with bit masks.

PSCAN$M_BIT_ANY
If the PSCAN$M_BIT_ANY flag is used, a match occurs if any bit in the pattern mask is also set in the process mask.

For item codes that describe bit masks, such as privilege masks and status words, this flag controls how the pattern bit mask specified by the item descriptor is compared with that in the process. By default, the bit masks are compared for equality.

The PSCAN$M_BIT_ANY flag is used only with bit masks.

PSCAN$M_CASE_BLIND
When you specify PSCAN$M_CASE_BLIND to compare the character string specified by the item descriptor with the character string value from the process, $PROCESS_SCAN does not distinguish between uppercase and lowercase letters.

The PSCAN$M_CASE_BLIND flag is used only with character-string item codes. The PSCAN$M_CASE_BLIND flag can be specified with either the PSCAN$M_ PREFIX_MATCH flag or the PSCAN$M_WILDCARD flag.

PSCAN$M_EQL
When you specify PSCAN$M_EQL, $PROCESS_SCAN compares the value specified by the item descriptor with the value from the process to see if there is an exact match.

PSCAN$M_EQL and PSCAN$M_NEQ are used with bit masks, character strings, and integers to control how the item is interpreted. Only one of the flags can be specified; if more than one of these flags is used, the SS$_BADPARAM error is returned. If you want to specify that bits not set in the pattern mask must not be set in the process mask, use PSCAN$M_EQL.

PSCAN$M_GEQ
When you specify PSCAN$M_GEQ, $PROCESS_SCAN selects a process if the value from the process is greater than or equal to the value specified by the item descriptor.

PSCAN$M_GEQ, PSCAN$M_GTR, PSCAN$M_LEQ, and PSCAN$M_LSS are used with integer item codes only. Only one of these four flags can be specified; if more than one of these flags is used, the SS$_BADPARAM error is returned.

PSCAN$M_GTR
When you specify PSCAN$M_GTR, $PROCESS_SCAN selects a process if the value from the process is greater than the value specified by the item descriptor.

PSCAN$M_GEQ, PSCAN$M_GTR, PSCAN$M_LEQ, and PSCAN$M_LSS are used with integer item codes only. Only one of these four flags can be specified; if more than one of these flags is used, the SS$_BADPARAM error is returned.

PSCAN$M_LEQ
When you specify PSCAN$M_LEQ, $PROCESS_SCAN selects a process if the value from the process is less than or equal to the value specified by the item descriptor.

PSCAN$M_GEQ, PSCAN$M_GTR, PSCAN$M_LEQ, and PSCAN$M_LSS are used with integer item codes only. Only one of these four flags can be specified; if more than one of these flags is used, the SS$_BADPARAM error is returned.

PSCAN$M_LSS
When you specify PSCAN$M_LSS, $PROCESS_SCAN selects a process if the value from the process is less than the value specified by the item descriptor.

PSCAN$M_GEQ, PSCAN$M_GTR, PSCAN$M_LEQ, and PSCAN$M_LSS are used with integer item codes only. Only one of these four flags can be specified; if more than one of these flags is used, the SS$_BADPARAM error is returned.

PSCAN$M_NEQ
When you specify PSCAN$M_NEQ, $PROCESS_SCAN selects a process if the value from the process is not equal to the value specified by the item descriptor.

PSCAN$M_EQL and PSCAN$M_NEQ are used with bit masks, character strings, and integers to control how the item is interpreted. Only one of the flags can be specified; if more than one of these flags is used, the SS$_BADPARAM error is returned.

PSCAN$M_OR
When you specify PSCAN$M_OR, $PROCESS_SCAN selects processes whose values match the current item descriptor or the next item descriptor. The next item descriptor must have the same item code as the item descriptor with the PSCAN$M_OR flag. Multiple items are chained together; all except the last item descriptor must have the PSCAN$M_OR flag.

The PSCAN$M_OR flag can be specified with any other flag and can be used with bit masks, character strings, and integers. If the PSCAN$M_OR flag is used between different item codes, or if it is missing between identical item codes, the SS$_BADPARAM error is returned.

PSCAN$M_PREFIX_MATCH
When you specify PSCAN$M_PREFIX_MATCH, $PROCESS_SCAN compares the character string specified in the item descriptor to the leading characters of the requested process value.

For example, to find all process names that start with the letters *AB*, use the string *AB* with the PSCAN$M_PREFIX_MATCH flag. If you do not specify the PSCAN$M_PREFIX_MATCH flag, the search looks for a process with the 2-character process name *AB*.

The PSCAN$M_PREFIX_MATCH flag also allows either the PSCAN$M_EQL or the PSCAN$M_NEQ flag to be specified. If you specify PSCAN$M_NEQ, the service matches those names that do *not* begin with the specified character string.

The PSCAN$M_PREFIX_MATCH flag is used only with character string item codes. The PSCAN$M_PREFIX_MATCH flag cannot be specified with the PSCAN$M_WILDCARD flag; if both of these flags are used, the SS$_BADPARAM error is returned.

PSCAN$M_WILDCARD
When you specify PSCAN$M_WILDCARD, the character string specified by the item descriptor is assumed to be a wildcard pattern. Acceptable wildcard characters are the asterisk (*), which allows the match to substitute any number of character in place of the asterisk, and the percent sign (%), which allows the match to substitute any one character in place of the percent sign. For example, if you want to search for all process names that begin with the letter *A* and end with the string *ER*, use the string *A*ER* with the PSCAN$M_WILDCARD flag. If the PSCAN$M_WILDCARD flag is not specified, the search looks for the 4-character process name *A*ER*.

The PSCAN$M_WILDCARD is used only with character string item codes. The PSCAN$M_WILDCARD flag cannot be specified with the PSCAN$M_PREFIX_MATCH flag; if both of these flags are used, the SS$_BADPARAM error is returned. The PSCAN$M_NEQ flag can be used with PSCAN$M_WILDCARD to exclude values during a wildcard search.

The following restrictions apply to the flags above:

- Only one of the flags PSCAN$M_EQL, PSCAN$M_NEQ, PSCAN$M_BIT_ALL, PSCAN$M_BIT_ANY can be specified
- PSCAN$M_CASE_BLIND item-specific flag also allows either the PSCAN$M_EQL or the PSCAN$M_NEQ flag to be specified
- Only one of the flags PSCAN$M_EQL and PSCAN$M_WILD_CARD can be specified

Description

The Process Scan system service creates and initializes a process context that is used by $GETJPI to scan processes on the local system or across the nodes in a VMScluster system. An item list is used to specify selection criteria to obtain information about specific processes, for example, all processes owned by one user or all batch processes.

The output of the $PROCESS_SCAN service is a process context longword named **pidctx**. This process context is then provided to $GETJPI as the **pidadr** argument. The process context provided by $PROCESS_SCAN enables $GETJPI to search for processes across the nodes in a VMScluster system and to select processes that match certain selection criteria.

The process context consumes process dynamic memory. This memory is deallocated when the end of the context is reached. For example, when the $GETJPI service returns SS$_NOMOREPROC or when $PROCESS_SCAN is called again with the same **pidctx** longword, the dynamic memory is deallocated. If you anticipate that a scan might be interrupted before it runs out of processes, $PROCESS_SCAN should be called a second time (without an **itmlst** argument) to release the memory. Dynamic memory is automatically released when the current image terminates.

$PROCESS_SCAN copies the item list and user buffers to the allocated dynamic memory. This means that the item lists and user buffers can be deallocated or reused immediately; they are not referenced during the calls to $GETJPI.

The item codes referenced by $PROCESS_SCAN are found in data structures that are always resident in the system, primarily the process control block (PCB) and the job information block (JIB). A scan of processes never forces a process that is swapped out of memory to be brought into memory to read nonresident information.

Required Access or Privileges

None

Required Quota

See the description for the PSCAN$_GETJPI_BUFFER_SIZE item.

Related Services

$CANEXH, $CREPRC, $DCLEXH, $DELPRC, $EXIT, $FORCEX, $GETJPI, $GETJPIW, $HIBER, $RESUME, $SETPRI, $SETPRN, $SETPRV, $SETRWM, $SUSPND, $WAKE

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The **pidctx** argument cannot be written by the caller; the item list cannot be read by the caller; or a buffer for a reference descriptor cannot be read.
SS$_BADPARAM	The item list contains an invalid item identifier, or an invalid combination of item-specific flags is present.
SS$_IVBUFLEN	The buffer length field is invalid. For immediate value descriptors, the buffer length must be 0. For reference descriptors, the buffer length cannot be 0 or longer than the maximum for the specified item code. This error is also returned if the total length of the item list plus the length of all of the buffer fields is too large to process.
SS$_IVSSRQ	The **pidctx** argument was not supplied, or the item list is improperly formed (for example, multiple occurrences of a given item code were interspersed with other item codes).

$PURGWS
Purge Working Set

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.

Format

SYS$PURGWS inadr

Argument

inadr

OpenVMS 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. The addresses are adjusted up or down to fall on CPU-specific page boundaries. Only the virtual page number portion of each virtual address is used; the low-order byte-within-page bits are ignored.

Description

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.

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 continues to execute and pages are faulted back into the working set as they are needed.

Required Access or Privileges

None

Required Quota

None

Related Services

$ADJSTK, $ADJWSL, $CRETVA, $CRMPSC, $DELTVA, $DGBLSC, $EXPREG, $LCKPAG, $LKWSET, $MGBLSC, $SETPRT, $SETSTK, $SETSWM, $ULKPAG, $ULWSET, $UPDSEC, $UPDSECW

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The input address array cannot be read by the caller.

$PURGE_WS (Alpha Only) Purge Working Set

On Alpha systems, 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.

This service accepts 64-bit addresses.

Format

SYS$PURGE_WS start_va_64 ,length_64

Arguments

start_va_64

OpenVMS usage:	address
type:	quadword address
access:	read only
mechanism:	by value

The starting virtual address of the pages to be purged from the working set. The specified virtual address will be rounded down to a CPU-specific page boundary.

length_64

OpenVMS usage:	byte count
type:	quadword (unsigned)
access:	read only
mechanism:	by value

Length of the virtual address space to be purged from the working set. The specified length will be rounded up to a CPU-specific page boundary so that it includes all CPU-specific pages in the requested range.

Description

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.

The $PURGE_WS service locates pages within the specified range and removes them if they are in the working set. To purge the entire working set, specify a range of pages from 0 through FFFFFFFF.FFFFFFFF (or to the highest possible process private virtual address, available from $GETJPI); in this case, the image continues to execute, and pages are faulted back into the working set.

Required Privileges

None

Required Quota

None.

Related Services

$ADJWSL, $LCKPAG_64, $LKWSET_64, $PURGWS, $ULKPAG_64, $ULWSET_64

Condition Values Returned

SS$_NORMAL	The service completed successfully.

$PUTMSG
Put Message

Writes informational and error messages to processes.

On Alpha systems, this service accepts 64-bit addresses.

Format

SYS$PUTMSG msgvec ,[actrtn] ,[facnam] ,[actprm]

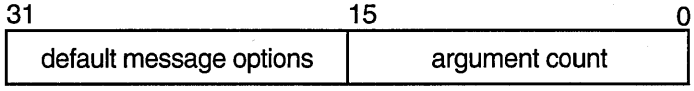
Arguments

msgvec

OpenVMS usage:	cntrlblk
type:	longword (unsigned)
access:	read only
mechanism:	by 32-bit or 64-bit reference (Alpha) by 32-bit reference (VAX)

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 32-bit or 64-bit address (on Alpha systems) or the 32-bit address (on VAX systems) 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–GE

The following table describes the message vector fields.

Descriptor Field	Definition
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 options field to the $GETMSG service as the **flags** argument.

If you specify the default message options field as 0, 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.

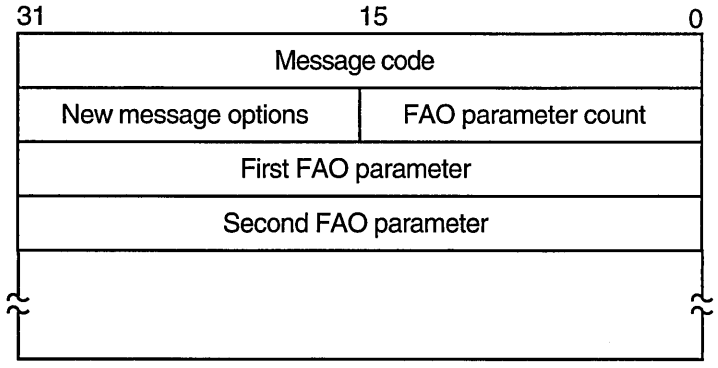
Message Descriptors

Following the first longword of the message vector are one or more message descriptors. A message descriptor can have one of four possible formats, depending on the type of message it describes. There are four types of messages:

- User-supplied
- System
- OpenVMS RMS
- System exception

The following diagrams depict the message descriptors for each type of message.

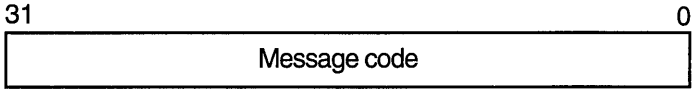
Message Descriptor for User-Supplied Messages



ZK-1718-GE

Message Descriptor Field	Definition
Message code	Longword value that uniquely identifies the message. The Description section discusses the message code; the *OpenVMS Command Definition, Librarian, and Message Utilities 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 are 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.

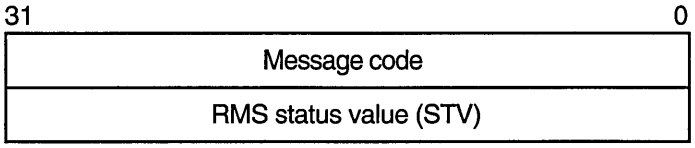
Message Descriptor for System Messages



ZK–1719–GE

Message Descriptor Field	Definition
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.

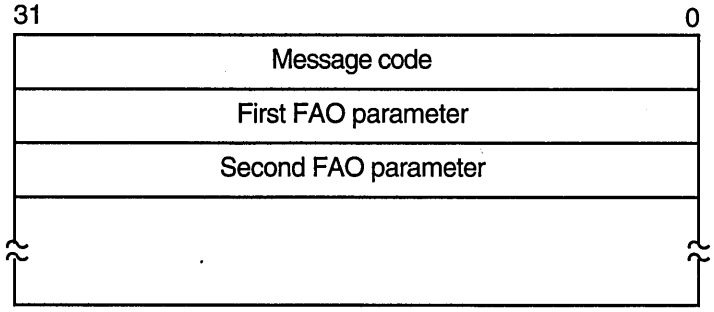
Message Descriptor for OpenVMS RMS Messages



ZK–1720–GE

Message Descriptor Field	Definition
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 OpenVMS 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



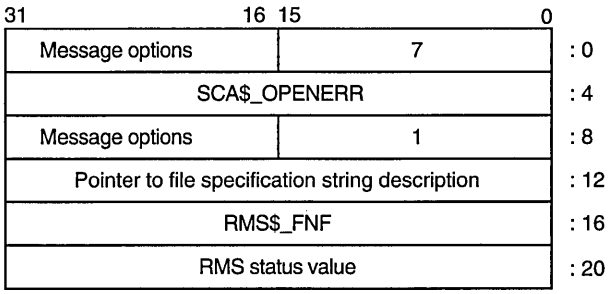
ZK–1721–GE

Message Descriptor Field	Definition
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.

Alpha

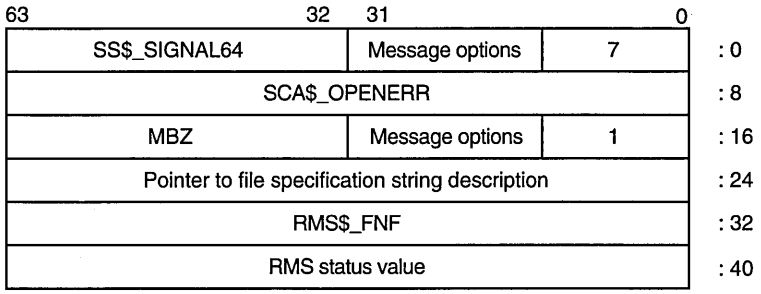
On Alpha systems, 64-bit message vectors can be used for applications that require them. A 64-bit message vector begins with the same argument count longword as the 32-bit message vector. After the argument count longword is another longword containing the value SS$_SIGNAL64, which signals that a 64-bit message vector follows. Subsequent message vector elements have a layout analogous to 32-bit message vectors but are 64-bits wide.

For example, the following diagram depicts the format of a 32-bit message vector:



ZK-8549A-GE

The 64-bit version of that same message vector would have the following format:



ZK-8550A-GE

The $PUTMSG service accepts either the 32-bit or the 64-bit form of the message vector on Alpha systems.♦

actrtn

OpenVMS usage:	procedure
type:	procedure value
access:	call without stack unwinding
mechanism:	by 32-bit or 64-bit reference (Alpha) by 32-bit reference (VAX)

User-supplied action routine to be executed during message processing. The **actrtn** argument is the 32-bit or 64-bit address (on Alpha systems) or the 32-bit address (on VAX systems) 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 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

OpenVMS usage:	char_string
type:	character-coded text string
access:	read only
mechanism:	by 32-bit or 64-bit descriptor–fixed-length string descriptor (Alpha) by 32-bit descriptor–fixed-length string descriptor (VAX)

Facility prefix to be used in the first or only message written by $PUTMSG. The **facnam** argument is the 32-bit or 64-bit address (on Alpha systems) or the 32-bit address (on VAX systems) 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.

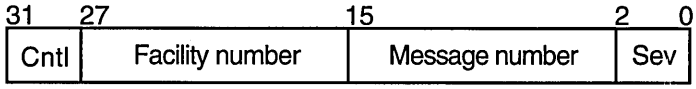
actprm

OpenVMS 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 the operating system, 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–GE

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 operating system 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 *OpenVMS Command Definition, Librarian, and Message Utilities 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 can 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.

Required Access or Privileges

None

Required Quota

None

Related Services

$ALLOC, $ASSIGN, $BRKTHRU, $BRKTHRUW, $CANCEL, $CREMBX, $DALLOC, $DASSGN, $DELMBX, $DEVICE_SCAN, $DISMOU, $GETDVI, $GETDVIW, $GETMSG, $GETQUI, $GETQUIW, $INIT_VOL, $MOUNT, $QIO, $QIOW, $SNDERR, $SNDJBC, $SNDJBCW, $SNDOPR

Condition Values Returned

SS$_NORMAL	The service completed successfully.

Example

```
#include <ssdef.h>
#include <rmsdef.h>
#include <starlet.h>

main()
{
    int msgvec[] = {3,                  /* Arg count and message flags */
                    SS$_ABORT,          /* Message code */
                    RMS$_FNF,           /* RMS Message code */
                    0};                 /* RMS Status value */

    return (sys$putmsg(msgvec));        /* Generate message */
}
```

$QIO
Queue I/O Request

Queues an I/O request to a channel associated with a device. This service completes asynchronously; for synchronous completion, use the Queue I/O Request and Wait ($QIOW) service.

For additional information about system service completion, refer to the Synchronize ($SYNCH) service.

On Alpha systems, this service accepts 64-bit addresses.

Format

SYS$QIO [efn] ,chan ,func ,[iosb] ,[astadr] ,[astprm] ,[p1] ,[p2] ,[p3] ,[p4] ,[p5] ,[p6]

Arguments

efn

OpenVMS 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

OpenVMS usage:	channel
type:	longword (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.

Specifying an invalid value for the **chan** argument will result in either SS$_IVCHAN or SS$_IVIDENT being returned.

func

OpenVMS usage:	function_code
type:	longword (unsigned)
access:	read only
mechanism:	by value

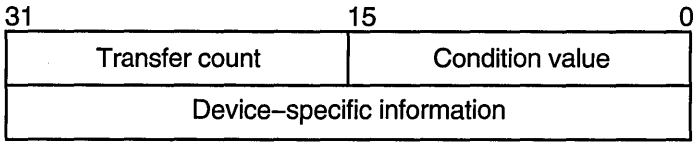
Device-specific function codes and function modifiers specifying the operation to be performed. The **func** argument is a longword containing the function code.

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 *OpenVMS I/O User's Reference Manual.*

iosb

OpenVMS usage:	io_status_block
type:	quadword (unsigned)
access:	write only
mechanism:	by 32-bit or 64-bit reference (Alpha) by 32-bit reference (VAX)

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–GE

The following table defines the I/O status block fields.

Status Block Field	Definition
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 *OpenVMS I/O User's Reference Manual.*
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 *OpenVMS I/O User's Reference Manual.*

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 first check the condition value returned in R0. If R0 contains a successful value, then you must check the condition value in the I/O status block.

astadr

OpenVMS usage:	ast_procedure
type:	procedure value
access:	call without stack unwinding
mechanism:	by 32-bit or 64-bit reference (Alpha) by 32-bit reference (VAX)

AST service routine to be executed when the I/O completes. The **astadr** argument is the address of the AST routine.

The AST routine executes at the access mode of the caller of $QIO.

astprm

OpenVMS usage:	user_arg
type:	quadword unsigned (Alpha); longword unsigned (VAX)
access:	read only
mechanism:	by 64-bit value (Alpha) by 32-bit value (VAX)

AST parameter to be passed to the AST service routine. On Alpha systems, the **astprm** argument is a quadword value containing the AST parameter. On VAX systems, the **astprm** argument is a longword value containing the AST parameter.

p1 to p6

OpenVMS usage:	varying_arg
type:	quadword (unsigned) (Alpha); longword unsigned (VAX)
access:	read only
mechanism:	by 32-bit or 64-bit reference or by 64-bit value depending on the I/O function (Alpha) by 32-bit reference or by 32-bit value depending on the I/O function (VAX)

Optional device-specific and function-specific I/O request parameters. For example, the **p1** parameter usually specifies a buffer by reference. Other parameters, such as the buffer size, disk block number, or carriage control are often passed by value.

For more information about these parameters, see the *OpenVMS I/O User's Reference Manual*.

Description

The Queue I/O Request 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 system dynamic memory to construct a database to queue the I/O request and might require additional memory depending on the queued device.

For $QIO, you can synchronize completion (1) by specifying the **astadr** argument to have an AST routine execute when the I/O completes or (2) by calling the Synchronize ($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 *DECnet for OpenVMS Networking Manual.*

Required Access or Privileges

LOG_IO or PHY_IO is required, depending upon the device type and the requested operation. DIAGNOSE is required to issue a $QIO with an associated diagnostic buffer. In addition, read or write access is generally required for the device. For more information, see the *Security Guide.*

Required Quota

The $QIO service uses the following quotas:

- 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

Related Services

$ALLOC, $ASSIGN, $BRKTHRU, $BRKTHRUW, $CANCEL, $CREMBX, $DALLOC, $DASSGN, $DELMBX, $DEVICE_SCAN, $DISMOU, $GETDVI, $GETDVIW, $GETMSG, $GETQUI, $GETQUIW, $INIT_VOL, $MOUNT, $PUTMSG, $QIOW, $SNDERR, $SNDJBC, $SNDJBCW, $SNDOPR, $IO_CLEANUP, $IO_PERFORM, $IO_SETUP

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$_ACCVIO	Either the I/O status block cannot be written by the caller, or the parameters for device-dependent function codes are specified incorrectly.
SS$_CONNECFAIL	The connection to a network object timed out or failed.
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$_FILALRACC	A logical link is already accessed on the channel (that is, a previous connect on the channel).

SS$_ILLEFC	You specified an illegal event flag number.
SS$_INSFMEM	The system dynamic memory is insufficient for completing the service.
SS$_INVLOGIN	The access control information was invalid at the remote node.
SS$_IVCHAN	You specified an invalid channel number, that is, a channel number of 0, or you failed to specify a channel number.
SS$_IVIDENT	You specified a channel number greater than the number of channels assigned for the process.
SS$_IVDEVNAM	The NCB has an invalid format or content.
SS$_LINKABORT	The network partner task aborted the logical link.
SS$_LINKDISCON	The network partner task disconnected the logical link.
SS$_LINKEXIT	The network partner task was started, but exited before confirming the logical link (that is, $ASSIGN to SYS$NET).
SS$_NOLINKS	No logical links are available. The maximum number of logical links as set for the executor MAXIMUM LINKS parameter was exceeded.
SS$_NOPRIV	The specified channel does not exist or 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$_NOSUCHNODE	The specified node is unknown.
SS$_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$_NOSUCHUSER	The remote node could not recognize the login information supplied with the connection request.
SS$_NOT64DEVFUNC	On Alpha systems, this fatal condition value is returned under the following circumstances: (1) The caller has specified a 64-bit virtual address in the P1 device dependent parameter but the device driver does not support 64-bit addresses with the requested I/O function. (2) The caller has specified a 64-bit address for a diagnostic buffer but the device driver does not support 64-bit addresses for diagnostic buffers. (3) Some device drivers may also return this condition value when 64-bit buffer addresses are passed using the P2 through P6 parameters and the driver does not support a 64-bit address with the requested I/O function.

SS$_PATHLOST	The path to the network partner task node was lost.
SS$_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$_TOOMUCHDATA	The task specified too much optional or interrupt data.
SS$_UNASEFC	The process is not associated with the cluster containing the specified event flag.
SS$_UNREACHABLE	The remote node is currently unreachable.

Condition Values Returned in the I/O Status Block

Device-specific condition values; the *OpenVMS I/O User's Reference Manual* 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; however, Digital recommends that you use an IOSB with this service to avoid premature completion.

For asynchronous completion, use the Queue I/O Request ($QIO) service.

In all other respects, $QIOW is identical to $QIO. For more information about $QIOW, refer to the description of $QIO.

For additional information about system service completion, refer to the Synchronize ($SYNCH) service.

On Alpha systems, this service accepts 64-bit addresses.

Format

SYS$QIOW [efn] ,chan ,func ,[iosb] ,[astadr] ,[astprm] ,[p1] ,[p2] ,[p3] ,[p4] ,[p5] ,[p6]

$READEF
Read Event Flags

Returns the current status of all 32 event flags in a local or common event flag cluster and indicates whether the specified event flag is set or clear.

On Alpha systems, this service accepts 64-bit addresses.

Format

SYS$READEF efn ,state

Arguments

efn

OpenVMS 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

OpenVMS usage:	mask_longword
type:	longword (unsigned)
access:	write only
mechanism:	by 32-bit or 64-bit reference (Alpha) by 32-bit reference (VAX)

State of all event flags in the specified cluster. The **state** argument is the 32-bit or 64-bit address (on Alpha systems) or the 32-bit address (on VAX systems) 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.

$RELEASE_VP (VAX Only) Release Vector Processor

On VAX systems, terminates the current process's status as a vector consumer.

Format

SYS$RELEASE_VP

Description

The Release Vector Processor service terminates the current process's status as a vector consumer. The $RELEASE_VP service waits for all pending vector instructions and vector memory operations to complete. It then declares that the process no longer needs a vector-present processor. As a result, the process relinquishes its use of the processor's vector registers and can be scheduled on another processor in the system.

In systems that do not have vector-present processors but do have the VAX Vector Instruction Emulation facility (VVIEF) in use, this service relinquishes the process's use of VVIEF. VVIEF remains mapped in the process's address space.

Required Access or Privileges

None

Required Quota

None

Related Services

$RESTORE_VP_EXCEPTION, $RESTORE_VP_STATE, $SAVE_VP_EXCEPTION

Condition Values Returned

SS$_NORMAL	The service completed successfully.

$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

Arguments

id

OpenVMS 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

OpenVMS 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 0 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.

Required Access or Privileges

Write access to the rights database is required.

Required Quota

None

Related Services

$ADD_HOLDER, $ADD_IDENT, $ASCTOID, $CREATE_RDB, $FIND_HELD, $FIND_HOLDER, $FINISH_RDB, $GRANTID, $IDTOASC, $MOD_HOLDER, $MOD_IDENT, $REM_IDENT, $REVOKID

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The **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$_NORIGHTSDB	The rights database does not exist.
SS$_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 OpenVMS RMS, this service can also return RMS status codes associated with operations on indexed files. For descriptions of these status codes, refer to the *OpenVMS Record Management Services Reference Manual*.

$REM_IDENT
Remove Identifier from Rights Database

Removes the specified identifier record and all its holder records (if any) from the rights database.

Format

SYS$REM_IDENT id

Argument

id

OpenVMS 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 Remove Identifier from Rights Database service removes from the rights database the specified identifier record, all its holder records (if any), and all records in identifiers that the deleted identifier held.

Required Access or Privileges

Write access to the rights database is required.

Required Quota

None

Related Services

$ADD_HOLDER, $ADD_IDENT, $ASCTOID, $CREATE_RDB, $FIND_HELD, $FIND_HOLDER, $FINISH_RDB, $GRANTID, $IDTOASC, $MOD_HOLDER, $MOD_IDENT, $REM_HOLDER, $REVOKID

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_INSFMEM	The process dynamic memory is insufficient for opening the rights database.
SS$_IVIDENT	The specified identifier is of invalid format.
SS$_NORIGHTSDB	The rights database does not exist.
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 OpenVMS RMS, this service can also return RMS status codes associated with operations on indexed files. For descriptions of these status codes, refer to the *OpenVMS Record Management Services Reference Manual.*

$RESCHED
Reschedule Process

Requests reschedule of a process.

Format

SYS$RESCHED

Arguments

None.

Description

The Reschedule Process service requests that the set of runnable processes on the system be evaluated by their priority, with the potential result that the current process may be descheduled and requeued.

$RESCHED is intended to allow a process running at priority *n* to voluntarily relinquish the remainder of its run quantum to another process of the same priority. When the set of all runnable processes is evaluated, one of the following will occur:

1. The process executing $RESCHED will be descheduled, while another process of equal or higher priority is selected to run. The descheduled process is placed at the end of its priority queue and all other processes at that priority will run before the process that called $RESCHED runs again. When the process does run again, $RESCHED completes and returns control to the application.
2. If, after the evaluation of all runnable processes, the process that executed $RESCHED remains the highest-priority runnable process, that process remains current and continues to run. In this case, $RESCHED returns immediately.

Required Access or Privileges

None

Required Quota

None

Related Services

None

Condition Values Returned

SS$_NORMAL	The service completed successfully.

$RESTORE_VP_EXCEPTION (VAX Only) Restore Vector Processor Exception State

On VAX systems, restores the saved exception state of the vector processor.

Format

SYS$RESTORE_VP_EXCEPTION excid

Argument

excid

OpenVMS usage:	context
type:	longword (unsigned)
access:	read only
mechanism:	by reference

Internal ID of the exception state saved by $SAVE_VP_EXCEPTION. The **excid** argument is the address of a longword containing this ID.

Description

The Restore Vector Exception State service restores from memory the vector exception state saved by a prior call to $SAVE_VP_EXCEPTION. After a routine invokes this service, the next vector instruction issued within the process causes the restored vector exception to be reported.

By default, when an AST or condition handler interrupts the execution of a mainline routine, the operating system saves the mainline routine's vector state, including its vector exception state. Any other routine that executes synchronously with, or asynchronously to, currently executing vectorized code and that performs vector operations itself must preserve the preempted routine's vector exception state across its own execution. It does so by using the $SAVE_VP_EXCEPTION and $RESTORE_VP_EXCEPTION services.

Used together, these services ensure that vector exceptions occurring as a result of activity in the original routine are serviced by existing condition handlers within that routine.

In systems that do not have vector-present processors but do have the VAX Vector Instruction Emulation facility (VVIEF) in use, VVIEF emulates the function of this service.

Required Access or Privileges

None

Required Quota

BYTLM

Related Services

$RELEASE_VP, $RESTORE_VP_STATE, $SAVE_VP_EXCEPTION

Condition Values Returned

SS$_NORMAL	The service completed successfully. The service also returns this status when executed in a system that does not have vector-present processors and that does not have the VAX Vector Instruction Emulation facility (VVIEF) loaded.
SS$_ACCVIO	The caller cannot read the exception ID longword.
SS$_NOSAVPEXC	No saved vector exception state exists for this exception ID.

$RESTORE_VP_STATE (VAX Only) Restore Vector State

On VAX systems, allows an AST routine or condition handler to restore the vector state of the mainline routine.

Format

SYS$RESTORE_VP_STATE

Arguments

None.

Description

The Restore Vector State service allows an AST routine or a condition handler to restore the vector state of the process's mainline routine.

By default, when an asynchronous routine (AST routine or condition handler) interrupts the execution of a mainline routine, the operating system creates a new vector state when the routine issues its first vector instruction. At this point, the vector state of the mainline routine is inaccessible to the asynchronous routine. If the asynchronous routine must manipulate the vector state of the mainline routine, it first calls $RESTORE_VP_STATE to restore the mainline's vector state.

In systems that do not have vector-present processors but do have the VAX Vector Instruction Emulation facility (VVIEF) in use, VVIEF emulates the functions of this service.

This service can be called only from a routine running in user mode.

Required Access or Privileges

None

Required Quota

None

Related Services

$RELEASE_VP, $RESTORE_VP_EXCEPTION, $SAVE_VP_EXCEPTION

Condition Values Returned

SS$_NORMAL	The service completed successfully. Vector state of the mainline has been restored. The service also returns this status when executed in a system that does not have vector-present processors and that does not have the VAX Vector Instruction Emulation facility (VVIEF) loaded.
SS$_BADSTACK	Bad user stack encountered.
SS$_BADCONTEXT	The mainline vector state is corrupt.

SS$_WRONGACMODE	The system service was called from an access mode other than user mode.

$RESUME
Resume Process

Causes a process previously suspended by the Suspend Process ($SUSPND) service to resume execution or cancels the effect of a subsequent suspend request.

Format

SYS$RESUME [pidadr] ,[prcnam]

Arguments

pidadr

OpenVMS 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. The **pidadr** argument can refer to a process running on the local node or a process running on another node in the VMScluster system.

You must specify the **pidadr** argument to delete processes in other UIC groups.

prcnam

OpenVMS 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. A process running on the local node can be identified with a 1- to 15-character string. To identify a process on a particular node on a cluster, specify the full process name, which includes the node name as well as the process name. The full process name can contain up to 23 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 the operating system 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.

Description

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.

If you specify neither the **pidadr** nor **prcnam** argument, the resume request is issued on behalf of the calling process.

If the longword value at address **pidadr** is 0, the PID of the target process is returned.

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 of outstanding resume requests is maintained.

Required Access or Privileges

Depending on the operation, the calling process might need one of the following privileges 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

Required Quota

None

Related Services

$CANEXH, $CREPRC, $DCLEXH, $DELPRC, $EXIT, $FORCEX, $GETJPI, $GETJPIW, $HIBER, $PROCESS_SCAN, $SETPRI, $SETPRN, $SETPRV, $SETRWM, $SUSPND, $WAKE

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$_INCOMPAT	The remote node is running an incompatible version of the operating system.
SS$_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.
SS$_NOSUCHNODE	The process name refers to a node that is not currently recognized as part of the cluster.
SS$_REMRSRC	The remote node has insufficient resources to respond to the request. (Bring this error to the attention of your system manager.)
SS$_UNREACHABLE	The remote node is a member of the cluster but is not accepting requests. (This is normal for a brief period early in the system boot process.)

$REVOKID
Revoke Identifier from Process

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]

Arguments

pidadr
OpenVMS 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 a 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
OpenVMS 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
OpenVMS 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$V_DYNAMIC	Allows unprivileged holders of the identifier to remove it from or add it to the process rights database by using the DCL command SET RIGHTS_LIST.
KGB$V_NOACCESS	Makes any access rights of the identifier null and void. This attribute is intended as a modifier for a resource identifier or the Subsystem attribute.
KGB$V_RESOURCE	Allows holders of an identifier to charge disk space to the identifier. It is used only for file objects.
KGB$V_SUBSYSTEM	Allows holders of the identifier to create and maintain protected subsystems by assigning the Subsystem ACE to the application images in the subsystem.

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

OpenVMS 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

OpenVMS 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

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.

The result of passing the **pidadr** or the **prcnam** argument, or both, to $REVOKID is summarized in the following table.

Note that a value of 0 in either of the following tables indicates that the contents of the address specified by the argument is the value 0. The word *omitted* indicates that the argument was not supplied.

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.
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 and 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. The INSFARG condition value is returned.
Omitted	Specified	Specified identifier value is used.
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 and identifier name is ignored.

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.

Required Access or Privileges

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.

Required Quota

None

Related Services

$ADD_HOLDER, $ADD_IDENT, $ASCTOID, $CREATE_RDB, $FIND_HELD, $FIND_HOLDER, $FINISH_RDB, $GRANTID, $IDTOASC, $MOD_HOLDER, $MOD_IDENT, $REM_HOLDER, $REM_IDENT

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$_ACCVIO	The **pidadr** argument cannot be read or written; **prcnam** cannot be read; **id** cannot be read or written; **name** cannot be read; or **prvatr** cannot be written.
SS$_INSFARG	You did not specify either the **id** or the **name** argument.
SS$_INSFMEM	The process dynamic memory is insufficient for opening the rights database.
SS$_IVIDENT	The specified identifier name is invalid; the identifier name is longer than 31 characters, contains an illegal character, or does not contain at least one nonnumeric character.
SS$_IVLOGNAM	You specified an invalid process name.
SS$_NONEXPR	You specified a nonexistent process.
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$_NOSYSNAM	The operation requires SYSNAM privilege.
SS$_RIGHTSFULL	The rights list of the process or system is full.
RMS$_PRV	The user does not have read access to the rights database.

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

$RMSRUNDWN
RMS Rundown

Closes all files opened by OpenVMS 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

Arguments

buf-addr

OpenVMS 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

OpenVMS 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

The RMS Rundown service closes all files opened by OpenVMS RMS for the image or process and halts I/O activity. This routine performs a $CLOSE service for each file opened for processing. In addition to closing all files and terminating I/O activity, the I/O rundown control routine releases all locks held on records in shared files, clears buffers, and returns other resources allocated for file processing. You should continue to call the rundown control routine until you receive the success completion status code of RMS$_NORMAL.

Note that, prior to the execution of the $CLOSE service, the rundown control routine cancels all outstanding file operations specified in a File Access Block (FAB) 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.

Required Access or Privileges

None

Required Quota

None

Related Services

$ALLOC, $ASSIGN, $BRKTHRU, $BRKTHRUW, $CANCEL, $CLOSE, $CREMBX, $DALLOC, $DASSGN, $DELMBX, $DEVICE_SCAN, $DISMOU, $GETDVI, $GETDVIW, $GETMSG, $GETQUI, $GETQUIW, $INIT_VOL, $MOUNT, $PUTMSG, $QIO, $QIOW, $SETDDIR, $SETDFPROT, $SNDERR, $SNDJBC, $SNDJBCW, $SNDOPR

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.

$SAVE_VP_EXCEPTION (VAX Only)
Save Vector Processor Exception State

On VAX systems, saves the pending exception state of the vector processor.

Format

SYS$SAVE_VP_EXCEPTION excid

Argument

excid

OpenVMS usage:	context
type:	longword (unsigned)
access:	read only
mechanism:	by reference

Internal ID of the exception state saved by $SAVE_VP_EXCEPTION. The **excid** argument is the address of a longword containing this ID.

Description

The Save Vector Processor Exception State service saves in memory any pending vector exception state and clears the vector processor's current exception state.

By default, when an AST or condition handler interrupts the execution of a mainline routine, the operating system saves the mainline routine's vector state, including its vector exception state. Any other routine that executes synchronously with, or asynchronously to, currently executing vectorized code and that performs vector operations itself must preserve the preempted routine's vector exception state across its own execution. It does so by using the $SAVE_VP_EXCEPTION and $RESTORE_VP_EXCEPTION services. Used together, these services ensure that vector exceptions occurring as a result of activity in the original routine are serviced by existing condition handlers within that routine.

In systems that do not have vector-present processors but do have the VAX Vector Instruction Emulation facility (VVIEF) in use, VVIEF emulates the functions of this service.

Required Access or Privileges

None

Required Quota

None

Related Services

$RELEASE_VP, $RESTORE_VP_EXCEPTION, $RESTORE_VP_STATE

Condition Values Returned

SS$_NORMAL	The service completed successfully. There were no pending vector exceptions. The service also returns this status when executed in a system that does not have vector-present processors and that does not have the VAX Vector Instruction Emulation facility (VVIEF) loaded.
SS$_WASSET	The service completed successfully. Pending vector exception state has been saved.
SS$_ACCVIO	The caller cannot write the exception ID longword.
SS$_INSFMEM	Insufficient system dynamic memory exists for completing the service.

$SCAN_INTRUSION
Scan Intrusion Database

Scans the intrusion database for suspects or intruders during a login attempt, audits login failures and updates records, or adds new records to the intrusion database.

Format

SYS$SCAN_INTRUSION logfail_status ,failed_user ,job_type ,[source_terminal] ,[source_node] ,[source_user] ,[source_addr] ,[failed_password] ,[parent_user] ,[parent_id] ,[flags]

Arguments

logfail_status

OpenVMS usage:	status code
type:	longword (unsigned)
access:	read only
mechanism:	by value

Reason why the user's login attempt failed. The **logfail_status** argument is a longword containing the login failure status code.

The **logfail_status** argument can contain any valid message code. For example, the value of the **logfail_status** argument is SS$_NOSUCHUSER if the user name the user entered does not exist on the system.

If the **logfail_status** argument contains a failure status, the service performs a suspect scan. Here, the service searches the intrusion database for intruder suspects as well as intruders. If the value of the **logfail_status** argument is a successful message, such as SS$_NORMAL, the service scans the database only for intruders. For more information about how the database works, see the *OpenVMS Guide to System Security*.

failed_user

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

User name associated with the unsuccessful login attempt. The **failed_user** argument is the address of a character-string descriptor pointing to the failed user name.

A failed user name consists of 1 to 32 alphanumeric characters.

job_type

OpenVMS usage:	job type
type:	longword (unsigned)
access:	read only
mechanism:	by value

Type of job that failed. The **job_type** argument is a longword indicating the type of job that failed.

The $JPIDEF macro defines the following values for the **job_type** argument:

- JPI$K_BATCH
- JPI$K_DETACHED
- JPI$K_DIALUP
- JPI$K_LOCAL
- JPI$K_NETWORK
- JPI$K_REMOTE

source_terminal

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

Source terminal where the login attempt is occurring. The **source_terminal** argument is the address of a character-string descriptor pointing to the device name of the terminal from which the login attempt originates.

A source terminal device name consists of 1 to 64 alphanumeric characters, including underscores (_) and colons (:).

source_node

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

Name of the node from which the user's login attempt originates. The **source_node** argument is the address of a character-string descriptor pointing to the source node name string.

A source node name consists of 1 to 1024 characters. No specific characters, format, or case is required for a source node name string.

source_user

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

User name associated with the login attempt. The **source_user** argument is the address of a character-string descriptor pointing to the source user name string.

A source user name consists of 1 to 32 alphanumeric characters, including dollar signs ($) and underscores (_).

source_addr

OpenVMS usage:	node address
type:	descriptor
access:	read only
mechanism:	by reference

Source DECnet for OpenVMS address from which the login attempt originates. The **source_addr** argument is the address of a descriptor containing the source node address.

failed_password

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

Password the user entered for the login attempt. The **failed_password** argument is the address of a character-string descriptor pointing to the plaintext password the user entered in order to log in.

A failed password is a password of 0 to 32 characters that did not allow the user to log in to the system. This argument is not stored in the intrusion database and is only used for auditing during break-in attempts.

parent_user

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

Parent process name of the failed login. The **parent_user** argument is the address of a character-string descriptor pointing to the parent process name of the failed login process.

A parent process name consists of 1 to 15 characters. This argument should be specified only for failed spawn commands.

parent_id

OpenVMS usage:	process_id
type:	longword (unsigned)
access:	read only
mechanism:	by value

Process identification of the parent process from which the login was attempted. The **parent_id** argument is a longword containing the parent process identification.

flags

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

Operational instructions for the service. The **flags** argument is a longword bit mask wherein each bit corresponds to an option.

Each flag option has a symbolic name. The $CIADEF macro defines the following valid names for the $SCAN_INTRUSION service.

Symbolic Name	Description
CIA$M_NOAUDIT	If set, this flag indicates that the service should instruct the security server to not audit the login failure or the break-in attempt. If the flag is set, you are expected to do your own auditing.

Symbolic Name	Description
CIA$M_IGNORE_RETURN	Specifies that the service should not wait for the return status from the security server. No return status from the server's function will be returned to the caller.
CIA$M_REAL_USERNAME	If set, indicates that the user name passed as the failed user name is read and known to the system.
CIA$M_SECONDARY_PASSWORD	Indicates that the failed password passed to the service was the secondary password. If the flag is clear, the password is assumed to be the primary password.

Description

The Scan Intrusion service performs the following functions:

- Scans the intrusion database for intruders so that successful logins are evaded if the system is taking evasive action.
- Adds login failures to the intrusion database.
- Changes records in the intrusion database from suspects to intruders when the number of login failures by the specified user or from the specified source reaches the value of the LGI_BREAK_LIM system parameter.
- Disables user accounts if the LGI_BRK_DISUSER flag is set and the number of login attempts on a real user has reached LGI_BRK_LIM.
- Audits login failures or break-in attempts on behalf of the caller.

The information that $SCAN_INTRUSION stores in the intrusion database is based on the setting of the LGI_BRK_TERM system parameter and the information passed by the caller. For more information about how the intrusion database functions and the use of the LGI system parameters, see the *OpenVMS Guide to System Security*.

Required Access or Privileges

$SCAN_INTRUSION requires the SECURITY privilege.

Required Quota

None

Related Services

$DELETE_INTRUSION, $SHOW_INTRUSION

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	One or more of the arguments were not readable.
SS$_BADBUFLEN	The length of one or more of the specified arguments is out of range.
SS$_BADPARAM	An invalid flag was specified in the **flags** argument.

SS$_NOSECURITY	The caller does not have SECURITY privilege.

This service can also return any of the following messages passed from the security server:

SECSRV$_INSUFINFO	Not enough information is supplied to form an intrusion record.
SECSRV$_INTRUDER	An intruder matching the information passed to the service exists in the intrusion database.
SECSRV$_NOMATCH	No intruders or suspects exist that match the information passed to the service.
SECSRV$_ SERVERNOTACTIVE	The security server is not currently active. Try the request again later.
SECSRV$_SUSPECT	A suspect matching the information passed to the service exists in the intrusion database.

$SCHDWK
Schedule Wakeup

Schedules the awakening (restarting) of a process that has placed itself in a state of hibernation with the Hibernate ($HIBER) service.

Format

SYS$SCHDWK [pidadr] ,[prcnam] ,daytim ,[reptim]

Arguments

pidadr

OpenVMS 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. The **pidadr** argument can refer to a process running on the local node or a process running on another node in the VMScluster system.

You must specify the **pidadr** argument to awaken processes in other UIC groups.

prcnam

OpenVMS 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. A process running on the local node can be identified with a string of from 1 to 15 characters.

To identify a process on a particular node on a cluster, specify the full process name, which includes the node name as well as the process name. The full process name can contain up to 23 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 the operating system 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.

daytim

OpenVMS 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

OpenVMS 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

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.

If you specify neither the **pidadr** nor the **prcnam** argument, the wakeup request is issued on behalf of the calling process. If the longword value at address **pidadr** is 0, the PID of the target process is returned.

$SCHDWK uses the 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 of outstanding wakeup requests is maintained.

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).

Required Access or Privileges

Depending on the operation, the calling process might need one of the following privileges 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

Required Quota

This service 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 calling process to schedule a wakeup request.

Related Services

$ASCTIM, $BINTIM, $CANTIM, $CANWAK, $GETTIM, $NUMTIM, $SETIME, $SETIMR

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$_EXQUOTA	The process has exceeded its AST limit quota.
SS$_INCOMPAT	The remote node is running an incompatible version of the operating system.
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$_NOPRIV	The process does not have the privilege to schedule a wakeup request for the specified process.
SS$_NOSUCHNODE	The process name refers to a node that is not currently recognized as part of the VMScluster system.
SS$_REMRSRC	The remote node has insufficient resources to respond to the request. (Bring this error to the attention of your system manager.)
SS$_UNREACHABLE	The remote node is a member of the cluster but is not accepting requests. (This is normal for a brief period early in the system boot process.)

$SCHED
Affect Process Scheduling

Affects process scheduling. This service is intended for use by a class scheduler process.

Format

SYS$SCHED func ,p1 ,p2 ,p3

Arguments

func

OpenVMS usage:	function_code
type:	longword (unsigned)
access:	write only
mechanism:	by value

Function code specifying the action $SCHED is to perform. The **func** argument is a longword containing this code.

See the Function Codes section for a list of valid function codes for $SCHED.

p1, p2, p3

OpenVMS usage:	longword
type:	longword (unsigned)
access:	varies
mechanism:	varies

The meaning of the **p1**, **p2**, and **p3** arguments depends on the function code specified in the **func** argument, and is defined in the Function Codes section.

Function Codes

This section defines each of the $SCHED function codes and describes the values of the **p1** argument, **p2** argument, and **p3** argument for each function.

CSH$_READ_ALL

When you specify CSH$_READ_ALL, $SCHED returns a buffer containing information, including an index, EPID, and priority, for all processes.

The format of the buffer is defined in the $CSHDEF macro and consists of a series of CSHP fields.

The following table shows the **p1** argument, **p2** argument, and **p3** argument values for the CSH$_READ_ALL function code.

Argument	Access	Description
p1	Read	Address of the buffer.
p2	Read	Address of the longword size of the buffer.
p3	Write	Address of the longword size of the per-process entry.

CSH$_READ_NEW

When you specify CSH$_READ_NEW, $SCHED returns a buffer containing information, including an index, EPID, and priority, for all processes for which a class assignment has not been made.

The format of the buffer is defined in the $CSHDEF macro and consists of a series of CSHP fields.

The following table shows the **p1** argument, **p2** argument, and **p3** argument values for the CSH$_READ_NEW function code.

Argument	Access	Description
p1	Read	Address of the buffer.
p2	Read	Address of the longword size of the buffer.
p3	Write	Address of the longword size of the per-process entry.

The following table describes the information returned in the buffer fields for both CSH$_READ_ALL and CSH$_READ_NEW.

Buffer Field	Definition
CSHP$T_ACCOUNT	Account string from the user authorization file (first eight characters).
CSHP$L_CPUTIM	Process CPU time used, in 10-millisecond ticks.
CSHP$L_EPID	Process ID (PID). If CSHP information is insufficient to determine the right class for a process, the PID can be used with the $GETJPI(W) system service to obtain additional detail.
CSHP$W_PIX	A unique integer assigned to the process for its duration. Applications may wish to use this value to index arrays.
CSHP$B_PRI	Current process priority.
CSHP$B_PRIB	Base process priority.
CSHP$L_STATUS	Undefined; reserved to Digital.

CSH$_READ_QUANT

When you specify CSH$_READ_QUANT, $SCHED returns a buffer containing information about how many ticks are left for each class. Data is returned in a series of longwords, one longword per class, starting with class number 0.

The following table defines the **p1** argument, **p2** argument, and **p3** argument values when specifying the CSH$_READ_QUANT function code.

Argument	Access	Description
p1	Read	Address of the buffer.
p2	Read	Address of the longword size of the buffer.
p3	—	Unused.

CSH$_SET_ATTN_AST

Enables attention asynchronous system traps (ASTs).

The following table defines the **p1** argument, **p2** argument, and **p3** argument values when specifying the CSH$_SET_ATTN_AST function code.

Argument	Access	Description
p1	Read	Address of an AST routine.
p2	Read	Access mode to deliver AST.
p3	—	Unused.

CSH$_SET_CLASS

Places processes in classes with or without windfall capability. The caller supplies a buffer consisting of CSHC blocks.

The format of the buffer is defined in the $CSHDEF macro. The following table describes the information contained in the buffer.

Buffer Field	Definition
CSHC$L_EPID	Process ID (PID) of the process to affect.
CSHC$W_CLASS	Class into which to place the process. Class 65535 (hexadecimal FFFF) has a special interpretation: the associated process is not to be class scheduled and will, therefore, never run out of class quantum. The largest class number is 8191.
CSHC$W_WINDFALL	Determines whether the process is to share windfall. A value of 1 permits the process to share windfall; a value of 0 prevents the process from sharing windfall. Values other than 0 and 1 are undefined and may cause unpredictable behavior in future releases of the operating system.

The following table defines the **p1** argument, **p2** argument, and **p3** argument values when specifying the CSH$_SET_CLASS function code.

Argument	Access	Description
p1	Read	Address of the buffer.
p2	Read	Address of the longword size of the buffer.
p3	Read	Address of the longword size of the entry used. Should be CSHC$K_LENGTH or equivalent.

CSH$_SET_NEW

Indicates to the class scheduler that the next READ_NEW will return information about the calling process.

The following table defines the **p1** argument, **p2** argument, and **p3** argument values when specifying the CSH$_SET_NEW function code.

Argument	Access	Description
p1	—	Unused.
p2	Read	PID (by value).
p3	—	Unused.

CSH$_SET_QUANT

Establishes class quantum and enables class scheduling. The caller supplies a buffer that allocates CPU ticks to classes, one longword per class, starting with class number 0. Class-scheduled processes will have their quantum deducted from the appropriate longword, and will be removed from execution if class quantum is decremented to 0.

The following table defines the **p1** argument, **p2** argument, and **p3** argument values when specifying the CSH$_SET_QUANT function code.

Argument	Access	Description
p1	Read	Address of the buffer.
p2	Read	Address of the longword size of buffer.
p3	—	Unused.

CSH$_SET_TIMEOUT

Establishes a nonstandard timeout. If the application does not issue a CSH$_SET_QUANT within the timeout period, all class scheduling is stopped and processes are returned to normal scheduling. The default value, 30 seconds, should be suitable for most circumstances.

The following table defines the **p1** argument, **p2** argument, and **p3** argument values when specifying the CSH$_SET_TIMEOUT function code.

Argument	Access	Description
p1	—	Unused.
p2	Read	Time in seconds (by value).
p3	—	Unused.

Description

The Affect Process Scheduling service is used by class scheduler processes to affect scheduling.

Use the **func** argument to specify which action $SCHED is to perform.

Required Access or Privileges

ALTPRI is required to affect processes. Group access is required to affect processes in the same UIC group. World access is required to affect processes in different UIC groups. SYSPRV is required to set the timeout value.

Required Quota

None

Related Services

$GETJPI, $GETJPIW, $SETPRI

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_BADPARAM	The function code is invalid.
SS$_ILLSER	The loadable image CLASS_SCHEDULER has not been loaded. Refer to the SYSMAN command SYS_LOADABLE in the *OpenVMS System Management Utilities Reference Manual* for instructions.

$SETAST
Set AST Enable

Enables or disables the delivery of asynchronous system traps (ASTs) for the access mode from which the service call was issued.

Format

SYS$SETAST enbflg

Argument

enbflg
OpenVMS 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

The Set AST Enable service enables or disables the delivery of ASTs for the access mode from which the service call was issued.

Required Access or Privileges

When an image is executing in user mode, ASTs are always enabled for more privileged access modes. If ASTs are disabled for a more privileged access mode, the operating system 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 reenable ASTs for that mode before returning to a less privileged access mode.

Required Quota

None

Related Services

$DCLAST, $SETPRA

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.

$SETCLUEVT (Alpha Only) Set Cluster Event

On Alpha systems, establishes a request for notification when a VMScluster configuration event occurs.

Format

SYS$SETCLUEVT event ,astadr ,[astprm] ,[acmode] ,[handle]

Arguments

event
OpenVMS usage: event_code
type: longword (unsigned)
access: read only
mechanism: by value

Event code indicating the type of cluster configuration event for which an AST is to be delivered. The **event** argument is a value indicating which type of event is of interest.

Each event type has a symbolic name. The $CLUEVTDEF macro defines the following symbolic names.

Symbolic Name	Description
CLUEVT$C_ADD	One or more OpenVMS nodes have been added to the VMScluster system.
CLUEVT$C_REMOVE	One or more OpenVMS nodes have been removed from the VMScluster system.

astadr
OpenVMS usage: ast_procedure
type: procedure value
access: call without stack unwinding
mechanism: by reference

Notification AST routine to receive control after a change in VMScluster configuration occurs.

astprm
OpenVMS usage: user_arg
type: longword (unsigned)
access: read only
mechanism: by value

Optional AST parameter to be passed to the AST service routine. The **astprm** argument is a longword value containing the AST parameter.

acmode
OpenVMS usage: access_mode
type: longword (unsigned)
access: read only
mechanism: by value

Optional access mode at which the configuration event AST is to execute. The **acmode** argument is a longword containing the access mode.

Each access mode has a symbolic name. 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 value of the access mode must not be more privileged than the access mode of the caller.

handle

OpenVMS usage: identifier
type: quadword (unsigned)
access: write only
mechanism: by reference

Optional identifier to receive a value that uniquely identifies this AST request. $SETCLUEVT sets this handle to a unique value so that it can later be used to identify the request in the $CLRCLUEVT and $TSTCLUEVT services.

Description

The Set Cluster Event service establishes a request for notification when a cluster configuration event occurs. The service establishes only one AST notification for a configuration event. To receive AST notification for all cluster configuration events, the $SETCLUEVT service must be reissued within the notification AST routine. The service will verify that the input parameters specify a valid request, allocate appropriate data structures to hold the request, and enqueue the request for notification.

You must specify an event type and an AST address. You can specify an AST parameter, the access mode, and an address into which to place the handle of this request.

Errors will be returned in the following cases:

- If quotas are exceeded, an error identifying the specific quota will be returned. It is important to note that this routine will return an error and will not retry an attempt to get quota if quota is exhausted on the first attempt. See the Condition Values Returned section for types of errors that may be returned.
- If the **astadr** argument is omitted, SS$_BADPARAM will be returned.
- If the **event** argument is omitted or incorrectly specified, SS$_BADPARAM will be returned.
- If the access mode parameter is more privileged than the mode of the caller, the mode of the caller will be used.
- If specified, the **handle** argument must be readable and writable from the mode of the caller. SS$_ACCVIO is returned if this is not the case.

Required Access or Privileges

None

Required Quota

None

Related Services

$CLRCLUEVT, $TSTCLUEVT

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	Unable to process parameters for improper use.
SS$_BADPARAM	The event was improperly specified.
SS$_EXASTLM	The process exceeded its quota for outstanding AST requests.
SS$_INSFMEM	The system dynamic memory is insufficient to complete the service.

$SETDDIR
Set Default Directory

Allows you to read and change the default directory string for the process.

Format

SYS$SETDDIR [new-dir-addr] ,[length-addr] ,[cur-dir-addr]

Arguments

new-dir-addr
OpenVMS 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
OpenVMS 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
OpenVMS 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 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. 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.

Required Access or Privileges
None

Required Quota

None

Related Services

$ALLOC, $ASSIGN, $BRKTHRU, $BRKTHRUW, $CANCEL, $CREMBX, $DALLOC, $DASSGN, $DELMBX, $DEVICE_SCAN, $DISMOU, $GETDVI, $GETDVIW, $GETMSG, $GETQUI, $GETQUIW, $INIT_VOL, $MOUNT, $PUTMSG, $QIO, $QIOW, $SNDERR, $SNDJBC, $SNDJBCW, $SNDOPR

Condition Values Returned

RMS$_NORMAL	The service completed successfully.
RMS$_DIR	The directory name contains an error.
RMS$_IAL	The argument list is invalid.

$SETDFPROT
Set Default File Protection

Allows you to read and write the default file protection for the process.

Format

SYS$SETDFPROT [new-def-prot-addr] ,[cur-def-prot-addr]

Arguments

new-def-prot-addr

OpenVMS 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

OpenVMS 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

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.

There is no predefined macro of the form $SETDEFPROT_G or $SETDEFPROT_S to call this service.

Required Access or Privileges

None

Required Quota

None

Related Services

$ALLOC, $ASSIGN, $BRKTHRU, $BRKTHRUW, $CANCEL, $CREMBX, $DALLOC, $DASSGN, $DELMBX, $DEVICE_SCAN, $DISMOU, $GETDVI, $GETDVIW, $GETMSG, $GETQUI, $GETQUIW, $GET_SECURITY, $INIT_VOL, $MOUNT, $PUTMSG, $QIO, $QIOW, $SET_SECURITY, $SNDERR, $SNDJBC, $SNDJBCW, $SNDOPR

Condition Values Returned

RMS$_NORMAL	The service completed successfully.
RMS$_IAL	The argument list is invalid.

$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

Argument

efn

OpenVMS 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

Assigns a condition handler address to the primary, secondary, or last chance exception vectors, or removes a previously assigned handler address from any of these three vectors.

Format

SYS$SETEXV [vector] ,[addres] ,[acmode] ,[prvhnd]

Arguments

vector

OpenVMS 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 exception vector; the value 1, the secondary vector; and the value 2, the last chance exception vector.

addres

OpenVMS usage:	procedure
type:	procedure value
access:	call without stack unwinding
mechanism:	by reference

Condition handler address to be established for the exception vector specified by the **vector** argument. The **addres** argument is a longword value containing the address of the condition handler routine.

If you do not specify **addres** or specify it as the value 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

OpenVMS 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

OpenVMS usage:	procedure value
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 $SETEXV writes the handler's procedure value.

Description

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. A process cannot modify a vector associated with a more privileged access mode.

The operating system 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.
- VAX On VAX systems, 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 OpenVMS 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.

Required Access or Privileges

None

Required Quota

None

Related Services

$DCLCMH, $SETSFM, $UNWIND

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

Changes the value of, or recalibrates, the system time.

Format

SYS$SETIME [timadr]

Argument

timadr

OpenVMS 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 the value of **timadr** or specify it as 0, $SETIME recalibrates the system time using the time-of-year clock.

Description

The Set System Time service (1) changes the value of or (2) recalibrates the system time which is defined by a quadword value that specifies the number of 100-nanosecond intervals since 00:00 o'clock, November 17, 1858.

System time is the reference used for nearly all timer-related software activities in the operating system. 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 $QIOW. You need the LOG_IO user privilege to perform this operation.

Required Access or Privileges

To set system time, the calling process must have OPER and LOG_IO privileges.

Required Quota

None

Related Services

$ASCTIM, $BINTIM, $CANTIM, $CANWAK, $GETTIM, $NUMTIM, $SCHDWK, $SETIMR

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The quadword that contains the new system time value cannot be read by the caller.
SS$_IVTIME	The caller specified no time value or a negative time value and an invalid processor clock was found.
SS$_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

Sets the timer to expire at a specified time.

On Alpha systems, this service accepts 64-bit addresses.

Format

SYS$SETIMR [efn] ,daytim ,[astadr] ,[reqidt] ,[flags]

Arguments

efn

OpenVMS 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

OpenVMS usage:	date_time
type:	quadword
access:	read only
mechanism:	by 64-bit reference (Alpha) by 32-bit reference (VAX)

Time at which the timer expires. The **daytim** argument is the 64-bit address (on Alpha systems) or the 32-bit address (on VAX systems) 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.

Alpha

On Alpha systems, if a specified absolute time value has already passed, the timer expires within 10 milliseconds.♦

VAX

On VAX systems, if a specified absolute time value has already passed, the timer expires at the next clock cycle, which is within 10 milliseconds.♦

On Alpha and VAX systems, the Convert ASCII String to Binary Time ($BINTIM) service converts an ASCII string time value to the quadword time value required by $SETIMR.

astadr

OpenVMS usage:	ast_procedure
type:	procedure value
access:	call without stack unwinding
mechanism:	by 64-bit reference (Alpha) by 32-bit reference (VAX)

AST service routine that is to execute when the timer expires. The **astadr** argument is the 64-bit address (on Alpha systems) or the 32-bit address (on VAX systems) of the procedure value of this routine. If you do not specify the value of **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

OpenVMS 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 0 as a request to cancel all timer requests.

You can specify unique values for **reqidt** for each timer request or give the same value to related timer requests. This permits selective canceling 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

OpenVMS 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. The **flags** argument is optional.

Description

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. This service requires dynamic memory and executes at the access mode of the caller, as does the AST routine if one is specified.

Required Access or Privileges

None

Required Quota

This service 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.

Related Services

$ASCTIM, $BINTIM, $CANTIM, $CANWAK, $GETTIM, $NUMTIM, $SCHDWK, $SETIME

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.

$SET_IMPLICIT_AFFINITY (Alpha Only) Modify Process Implicit Affinity

On Alpha systems, controls or retrieves the activation state for the implicit affinity system capability of a specific kernel thread or of the global process default.

This service accepts 64-bit addresses.

Format

SYS$SET_IMPLICIT_AFFINITY [pidadr] [,prcnam] [,state] [,cpu_id] [,prev_mask]

Arguments

pidadr

OpenVMS usage:	process_id
type:	longword (unsigned)
access:	read only
mechanism:	by 32-bit or 64-bit reference

Process identification (PID) of a kernel thread whose implicit affinity is to be modified or returned. The **pidadr** argument is the 32-bit or 64-bit address of a longword that contains the PID.

Process selection is made through a combination of the **pidadr** and **prcnam** arguments. If neither are specified or if both have a zero value, the service operations are made to the user capability mask of the initial thread of the current calling process. The **pidadr** argument takes precedence over the **prcnam** argument where both are supplied in the service call.

If the bit constant CAP$M_IMPLICIT_DEFAULT_ONLY is specified in the **state** argument, then the implicit affinity state portion of the default capability mask is modified or returned instead.

prcnam

OpenVMS usage:	process_name
type:	character-coded text string
access:	read only
mechanism:	by 32-bit or 64-bit descriptor–fixed-length string descriptor

Process name of the process whose implicit affinity capability state is to be modified or returned. The **prcnam** argument is the 32-bit or 64-bit address of a character string descriptor pointing to the process name string. A process can be identified with a 1- to 15-character string. The service operations are made to the user capability mask of the initial thread of the specified process.

If **pidadr** and **prcnam** are both specified, then **pidadr** is modified or returned and **prcnam** is ignored. If neither argument is specified, then the context of the initial thread of the calling process is modified or returned.

state

OpenVMS usage:	mask_quadword
type:	quadword (unsigned)
access:	read only
mechanism:	by 32-bit or 64-bit reference

State options that can be selected for the affected thread's implicit affinity. The **state** argument is a pointer to a quadword bit vector wherein a bit corresponds to a requested state for the implicit affinity feature. Only the bits specified below are used; the remainder of the quadword bits are reserved.

Each option (bit) has a symbolic name, defined in the $CAPDEF macro. The **state** argument is constructed 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
CAP$M_IMPLICIT_DEFAULT_ ONLY	Indicates the specified operations are to be performed on the global cell instead of on a specific kernel thread. This bit supersedes any individual kernel thread specified in **pidadr** or **prcnam**. Specifying this bit constant applies the implicit affinity operations to all newly created processes.
CAP$M_IMPLICIT_AFFINITY_SET	Indicates that the implicit affinity capability bit is to be set for the specified kernel thread. This is mutually exclusive with CAP$M_IMPLICIT_AFFINITY_ CLEAR.
CAP$M_IMPLICIT_AFFINITY_ CLEAR	Indicates that the implicit affinity capability bit is to be cleared for the specified kernel thread. This is mutually exclusive with CAP$M_IMPLICIT_ AFFINITY_SET.

cpu_id

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

Identifier of the CPU requested as the first CPU on which this kernel thread is to execute. The **cpu_id** is a longword containing this number, which is in the supported range of individual CPUs from 0 to SYI$_MAX_CPUS −1.

If no explicit CPU is needed, specifying a value of −1 in this argument indicates the system is to select the initial association based on system dynamics and load balancing.

Note that, regardless of what explicit CPU is supplied to this argument, it will be taken only as a suggestion. This service will attempt to make the requested association, but it will be superseded by another CPU if the system dynamics are adversely affected by the operation.

prev_mask

OpenVMS usage:	mask_quadword
type:	quadword (unsigned)
access:	write only
mechanism:	by 32-bit or 64-bit reference

Previous implicit affinity state mask for the specified kernel thread before execution of this call to $SET_IMPLICIT_AFFINITY. The **prev_mask** argument is the 32-bit or 64-bit address of a quadword into which $SET_IMPLICIT_AFFINITY writes a bit mask specifying the implicit affinity state.

The current state of the kernel thread's current implicit affinity feature can be determined by testing the returned mask with the symbolic bit definitions described for the **state** argument. These bit definitions are found in the $CAPDEF macro.

Description

The Modify Process Implicit Affinity system service modifies or returns the implicit affinity state for the specified kernel thread or from the system default process creation cell.

Setting a kernel thread's implicit affinity function indicates to the system that it is to schedule the process in ways that will maximize the cache and TB performance in the current symmetric multiple processor (SMP) configuration. This might tend to bias the process towards specific CPUs more than the standard scheduling algorithm would normally have allowed.

Required Access or Privileges

The caller must have the ALTPRI privilege to call SYS$SET_IMPLICIT_AFFINITY to modify its own implicit affinity capability bit. To modify another process' capability mask, the caller must have:

ALTPRI—To modify any process with a matching UIC
ALTPRI and GROUP—To modify any process in the same UIC group
ALTPRI and WORLD—To modify any process

To call SYS$SET_IMPLICIT_AFFINITY simply to retrieve the state of a specific process or global bit, the caller need only have the following privileges:

None—To retrieve the state of itself or any process with a matching UIC
GROUP—To retrieve the state of any process in the same UIC group
WORLD—To retrieve the state of any process

Related Services

$CPU_CAPABILITIES, $PROCESS_CAPABILITIES, $PROCESS_AFFINITY

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_BADPARAM	One of more arguments has an invalid value.
SS$_ACCVIO	The service cannot access the locations specified by one or more arguments.
SS$_NOSUCHTHREAD	The specified kernel thread does not exist.
SS$_NONEXPR	The specified process does not exist, or an invalid process identification was specified.
SS$_IVLOGNAM	The process name string has a length of 0 or more than 15 characters.
SS$_NOPRIV	Insufficient privilege for attempted operation.

SS$_CPUCAP	No CPU can run the specified process with new capabilities.
SS$_INSFARG	Fewer than the required number of arguments were specified or no operation was specified.

$SETPRA
Set Power Recovery AST

Establishes a routine to receive control after a power recovery is detected.

Format

SYS$SETPRA astadr ,[acmode]

Arguments

astadr

OpenVMS usage:	ast_procedure
type:	procedure value
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 this routine.

If you specify **astadr** as the value 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

OpenVMS 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 access modes.

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

Description

The Set Power Recovery AST service establishes a routine to receive control after a power recovery is detected.

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.

Required Access or Privileges

None

Required Quota

One unit of quota is deducted from the process's ASTLM.

Related Services

$DCLAST, $SETAST

For more information, see the chapter on AST services in the *OpenVMS Programming Concepts Manual*.

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_EXQUOTA	The process exceeded its quota for outstanding AST requests.

$SETPRI
Set Priority

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] ,[nullarg] ,[nullarg]

Arguments

pidadr

OpenVMS 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. The **pidadr** argument can refer to a process running on the local node or a process running on another node in the cluster.

prcnam

OpenVMS 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 the process name. A process running on the local node can be identified with a 1- to 15-character string. To identify a process on a particular node on a cluster, specify the full process name, which includes the node name as well as the process name. The full process name can contain up to 23 characters.

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.

pri

OpenVMS 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
OpenVMS 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.

Alpha

policy
OpenVMS usage: longword_unsigned
type: longword (unsigned)
access: read only
mechanism: by reference

On Alpha systems, address of a longword containing the new scheduling policy for the process. The $JPIDEF macro defines the following symbols for the **policy** argument.

Symbol	Meaning
JPI$K_DEFAULT_POLICY	The normal scheduling policy. The priority interval for this policy is defined as [0..*n*], such that priorities [0..15] are interactive and priorities [16..*n*] are real time.
JPI$K_PSX_FIFO_POLICY	Posix FIFO scheduling policy. The priority interval for this policy is defined as [*n*..*m*] real-time priorities.
JPI$K_PSX_RR_POLICY	Posix round-robin policy. The priority interval for this policy is defined as [*n*..*m*] real-time priorities.♦

Alpha

prvpol
OpenVMS usage: longword_unsigned
type: longword (unsigned)
access: write only
mechanism: by reference

On Alpha systems, address of a longword into which the previous scheduling policy for the process is written. If the **policy** argument is null, no change in policy is requested and **prvpol** returns the current policy.

The valid priority intervals for specific scheduling policies may change in the future. Applications should, therefore, not use embedded numeric constants for scheduling priority, but should use the appropriate $GETSYI item codes to fetch the legal priority intervals. The application may then dynamically select a priority value that is within the interval. The $GETSYI item codes are:

- SYI$_DEF_PRIO_MAX
- SYI$_DEF_PRIO_MIN
- SYI$_PSXFIFO_PRIO_MAX
- SYI$_PSXFIFO_PRIO_MIN

- SYI$_PSXRR_PRIO_MAX
- SYI$_PSXRR_PRIO_MIN

See the Item Codes section of the $GETSYI service description for more information about these item codes.♦

nullarg

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

Placeholding argument reserved to Digital.

Description

The Set Priority service changes the base priority of the process or, optionally, changes the scheduling policy of the process. The base priority is used to determine the order in which executable processes are to run.

The scheduling policy denotes the following:

- The basic scheduling discipline (FIFO, round-robin, and so forth).
- The preemption/compensation rules by which a running process is descheduled in favor of another process and, ultimately, rescheduled.

A source process may modify the priority or scheduling policy of a target process if *any* of the following are true:

- The source and target processes are in the same job tree.
- The source and target processes have the same UIC.
- The source process has WORLD privilege enabled.
- The source and target processes are in the same process group.

The value to which the priority of a process may be set can be subject to limitations. If the source has ALTPRI privilege enabled, the target may be set to any valid priority. Otherwise, the priority value specified by the source process is compared to the authorized priority of the target process and the smaller of the two values is used as the new base priority of the target process.

If you specify neither the **pidadr** nor the **prcnam** argument, $SETPRI sets the base priority of the calling process.

If the longword at address **pidadr** is the value 0, the PID of the target process is returned.

The base priority of a process remains in effect until specifically changed or until the process is deleted.

To determine the priority set by the $SETPRI service, use the Get Job/Process Information ($GETJPI) service.

Required Access or Privileges

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. If a process does not have ALTPRI privilege, the priority value specified by the source process is compared to the authorized priority of the target process and the smaller of the two values is used as the new base priority of the target process.

Required Quota

None

Related Services

$CANEXH, $CREPRC, $DCLEXH, $DELPRC, $EXIT, $FORCEX, $GETJPI, $GETJPIW, $HIBER, $PROCESS_SCAN, $RESUME, $SETPRN, $SETPRV, $SETRWM, $SUSPND, $WAKE

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$_ILLPOLICY	An invalid scheduling policy was specified.
SS$_ILLPRIPOL	Setting the process to the specified priority and/or policy would result in an illegal policy/priority combination. The illegal combination may occur between the SETPRI policy and priority parameters themselves, or it may occur between either of the parameters and the current policy and/or priority of the target process.
SS$_INCOMPAT	The remote node is running an incompatible version of the operating system.
SS$_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.
SS$_NOSUCHNODE	The process name refers to a node that is not currently recognized as part of the cluster.
SS$_REMRSRC	The remote node has insufficient resources to respond to the request. (Bring this error to the attention of your system manager.)
SS$_UNREACHABLE	The remote node is a member of the cluster but is not accepting requests. (This is normal for a brief period early in the system boot process.)

$SETPRN
Set Process Name

Allows a process to establish or to change its own process name.

Format

SYS$SETPRN [prcnam]

Argument

prcnam

OpenVMS 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

The Set Process Name service allows a process to establish or to change its own process name, which 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).

Required Access or Privileges

None

Required Quota

None

Related Services

$CANEXH, $CREPRC, $DCLEXH, $DELPRC, $EXIT, $FORCEX, $GETJPI, $GETJPIW, $HIBER, $PROCESS_SCAN, $RESUME, $SETPRI, $SETPRV, $SETRWM, $SUSPND, $WAKE

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

Allows a process to change the protection on a page or range of pages.

Format

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

Arguments

inadr
OpenVMS 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. Addresses are adjusted up or down to fall on CPU-specific page boundaries. Only the virtual page number portion of each virtual address is used; the low-order byte-within-page bits are ignored.

If the starting and ending virtual addresses are the same, the protection is changed for a single page.

retadr
OpenVMS 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
OpenVMS 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 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

OpenVMS 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.

Symbol	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

Alpha

OpenVMS Alpha systems convert PRT$C_NA to the next highest protection, kernel-read.♦

If you specify the protection as the value 0, the protection defaults to kernel read only.

prvprt

OpenVMS 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

The Set Protection on Pages service allows a process to change the protection on a page or range of pages.

Required Access or Privileges

None

Required Quota

If a process changes the protection for 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.

Related Services

$ADJSTK, $ADJWSL, $CRETVA, $CRMPSC, $DELTVA, $DGBLSC, $EXPREG, $LCKPAG, $LKWSET, $MGBLSC, $PURGWS, $SETSTK, $SETSWM, $ULKPAG, $ULWSET, $UPDSEC, $UPDSECW

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, less than 0, or 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; an attempt was made to change the protection of a valid global page, of an invalid global noncopy-on-reference page, or a PFN global or private page.
SS$_PAGOWNVIO	The process attempted to change the protection on a page owned by a more privileged access mode.

$SETPRT_64 (Alpha Only)
Set Protection on Pages

On Alpha systems, allows a process to change the protection on a page or range of pages.

This service accepts 64-bit addresses.

Format

SYS$SETPRT_64 start_va_64 ,length_64 ,acmode ,prot ,return_va_64 ,return_length_64 ,return_prot_64

Arguments

start_va_64
OpenVMS usage: address
type: quadword address
access: read only
mechanism: by value

The starting virtual address of the range of pages whose protection is to be changed. The specified virtual address will be rounded down to a CPU-specific boundary.

length_64
OpenVMS usage: byte count
type: quadword (unsigned)
access: read only
mechanism: by value

Length of the virtual address space whose protection is to be changed. The specified length will be rounded up to a CPU-specific page boundary so that it includes all CPU-specific pages in the requested range.

acmode
OpenVMS usage: access_mode
type: longword (unsigned)
access: read only
mechanism: by value

Access mode associated with the call to $SETPRT_64. The **acmode** argument is a longword containing the access mode.

The $PSLDEF macro in STARLET.MLB and the file PSLDEF.H in SYS$STARLET_C.TLB define the following symbols and their values for the four access modes:

Value	Symbolic Name	Access Mode
0	PSL$C_KERNEL	Kernel
1	PSL$C_EXEC	Executive
2	PSL$C_SUPER	Supervisor
3	PSL$C_USER	User

The most privileged access mode used is 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

OpenVMS 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 for MACRO-32 and the include file PRTDEF.H for C define the symbolic names for the protection codes.

return_va_64

OpenVMS usage:	address
type:	quadword address
access:	write only
mechanism:	by 32-bit or 64-bit reference

The lowest process virtual address of the range of pages whose protection was actually changed. The **return_va_64** argument is the 32-bit or 64-bit virtual address of a naturally aligned quadword into which the service returns the virtual address.

return_length_64

OpenVMS usage:	byte count
type:	quadword (unsigned)
access:	write only
mechanism:	by 32-bit or 64-bit reference

The length of the virtual address range whose protection was actually changed. The **return_length_64** argument is the 32-bit or 64-bit virtual address of a naturally aligned quadword into which the service returns the length of the virtual address range in bytes.

return_prot_64

OpenVMS usage:	page_protection
type:	longword (unsigned)
access:	write only
mechanism:	by 32-bit or 64-bit reference

Protection previously assigned to the last page in the range. The **return_prot_64** argument is the 32-bit or 64-bit virtual address of a naturally aligned longword into which $SETPRT_64 writes the protection of this page. The **return_prot_64** argument is useful only when protection for a single page is being changed.

Description

The Set Protection on Pages service allows a process to change the protection on a page or range of pages. For pages in a global section, the new protection can alter only copy-on-reference pages.

If the condition value SS$_ACCVIO is returned by this service, a value *cannot* be returned in the memory locations pointed to by the **return_va_64**, **return_length_64**, and **return_prot** arguments.

If a condition value other than SS$_ACCVIO is returned, the returned address and returned length indicate the pages that were successfully changed before the error occurred. If no pages were changed, the **return_va_64** argument will contain the value -1, and a value *cannot* be returned in the memory location pointed to by the **return_length_64** argument.

Required Privileges

None

Required Quota

If a process changes the protection for any pages in a private section from read-only to read/write, $SETPRT_64 uses the paging file (PGFLQUOTA) quota of the process.

Related Services

$CRETVA_64, $CRMPSC_FILE_64, $CRMPSC_GFILE_64, $CRMPSC_GPFILE_64, $EXPREG_64, $MGBLSC_64

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The **return_va_64** or the **return_length_64** argument cannot be written by the caller.
SS$_EXPGFLQUOTA	The process exceeded its paging file quota while changing a page in a read-only private section to a read/write page.
SS$_IVPROTECT	The specified protection code has a numeric value of 1 or is greater than 15.
SS$_LENVIO	A page in the specified range is not in process private address space.
SS$_NOSUCHPAG	An attempt was made to change the protection on a nonexistent page.
SS$_PAGNOTINREG	A page in the specified range is not within the specified region.
SS$_PAGTYPVIO	A page in the specified range is not in process private address space.
SS$_PAGOWNVIO	The process attempted to change the protection on a page owned by a more privileged access mode.

$SETPRV
Set Privileges

Enables or disables specified privileges for the calling process.

Format

SYS$SETPRV [enbflg] ,[prvadr] ,[prmflg] ,[prvprv]

Arguments

enbflg
OpenVMS usage: boolean
type: longword (unsigned)
access: read only
mechanism: by value

Indicator specifying whether the specified privileges are to be enabled or disabled. The **enbflg** argument is a longword 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
OpenVMS 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 SYS2–4 provides the symbolic name and description of each privilege.

Table SYS2–4 User Privileges

Privilege	Symbolic Name	Description
ACNT	PRV$M_ACNT	Create processes for which no accounting is done
ALLSPOOL	PRV$M_ALLSPOOL	Allocate a spooled device
ALTPRI	PRV$M_ALTPRI	Set (alter) any process priority
AUDIT	PRV$V_AUDIT	Generate audit records
BUGCHK	PRV$M_BUGCHK	Make bugcheck error log entries
BYPASS	PRV$M_BYPASS	Bypass all protection
CMEXEC	PRV$M_CMEXEC	Change mode to executive

(continued on next page)

Table SYS2–4 (Cont.) User Privileges

Privilege	Symbolic Name	Description
CMKRNL	PRV$M_CMKRNL	Change mode to kernel
DETACH	PRV$M_DETACH	Create detached processes
DIAGNOSE	PRV$M_DIAGNOSE	May diagnose devices
DOWNGRADE	PRV$V_DOWNGRADE	May downgrade classification
EXQUOTA	PRV$M_EXQUOTA	May exceed quotas
GROUP	PRV$M_GROUP	Group process control
GRPNAM	PRV$M_GRPNAM	Place name in group logical name table
GRPPRV	PRV$V_GRPPRV	Group access by means of system protection field
IMPORT	PRV$V_IMPORT	Mount a nonlabeled tape volume
LOG_IO	PRV$M_LOG_IO	Perform logical I/O operations
MOUNT	PRV$M_MOUNT	Issue mount volume QIO
NETMBX	PRV$M_NETMBX	Create a network device
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$M_PSWAPM	Change process swap mode
READALL	PRV$V_READALL	Possess read access to everything
SECURITY	PRV$V_SECURITY	May perform security functions
SETPRV	PRV$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

(continued on next page)

Table SYS2–4 (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$V_UPGRADE	May upgrade classification
VOLPRO	PRV$M_VOLPRO	Override volume protection
WORLD	PRV$M_WORLD	World process control

If you do not specify **prvadr** or assign it the value 0, the privileges are not altered.

prmflg
OpenVMS usage: boolean
type: longword (unsigned)
access: read only
mechanism: by value

Indicator specifying whether the privileges are to be affected permanently or temporarily. The **prmflg** argument is a longword 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
OpenVMS 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 assign it the value 0, the previous privilege mask is not returned.

Description

The Set Privileges service enables or disables specified privileges for the calling process.

The operating system 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 $GETJPI (Get Job/Process Information) service and specifying the desired privilege mask or masks as item codes in the **itmlst** argument. You construct the item code for a privilege mask by prefixing the name of the privilege mask with the characters *JPI$_* (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 *OpenVMS DCL Dictionary* for details.

Required Access or Privileges

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.

Required Quota

None

Related Services

$CANEXH, $CREPRC, $DCLEXH, $DELPRC, $EXIT, $FORCEX, $GETJPI, $GETJPIW, $HIBER, $PROCESS_SCAN, $RESUME, $SETPRI, $SETPRN, $SETRWM, $SUSPND, $WAKE

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$_ACCVIO	The privilege mask cannot be read or the previous privilege mask cannot be written by the caller.
SS$_IVSTSFLG	You specified a value other than 1 or 0 in either the **prmflg** argument or the **enblfg** argument.

$SETRWM
Set Resource Wait Mode

Allows a process to specify what action system services should take when system resources required for their execution are unavailable.

Format

SYS$SETRWM [watflg]

Argument

watflg

OpenVMS 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.

The operating system 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 reenabled or until the process is deleted.

Description

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.

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

Required Access or Privileges

None

Required Quota

None

Related Services

$CANEXH, $CREPRC, $DCLEXH, $DELPRC, $EXIT, $FORCEX, $GETJPI, $GETJPIW, $HIBER, $PROCESS_SCAN, $RESUME, $SETPRI, $SETPRN, $SETPRV, $SUSPND, $WAKE

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.

$SETSHLV
Set Automatic Unshelving

Controls whether a process automatically unshelves files.

Format

SYS$SETSHLV [pidadr] ,[prcnam] ,[shlvflg]

Arguments

pidadr

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

Process identification (PID) of the process. The **pidadr** argument is the address of the PID. The **pidadr** argument can only refer to a process running on the local node. You cannot modify a process on a remote node.

You must specify the **pidadr** argument to modify a process whose UIC group number is different from that of the calling process.

prcnam

OpenVMS usage:	process_name
type:	character–coded text string
access:	read only
mechanism:	by descriptor–fixed length string descriptor

Process name of the process. The **prcnam** argument is the address of a character string descriptor pointing to the process name. You identify a process with a 1- to 15-character string.

You can only use the **prcnam** argument to modify a process in the same UIC group as the calling process. To modify a process in another UIC group, you must specify the **pidadr** argument.

shlvflg

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

Value specifying whether automatic unshelving is to be turned on or off. The **shlvflg** argument is a longword containing this value. The value 0 turns automatic unshelving on. The value 1 turns automatic unshelving off.

Description

The Set Automatic Unshelving service controls whether a process automatically unshelves files.

The **pidadr** and **prcnam** default to the current process. If the longword at address **pidadr** is 0, the PID of the target process is returned.

The setting for automatic unshelving is inherited by subprocesses.

The DCL command SET PROCESS/[NO]AUTOUNSHELVE also controls automatic unshelving for a process; refer to the *OpenVMS DCL Dictionary* for details.

Required Access or Privileges

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

- GROUP privilege to modify a process in the same group, unless the target process has the same UIC as the calling process.
- WORLD privilege to modify any process in the system.

Required Quota

None

Related Services

$GETJPI

Condition Values Returned

SS$_WASCLR	The service completed successfully. Automatic unshelving was previously on.
SS$_WASSET	The service completed successfully. Automatic unshelving was previously off.
SS$_ACCVIO	An argument was not accessible by the caller.
SS$_BADPARAM	The **shlvflg** argument was invalid.
SS$_IVLOGNAM	The **prcnam** argument was invalid. The process name string had either 0 characters or more than 15 characters.
SS$_NONEXPR	The specified process did not exist, or the specified process identification was invalid.
SS$_NOPRIV	The caller did not have the privilege to modify other processes.
SS$_REMOTE_PROC	The specified process was not on the local node. The service cannot modify a process on a remote node.

$SETSTK
Set Stack Limits

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]

Arguments

inadr

OpenVMS 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

OpenVMS 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

OpenVMS 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 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.

Required Access or Privileges

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.

Required Quota

None

Related Services

$ADJSTK, $ADJWSL, $CRETVA, $CRMPSC, $DELTVA, $DGBLSC, $EXPREG, $LCKPAG, $LKWSET, $MGBLSC, $PURGWS, $SETPRT, $SETSWM, $ULKPAG, $ULWSET, $UPDSEC, $UPDSECW

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

Allows a process to control whether it can be swapped out of the balance set.

Format

SYS$SETSWM [swpflg]

Argument

swpflg

OpenVMS 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

The Set Process Swap Mode service allows a process to control whether it can be swapped out of the balance set.

When the 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 reenabled 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.

Required Access or Privileges

To change its process swap mode, the calling process must have PSWAPM privilege.

Required Quota

None

Related Services

$ADJSTK, $ADJWSL, $CRETVA, $CRMPSC, $DELTVA, $DGBLSC, $EXPREG, $LCKPAG, $LKWSET, $MGBLSC, $PURGWS, $SETPRT, $SETSTK, $ULKPAG, $ULWSET, $UPDSEC, $UPDSECW

To lock some but not necessarily all process pages into the balance set, use the Lock Pages in Memory ($LCKPAG) service.

For more information, see the chapter on memory management in the *OpenVMS Programming Concepts Manual.*

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

Modifies the user authorization file (UAF) record for a specified user.

Format

SYS$SETUAI [nullarg] ,[contxt] ,usrnam ,itmlst ,[nullarg] ,[nullarg] ,[nullarg]

Arguments

nullarg

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

Placeholding argument reserved to Digital.

contxt

OpenVMS usage:	longword
type:	longword (unsigned)
access:	modify
mechanism:	by reference

A longword used to maintain authorization file context. The **contxt** argument is the address of a longword to receive a $SETUAI context value. On the initial call, this longword should contain the value –1. On subsequent calls, the value of the **contxt** argument from the previous call should be passed back in.

usrnam

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

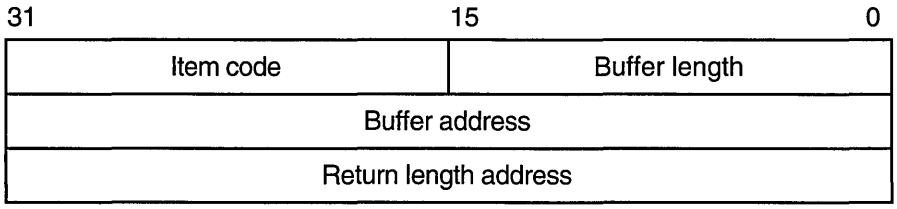
Name of the user whose UAF record is modified. The **usrnam** argument is the address of a descriptor pointing to a character text string containing the user name. The user name string can contain a maximum of 32 alphanumeric characters.

itmlst

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

Item list specifying which information from the specified 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 format of a single item descriptor.



ZK-5186A-GE

The following table defines the item descriptor fields.

Descriptor Field	Definition
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 the buffer length field 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.
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.

The symbolic codes have the following format:

UAI$_code

Item Codes

UAI$_ACCOUNT

When you specify UAI$_ACCOUNT, $SETUAI sets, as a blank-padded 32-character string, the account name of the user.

An account name can include up to 8 characters. Because the account name is a blank-filled string, however, the buffer length field of the item descriptor should specify 32 (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, $SETUAI sets, as an OpenVMS RMS file name component, the name of the command language interpreter used to execute the specified batch job. The file specification set assumes the device name and directory SYS$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, in pages (on VAX systems) or pagelets (on Alpha systems), 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$_DIALUP_ACCESS_P
When you specify UAI$_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$_DIOLM
When you specify UAI$_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$_ENCRYPT
When you specify UAI$_ENCRYPT, $SETUAI sets one of the values shown in the following table to identify the encryption algorithm for the primary password.

Symbolic Name	Description
UAI$C_AD_II	Uses a CRC algorithm and returns a longword hash value. It was used in VAX/VMS releases prior to Version 2.0.
UAI$C_PURDY	Uses a Purdy algorithm over salted input. It expects a blank-padded user name and returns a quadword hash value. This algorithm was used during VAX/VMS Version 2.0 field test.
UAI$C_PURDY_V	Uses the Purdy algorithm over salted input. It expects a variable length user name and returns a quadword hash value. This algorithm was used in VMS releases prior to Version 5.4.

Symbolic Name	Description
UAI$C_PURDY_S	Uses the Purdy algorithm over salted input. It expects a variable length user name and returns a quadword hash value. This is the current algorithm that the operating system uses for all new password changes.
UAI$C_PREFERED_ ALGORITHM	Represents the latest encryption algorithm that the operating system uses to encrypt new passwords. Currently, it equates to UAI$C_PURDY_S. Digital recommends that you use this symbol in source modules.

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, $SETUAI sets one of the following values, indicating the encryption algorithm for the secondary password. Refer to the UAI$_ENCRYPT item code for a description of the algorithms.

UAI$C_AD_II
UAI$C_PURDY
UAI$C_PURDY_V
UAI$C_PURDY_S
UAI$C_PREFERED_ALGORITHM

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.

Symbol	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$V_DISFORCE_PWD_CHANGE	User will not be forced to change expired passwords at login.
UAI$V_DISIMAGE	User cannot issue the RUN or MCR commands or use the foreign command mechanism in DCL.
UAI$V_DISMAIL	Announcement of new mail is suppressed.
UAI$V_DISPWDDIC	Automatic checking of user-selected passwords against the system dictionary is disabled.
UAI$V_DISPWDHIS	Automatic checking of user-selected passwords against previously used passwords is disabled.
UAI$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$V_GENPWD	User is required to use generated passwords.
UAI$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$V_PWD2_EXPIRED	Secondary password is expired.
UAI$V_RESTRICTED	User is limited to operating under a restricted account. Clear the CAPTIVE flag (UAI$V_CAPTIVE), if set, before setting the RESTRICTED flag. (See the *Security Guide* for a description of restricted and captive accounts.)

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$_LASTLOGIN_I
When you specify UAI$_LASTLOGIN_I, $SETUAI sets, as a quadword absolute time value, the date of the last interactive login.

UAI$_LASTLOGIN_N
When you specify UAI$_LASTLOGIN_N, $SETUAI sets, as a quadword absolute time value, the date of the last noninteractive login.

UAI$_LGICMD
When you specify UAI$_LGICMD, $SETUAI sets, as an OpenVMS 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$_LOGFAILS
When you specify UAI$_LOGFAILS, $SETUAI sets 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, $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$_PASSWORD

When you specify UAI$_PASSWORD, $SETUAI sets the specified plaintext string as the primary password for the user and updates the primary password change date. You must have SYSPRV privilege to set passwords for any user account (including your own).

The UAI$_PASSWORD and UAI$_PASSWORD2 item codes provide the building blocks for designing a site-specific SET PASSWORD utility. Note that if you create such a utility, you should also set the LOCKPWD bit in the user authorization file (UAF) to prevent users from using the DCL command SET PASSWORD and to prevent the LOGINOUT process from forcing password changes. If you create a site-specific SET PASSWORD utility, install the utility with SYSPRV privilege.

You must adhere to the following guidelines when specifying a password with UAI$_PASSWORD or UAI$_PASSWORD2:

- The password must meet the minimum password length defined for the user account
- The password cannot exceed 32 characters in length
- The password must be different from the previous password.

To clear the primary password, specify the value 0 in the buffer length field.

UAI$_PASSWORD2

When you specify UAI$_PASSWORD2, $SETUAI sets the specified plaintext string as the secondary password for the user and updates the secondary password change date. You must have SYSPRV privilege to set passwords for any user account (including your own).

To clear the secondary password, specify the value 0 in the buffer length field.

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, in pages (on VAX systems) or pagelets (on Alpha systems), 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 that 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$_PWD_DATE
When you specify UAI$_PWD_DATE, $SETUAI sets, 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).

A value of –1 indicates that the password could be marked as preexpired.

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 delta 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).

A quadword of 0 means that none of the password mechanisms will take effect.

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$_PWD2_DATE
When you specify UAI$_PWD2_DATE, $SETUAI sets, as a quadword absolute time value, the last date 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).

A value of –1 indicates that the password could be marked as preexpired.

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$_SALT
When you specify UAI$_SALT, $SETUAI sets the salt field of the user's record to the value you provide. The salt value is used in the operating system hash algorithm to generate passwords. $SETUAI does not generate a new salt value for you.

Because this decimal number is a word in length, the buffer length field in the item descriptor should specify 2 (bytes).

By copying the item codes UAI$_SALT, UAI$_ENCRYPT, UAI$_PWD, UAI$_PWD_DATE, and UAI$_FLAGS, a site-security administrator can construct a utility that propagates password changes throughout the network. Note, however, that Digital does not recommend using the same password on more than one node in a network.

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). For the format of the UIC, see the *Security Guide*.

UAI$_USER_DATA

When you specify UAI$_USER_DATA, $SETUAI sets up to 255 bytes of information in the user data area of the system user authorization file (SYSUAF). This is the supported method for modifying the user data area of the SYSUAF. Digital no longer supports direct user modification of the SYSUAF.

To clear all the information in the user data area of the SYSUAF, specify $SETUAI with a buffer length field of 0.

UAI$_WSEXTENT

When you specify UAI$_WSEXTENT, $SETUAI sets the working set extent, in pages (on VAX systems) or pagelets (on Alpha systems), specified for the specified 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, in pages (on VAX systems) or pagelets (on Alpha systems), 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 Set User Authorization Information service is used to modify the user authorization file (UAF) record for a specified user.

Required Access or Privileges

The following list describes 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. Note, however, that you cannot change a UAF record whose UIC matches exactly the requester's UIC. Group managers with GRPPRV privilege are limited in the extent to which they can modify the UAF records of users in the same group; values such as privileges and quotas can be changed only if the modification does not exceed the values set in a group manager's UAF record.
- No privilege—Does not allow access to any UAF record.

Required Quota

None

Related Services

$GETUAI

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$_BADPARAM	The function code is invalid; the item list contains an invalid item code; a buffer descriptor has an invalid length; or the reserved parameter has a nonzero value.
SS$_NOGRPPRV	The user does not have the privileges required to modify the authorization information for other members of the UIC group.
SS$_NOSYSPRV	The user does not have the privileges required to modify the authorization information associated with the user or for users outside of the user's UIC group.
RMS$_RSZ	The UAF record is smaller than required; the caller's SYSUAF is likely corrupt.

This service can also return OpenVMS 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 *OpenVMS Record Management Services Reference Manual.*

$SET_RESOURCE_DOMAIN
Set Resource Domain

Controls the association between a calling process and resource domains.

Format

SYS$SET_RESOURCE_DOMAIN func ,rsdm_id ,domain_number ,[nullarg] ,[access] ,[acmode]

Arguments

func

OpenVMS usage: function_code
type: longword (unsigned)
access: read only
mechanism: by value

Function code specifying the action that $SET_RESOURCE_DOMAIN is to perform. The **func** argument is a longword containing this function code. See the Function Codes section for a description of $SET_RESOURCE_DOMAIN function codes.

rsdm_id

OpenVMS usage: longword
type: longword (unsigned)
access: write only to join, read only to leave
mechanism: by reference

Resource domain identification. The **rsdm_id** argument is the address of a longword specifying the association of the calling process with the resource domain.

The RSDM$_JOIN_DOMAIN function returns a resource domain identification. The RSDM$_LEAVE function requires the **rsdm_id** argument as input to specify which resource domain association the process is leaving.

The resource domain identification may be used as input to the $ENQ and $ENQW system services.

domain_number

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

Domain number that identifies the resource domain. The **domain_number** argument is a longword value containing the resource domain number.

The **domain_number** argument is required for the RSDM$_JOIN_DOMAIN function but ignored for the RSDM$_LEAVE function.

nullarg

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

Placeholder reserved to Digital. You must specify 0.

access

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

Types of access desired when using the lock management services within the resource domain. The **access** argument is a longword bit mask specifying the access types required; these can include read, write, and lock. The following table lists the symbols that the $RSDMDEF macro defines, their descriptions, and the lock management system services that may require each type of access.

Symbol	Access Description	System Service
RSDM$M_READ	Read lock value blocks	$DEQ, $ENQ, $ENQW, $GETLKI, $GETLKIW
RSDM$M_WRITE	Write lock value blocks	$DEQ, $ENQ, $ENQW,
RSDM$M_LOCK	Take locks	$ENQ, $ENQW

The service grants the desired access, provided your process has the necessary access rights to the resource domain. If you do not specify the **access** argument or if you specify 0, $SET_RESOURCE_DOMAIN attempts to access the domain in the following order:

1. Read, write, lock
2. Read, lock
3. Write, lock
4. Lock

The access attempt terminates with the first success.

The **access** argument defaults to 0. It is ignored for the RSDM$_LEAVE function.

acmode

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

Access mode requested for the association to the resource domain. The most privileged access mode granted is the access mode of the caller. Locks may not be taken from access modes less privileged than the access mode of the association.

The **acmode** argument is a longword containing the access mode. The $PSLDEF macro defines the following symbols for the access modes.

Symbolic Name	Access Mode	Privilege Rank
PSL$C_KERNEL	Kernel	High
PSL$C_EXEC	Executive	–
PSL$C_SUPER	Supervisor	–
PSL$C_USER	User	Low

The **acmode** argument is optional for the RSDM$_JOIN_DOMAIN function. If you do not specify the **acmode** argument, the access mode is set to the access mode of the calling process. The **acmode** argument is ignored for the RSDM$_ LEAVE function.

Function Codes

RSDM$_JOIN_DOMAIN

A process has the option of forming multiple associations with one or more resource domains. Each association can have different access rights to the resource domain, such as to read lock value blocks or to write lock value blocks. This request sets up a new association with a resource domain.

$SET_RESOURCE_DOMAIN verifies the desired access against the security profile of the resource domain. If the desired access is allowed, a new association to the resource domain is created, and a resource domain identification for the association is returned.

This function code returns the following condition values:

SS$_NORMAL
SS$_BADPARAM
SS$_EXQUOTA
SS$_INSFMEM
SS$_NOOBJSRV
SS$_NOPRIV

RSDM$_LEAVE

This operation requests that a process end an association with a resource domain. A process must leave a resource domain association in the same mode as, or in a more privileged mode than, the mode in which it joined the resource domain.

Before a process can end its association with a resource domain, it must release all locks taken using that association.

This function code returns the following condition values:

SS$_NORMAL
SS$_BADPARAM
SS$_IVMODE
SS$_RSDM_ACTIVE
SS$_RSDMNOTFOU

Description

The Set Resource Domain system service enables a process to use the lock management system services $DEQ, $ENQ, $ENQW, $GETLKI, and $GETLKIW.

The lock management services enable processes with the appropriate access rights to take and release locks on resource names and to perform other functions related to lock management. Applications use resource names to represent resources to which they want to synchronize access. A resource domain is a namespace for resource names. A process must join a resource domain to take and release locks and to read and write value blocks associated with resources in that resource domain.

When a process requests to join a resource domain, $SET_RESOURCE_DOMAIN performs an access check. After $SET_RESOURCE_DOMAIN verifies the desired access to the resource domain, the service creates an association between the resource domain and the calling process. The association is represented by a resource domain identification. A process can request different types of access to the same resource domain; the type of access is a characteristic of the association with the resource domain. Each time a process joins a resource domain, a new association is created. Processes use their resource domain identifications when using $ENQ or $ENQW to request a new lock.

The service can grant the following three types of access to resource domains:

- The right to read lock value blocks
- The right to write lock value blocks
- The right to take and release locks

Required Access or Privileges

None

Required Quota

$SET_RESOURCE_DOMAIN uses system dynamic memory, which uses BYTLM quota, for the creation of the resource domain data structures.

Related Services

$DEQ, $ENQ, $ENQW, $GETLKI, $GETLKIW

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_BADPARAM	The **func**, the **domain_number**, or the **rsdm_id** argument was specified incorrectly.
SS$_EXQUOTA	The caller has insufficient BYTLM quota.
SS$_INSFMEM	There is insufficient memory to join the resource domain.
SS$_IVMODE	An attempt was made to leave an association created by a more privileged access mode.
SS$_NOOBJSRV	The audit server process, which maintains the security profile for resource domains, is not running. The process access rights to the domain cannot be determined, so access is denied.

SS$_NOPRIV	Access to the resource domain was denied.
SS$_RSDM_ACTIVE	Unable to leave the resource domain because there are locks still associated with this resource domain.
SS$_RSDMNOTFOU	The resource domain was not found.

$SET_SECURITY
Set Security Characteristics

Modifies the security characteristics of a protected object.

Format

SYS$SET_SECURITY [clsnam] ,[objnam] ,[objhan] ,[flags] ,[itmlst] ,[contxt] ,[acmode]

Arguments

clsnam

OpenVMS usage:	char_string
type:	character-coded text string
access:	read only
mechanism:	by descriptor

Name of the object class. The **clsnam** argument is the address of a descriptor pointing to a string that contains the name of the object class. The following is a list of the protected object class names:

CAPABILITY
COMMON_EVENT_CLUSTER
DEVICE
FILE
GROUP_GLOBAL_SECTION
LOGICAL_NAME_TABLE
QUEUE
RESOURCE_DOMAIN
SECURITY_CLASS
SYSTEM_GLOBAL_SECTION
VOLUME

objnam

OpenVMS usage:	char_string
type:	character-coded text string
access:	read only
mechanism:	by descriptor

Name of the protected object whose associated security profile is going to be retrieved. The **objnam** argument is the address of a descriptor pointing to a string containing the name of the protected object.

The format of an object name is class specific. The following table lists object names and describes their formats.

Object Class	Object Name Format
CAPABILITY	A character string. Currently, the only capability object is VECTOR.
COMMON_EVENT_CLUSTER	Name of the event flag cluster, as defined in the Associate Common Event Flag Cluster ($ASCEFC) system service.

Object Class	Object Name Format
DEVICE	Standard device specification, described in the *OpenVMS User's Manual.*
FILE	Standard file specification, described in the *OpenVMS User's Manual.*
GROUP_GLOBAL_SECTION	Section name, as defined in the Create and Map Section ($CRMPSC) system service.
LOGICAL_NAME_TABLE	Table name, as defined in the Create Logical Name Table ($CRELNT) system service.
QUEUE	Standard queue name, as described in the Send to Job Controller ($SNDJBC) system service.
RESOURCE_DOMAIN	An identifier or octal string enclosed in brackets.
SECURITY_CLASS	Any class name shown in the Object Class column of this table, or a class name followed by a period (.) and the template name. Use the DCL command SHOW SECURITY to display possible template names.
SYSTEM_GLOBAL_SECTION	Section name, as defined in the Create and Map Section ($CRMPSC) system service.
VOLUME	Volume name or name of the device on which the volume is mounted.

objhan

OpenVMS usage: object_handle
type: longword (unsigned)
access: read only
mechanism: by reference

Data structure identifying the object to address. The **objhan** argument is an address of a longword containing the object handle. You can use the **objhan** argument as an alternative to the **objnam** argument; for example, a channel number clearly specifies the file open on the channel and can serve as an object handle. The following table shows the format of the object classes.

Object Class	Object Handle Format
COMMON_EVENT_CLUSTER	Event flag number
DEVICE	Channel number
FILE	Channel number
RESOURCE_DOMAIN	Resource domain identifier
VOLUME	Channel number

flags

OpenVMS usage: flags
type: mask_longword
access: read only
mechanism: by value

Mask specifying processing options. The **flags** argument is a longword bit vector wherein a bit, when set, specifies the corresponding option. The **flags** argument requires the **contxt** argument. The following table describes each flag.

Symbolic Name	Description
OSS$M_LOCAL	Do not update the master profile for the specified object. This flag allows you to call $SET_SECURITY several times to modify a local copy of a profile; once the modifications are satisfactory, you can clear the OSS$M_LOCAL flag, set the OSS$M_RELCTX flag, and have $SET_SECURITY update the master profile. The flag applies only to calls made with the **contxt** argument.
OSS$M_RELCTX	Release the context structure at the completion of this request.

The $OSSDEF macro defines symbolic names for the flag bits. You construct the **flags** argument by specifying the symbolic names of each desired option.

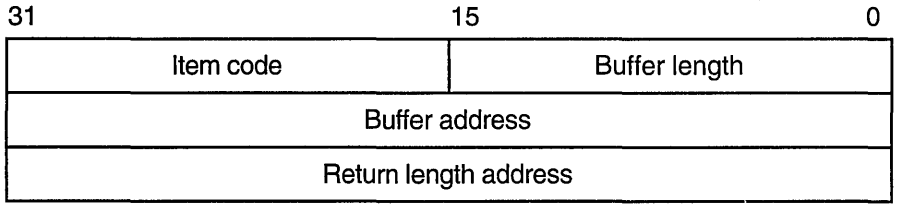
itmlst

OpenVMS 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 modified. 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.

With the item list, the user modifies the protected object's characteristics. The user defines which security characteristics to modify. If this argument is not present, only the **flags** argument is processed. Without the **itmlst** argument, you can *only* manipulate the security profile locks or release **contxt** resources.

The following data structure depicts the format of a single item descriptor.



ZK-5186A-GE

The following table defines the item descriptor fields.

Descriptor Field	Definition
Buffer length	A word containing an integer specifying the length (in bytes) of the buffer from which $SET_SECURITY is to read 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, $SET_SECURITY truncates the data.
Item code	A word containing a symbolic code specifying the item of information that $SET_SECURITY is to modify. The $OSSDEF macro defines these codes. A description of each item code is given in the Item Codes section.
Buffer address	A longword containing the address of the buffer from which $SET_SECURITY is to read the information.
Return length address	Not used.

contxt

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

Value used to maintain protected object processing context when dealing with a single protected object across multiple $GET_SECURITY/$SET_SECURITY calls. Whenever the context value is nonzero, the class name, object name, or object handle arguments are disregarded. An input value of 0 indicates that a new context should be established.

Because an active context block consumes process memory, be sure to release the context block by setting the RELCTX flag when the profile processing is complete. $SET_SECURITY sets the context argument to 0 once the context is released.

acmode

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

Access mode to be used in the object protection check. 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** with the caller's access mode and uses the least privileged mode. The access modes are defined in the system macro $PSLDEF library. Digital recommends that this argument be omitted (passed as zero).

Item Codes

The following table provides a summary of item codes that are valid as an item descriptor in the **itmlst** argument. The table lists the $SET_SECURITY item codes and gives a corresponding description. Complete descriptions of each item code are provided after the table.

Item Code	Description
OSS$_ACL_ADD_ENTRY	Adds an access control entry (ACE).
OSS$_ACL_DELETE	Deletes all unprotected ACEs in an ACL.
OSS$_ACL_DELETE_ALL	Deletes the ACL, including protected ACEs.
OSS$_ACL_DELETE_ENTRY	Deletes an ACE.
OSS$_ACL_FIND_ENTRY	Locates an ACE.
OSS$_ACL_FIND_NEXT	Positions the next ACE.
OSS$_ACL_FIND_TYPE	Locates an ACE of the specified type.
OSS$_ACL_MODIFY_ENTRY	Replaces an ACE at the current position.
OSS$_ACL_POSITION_BOTTOM	Sets a marker that points to the end of the ACL.
OSS$_ACL_POSITION_TOP	Sets a marker that points to the beginning of the ACL.
OSS$_OWNER	Sets the UIC or general identifier of the object's owner.
OSS$_PROTECTION	Sets the protection code of the object.

OSS$_ACL_ADD_ENTRY
When you specify OSS$_ACL_ADD_ENTRY, $SET_SECURITY adds an access control entry (ACE) pointed to by the buffer address so that it is front of the current ACE in the access control list (ACL). See OSS$_ACL_POSITION for more information on explicit access control list positioning.

OSS$_ACL_DELETE
When you specify OSS$_ACL_DELETE, $SET_SECURITY deletes all unprotected ACEs in an ACL.

OSS$_ACL_DELETE_ALL
When you specify OSS$_ACL_DELETE_ALL, $SET_SECURITY deletes an entire ACL, including protected ACEs.

OSS$_ACL_DELETE_ENTRY
When you specify OSS$_ACL_DELETE_ENTRY, $SET_SECURITY deletes an ACE pointed to by the buffer address or, if the buffer address is specified as 0, the ACE at the current position.

OSS$_ACL_FIND_ENTRY
When you specify OSS$_ACL_FIND_ENTRY, $SET_SECURITY locates an ACE pointed to by the buffer address. OSS$_ACL_FIND_ENTRY sets the position within the ACL for succeeding ACL operations; for example, for a deletion or modification of the ACE. If the buffer address is 0, it returns SS$_ACCVIO.

OSS$_ACL_FIND_NEXT
When you specify OSS$_ACL_FIND_NEXT, $SET_SECURITY advances the current position to the next ACE in the ACL.

OSS$_ACL_FIND_TYPE
When you specify OSS$_ACL_FIND_TYPE, $SET_SECURITY returns an ACE of a particular type if there is one in the buffer pointed to by the buffer address. OSS$_ACL_FIND_TYPE sets the position within the ACL for succeeding ACL operations. If the buffer address is 0, it returns SS$_ACCVIO.

OSS$_ACL_MODIFY_ENTRY
When you specify OSS$_ACL_MODIFY_ENTRY, $SET_SECURITY replaces an ACE at the current position with the ACE pointed to by the buffer address.

OSS$_ACL_POSITION_BOTTOM
When you specify OSS$_ACL_POSITION_BOTTOM, $SET_SECURITY sets the ACL position to point to the bottom of the ACL.

OSS$_ACL_POSITION_TOP
When you specify OSS$_ACL_POSITION_TOP, $SET_SECURITY sets the ACL position to point to the top of the ACL.

OSS$_OWNER
When you specify OSS$_OWNER, $SET_SECURITY sets the owner UIC of the selected object to the value in the buffer. The buffer size must be 4 bytes.

OSS$_PROTECTION
When you specify OSS$_PROTECTION, $SET_SECURITY sets the selected object's protection code to the value in the buffer. The buffer size must be 2 bytes.

Description

The Set Security service modifies the security characteristics of a protected object. Security characteristics include such information as the protection code, the owner, and the access control list (ACL). The security management services, $SET_SECURITY and $GET_SECURITY, maintain a single master copy of a profile for every protected object in a VMScluster system. They also ensure that only one process at a time can modify an object's security profile.

When you call $SET_SECURITY, the service performs the following steps:

1. It selects the specified protected object.
2. It fetches a local copy of the object's security profile, unless the service is operating on an existing context.
3. It modifies the local profile.
4. It updates the master copy of the profile if the local flag is clear and there was no error.
5. It deletes the local copy of the profile and returns if RELCTX is specified or if no context is specified.

There are different ways of identifying which protected object $SET_SECURITY should process:

- Whenever the **contxt** argument has a nonzero value, $SET_SECURITY uses the context to select the object and ignores the class name, object name, and object handle.

- With some types of objects, such as a file or a device, it is possible to select an object on the basis of its **objhan** and **clsnam** values.
- When the **clsnam** and **objnam** arguments are provided, $SET_SECURITY uses an object's class name and object name to select the object.

The context for a security management operation can be established through either $GET_SECURITY or $SET_SECURITY. Whenever the context is set by one service, the other service can use it provided the necessary locks are being held. A caller to $GET_SECURITY needs to set the write lock flag (OSS$M_WLOCK) to inspect a profile value, maintain the lock on the object's profile, and then modify some value through a call to $SET_SECURITY.

There are many situations in which the **contxt** argument is essential. By establishing a context for an ACL operation, for example, a caller can retain an ACL position across calls to $GET_SECURITY so that a set of ACEs can be read and modified sequentially. A security context is released by a call to $SET_SECURITY or $GET_SECURITY that sets the OSS$M_RELCTX flag. Once the context is deleted, the user-supplied context longword is reset to 0.

Required Access or Privileges

Control access to the object is required.

Required Quota

None

Related Services

$GET_SECURITY

Condition Values Returned

SS$_NORMAL	The service completed successfully
SS$_ACCVIO	The parameter cannot be read and the buffer cannot be written.
SS$_BADPARAM	You specified an invalid object, attribute code, or item size.
SS$_INSFARG	The **clsnam** and **objnam** arguments are not specified, the **clsnam** and **objhan** arguments are not specified, or the **contxt** argument is not specified.
SS$_INVBUFLEN	The buffer size for one of the item codes was invalid.
SS$_INVITMCLS	The item code that you specified is not supported for the class.
SS$_MMATORB	The attempted update cannot be performed. The object profile was changed by another process.
SS$_NOCLASS	The named object class does not exist.
SS$_OBJLOCKED	The selected object is currently write locked.

$SHOW_INTRUSION
Show Intrusion Information

Searches for and returns information about records in the intrusion database matching the caller's specifications.

Format

SYS$SHOW_INTRUSION user_criteria ,intruder ,intruder_len ,breakin_block ,[flags] ,[context]

Arguments

user_criteria

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

Description of intruder or suspect. The **user_criteria** argument is the address of a character-string descriptor pointing to a buffer containing the user criteria to match an intrusion record's user specification in the intrusion database.

The **user_criteria** argument is a character string of between 1 and 1058 bytes containing characters to match the user specification on records in the intrusion database.

A user specification is any combination of the suspect's or intruder's source node name, source user name, source DECnet for OpenVMS address, local failed user name, local terminal, or the string UNKNOWN. The user specification for an intrusion record is based on the input to the $SCAN_INTRUSION service and the settings of the LGI system parameter. For more information, see the *OpenVMS Guide to System Security*.

Wildcards are allowed for the **user_criteria** argument. For more information about using wildcards to scan the intrusion database, see the Description section.

intruder

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

User specification of the matched intruder or suspect record in the intrusion database. The **intruder** argument is the address of a character-string descriptor pointing to a buffer to receive the user specification of the matched record in the intrusion database.

The **intruder** argument is a 1058-byte string that will receive the user specification of a record in the intrusion database that matches the specifications in the **user_criteria** and **flags** arguments.

intruder_len

OpenVMS usage:	string length
type:	longword (unsigned)
access:	write only
mechanism:	by reference

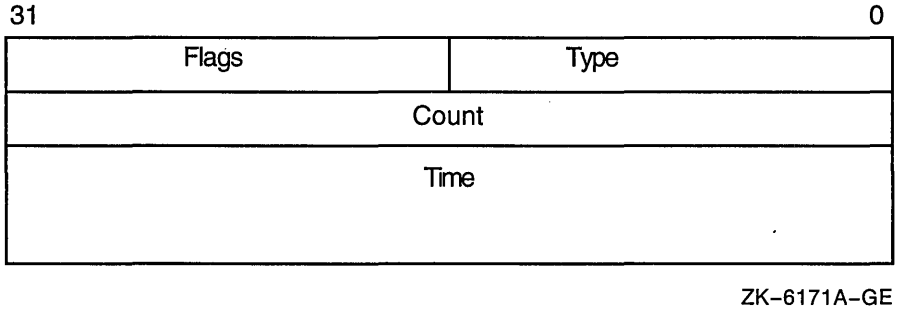
Length of returned string in the intrusion buffer. The **intruder_len** argument is the address of a longword to receive the length of the returned intrusion buffer.

The possible range of the **intruder_len** argument is 0 to 1058 bytes. If the longword specified by the argument contains a 0 after the call to the service, either the service did not find a record that matched the user criteria in the intrusion database, or there are no more matching items in the intrusion database.

breakin_block

OpenVMS usage:	record
type:	block of 2 words (unsigned), 1 longword (unsigned), and 1 quadword (unsigned)
access:	write only
mechanism:	by reference

Block to receive various information in the intrusion database about a record matching the user criteria. The **breakin_block** argument is the address of a structure with the following format.



ZK-6171A-GE

The following table defines the break-in block fields.

Field	Description
Type	Unsigned word containing the type of the matched record. The possible values for the type field are TERM_USER, TERMINAL, USERNAME, and NETWORK. These constants are defined in $CIADEF in STARLET.
Flags	Boolean set to TRUE (1) if the matched record is an intruder. If the value is set to FALSE (0), the matched record is only a suspect.
Count	Unsigned longword containing the number of login failures or break-in attempts made by the specified intruder or suspect.
Time	Quadword time format indicating the time when the record will expire.

flags

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

Type of records in the intrusion database about which information is to be returned. The **flags** argument is a longword bit mask wherein each bit corresponds to an option.

Each option has a symbolic name. The $CIADEF macro defines the following valid names.

Symbolic Name	Description
CIA$M_ALL	All records will be shown. If the **flags** argument is omitted, this value is assumed.
CIA$M_INTRUDERS	Only intruder records matching the criteria specified by the **user_criteria** argument will be returned. The value of the flag field in the break-in block will always be 1.
CIA$M_SUSPECTS	Only suspect records matching the criteria specified by the **user_criteria** argument will be returned. The value of the flag field in the break-in block will always be 0.

Each of these options is mutually exclusive.

context

OpenVMS usage: context
type: longword (unsigned)
access: write only
mechanism: by reference

Context information to keep between related calls to the $SHOW_INTRUSION service. The **context** argument is the address of a longword that receives a context from the service.

The initial value contained in the unsigned longword pointed to by the **context** argument must be 0. The contents of the unsigned longword must not be changed after the service has set its value. If the contents of the **context** argument are changed between calls to the service, SS$_BADCONTEXT will be returned.

Contexts become invalid after one-half hour of non-use. This means that if you call the $SHOW_INTRUSION service with a wildcard in the **user_criteria** argument and do not call the service to get the next matching record within one-half hour, the context becomes invalid. If the context has become invalid, you must restart your search of the intrusion database from the beginning by resetting the context to 0.

Description

The Show Intrusion service returns information about records in the intrusion database that match the criteria you specify.

You can retrieve information about multiple records in the intrusion database by specifying wildcards for the **user_criteria** argument. For example, specifying an asterisk (*) for the **user_criteria** argument and CIA$M_ALL_RECORDS for the **flags** argument will return information about all records in the database. Specifying TTA4* for the **user_criteria** argument and CIA$M_SUSPECTS_ ONLY for the **flags** argument will return information about all suspects who have had failures on terminal TTA4.

If you specify a wildcard string for the **user_criteria** argument, you must also include a **context** argument. Because the service can only return information about one intrusion record at a time, you must call the service repeatedly to retrieve information about more than one record. The service will return SS$_ NOMOREITEMS when information about all of the matching records has been returned. No intrusion information is returned from the call that returns SS$_ NOMOREITEMS.

Required Access or Privileges

SECURITY privilege is required.

Required Quota

None

Related Services

$DELETE_INTRUSION, $SCAN_INTRUSION

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The **user_criteria** or **context** argument cannot be read, or the **intruder**, **intruder_len**, **breakin_block**, or **context** argument cannot be written.
SS$_BADBUFLEN	The length of one of the specified arguments is out of range.
SS$_BADCONTEXT	The **context** argument did not contain a 0 on the first call to the service. The **context** argument's value changed between consecutive calls to the service.
SS$_BADPARAM	An invalid value was specified in the **flags** argument, or mutually exclusive options were specified in the **flags** argument.
SS$_NOMOREITEMS	All items matching the specified criteria have been returned.
SS$_NOSECURITY	The caller does not have SECURITY privilege.

This service can also return any of the following messages passed from the security server:

SECSRV$_ NOSUCHINTRUDER	No records matching the specified criteria were found in the intrusion database.
SECSRV$_ SERVERNOTACTIVE	The security server is not currently active. Try the request again later.

$SIGNAL_ARRAY_64
Signal Array

Returns the address of a 64-bit signal array. A 32-bit signal array and a mechanism array are passed to a condition handler when it is called. $SIGNAL_ARRAY_64 provides the address of the 64-bit signal array, which might be required for programs that use 64-bit address space.

This service accepts 64-bit addresses.

Format

$SIGNAL_ARRAY_64 mcharg, sigarg_64

Arguments

mcharg

OpenVMS usage:	mechanism array
type:	vector quadword (unsigned)
access:	read only
mechanism:	by 32-bit or 64-bit reference, array reference

The mechanism array. The **mcharg** argument is the 32-bit or 64-bit address of this array, which was passed to the condition handler. $SIGNAL_ARRAY_64 uses this structure to determine the 64-bit signal array address.

sigarg_64

OpenVMS usage:	64-bit signal array
type:	vector quadword (unsigned)
access:	write only
mechanism:	by 32-bit or 64-bit reference, array reference

The 32-bit or 64-bit address of the 64-bit signal array is returned in this argument.

Description

$SIGNAL_ARRAY_64 provides the address of the 64-bit version of the signal array for condition handlers that need it. It is normally needed only by applications that use 64-bit address space and want to handle errors involving addresses in that region.

For example, if an access violation occurs on a 64-bit address, the 32-bit signal array passed to the handler will contain only the low 32 bits of the effective address, since each entry is a longword. The 64-bit signal array, which can be obtained using this service, contains quadword entries, so the 64-bit address can be fully expressed.

Required Access or Privileges

None.

Required Quota

None.

Related Services

$PUTMSG, which accepts either a 32-bit or 64-bit signal array as an argument.

Condition Values Returned

SS$_NORMAL	The service completed succesfully.
SS$_ACCVIO	The **sigarg_64** argument cannot be written.
SS$_BADPARAM	The **mcharg** argument is not a mechanism array in the expected format.

$SNDERR
Send Message to Error Logger

Writes a user-specified message to the system error log file, preceding it with the date and time.

Format

SYS$SNDERR msgbuf

Argument

msgbuf

OpenVMS 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

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. The $SNDERR service requires system dynamic memory.

Required Access or Privileges

To send a message to the error log file, the calling process must have BUGCHK privilege.

Required Quota

None

Related Services

$ALLOC, $ASSIGN, $BRKTHRU, $BRKTHRUW, $CANCEL, $CREMBX, $DALLOC, $DASSGN, $DELMBX, $DEVICE_SCAN, $DISMOU, $GETDVI, $GETDVIW, $GETMSG, $GETQUI, $GETQUIW, $INIT_VOL, $MOUNT, $PUTMSG, $QIO, $QIOW, $SNDJBC, $SNDJBCW, $SNDOPR

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

Creates, stops, and manages queues and the batch and print jobs in those queues. The $SNDJBC service completes asynchronously; to synchronize the completion of most operations, use the Send to Job Controller and Wait ($SNDJBCW) service.

Format

SYS$SNDJBC [efn] ,func [,nullarg] [,itmlst] [,iosb] [,astadr] [,astprm]

Arguments

efn

OpenVMS 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

OpenVMS 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 can 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.

nullarg

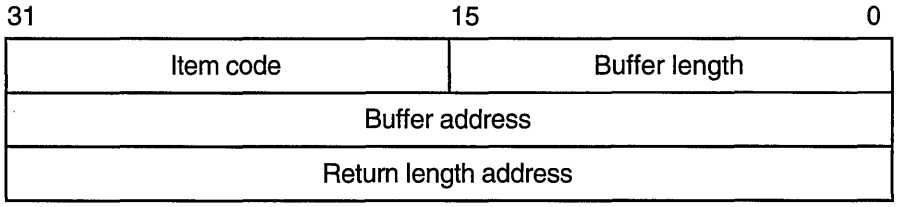
OpenVMS usage:	null_arg
type:	longword (unsigned)
access:	read only
mechanism:	by value

Placeholding argument reserved to Digital.

itmlst

OpenVMS 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–5186A–GE

The following table defines the item descriptor fields.

Descriptor Field	Definition
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 symbol names.
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 to receive the length (in bytes) of information returned by $SNDJBC. If you specify this address as 0, no length is returned.

The item codes' symbolic names 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. The default value for the Boolean item codes is false. For all Boolean item codes, the buffer length, buffer address, and return length fields of the item descriptor must be 0.

- Input value item code. Input value item codes specify an input value to be used by $SNDJBC. The buffer length and buffer address fields of the item descriptor must be nonzero; the return length field must be 0. 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. The buffer length and buffer address fields of the item descriptor must be nonzero; the return length field can be 0 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 uppercase alphabetic characters, lowercase alphabetic characters (which are converted to uppercase), numeric characters, the dollar sign ($), and the underscore (_).

iosb

OpenVMS 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 $SNDJBC can be an error return from a system service or an OpenVMS RMS service that is used in executing the request. For the SJC$_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 in the Condition Values Returned in the I/O Status Block section.

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

OpenVMS usage:	ast_procedure
type:	procedure value
access:	call without stack unwinding
mechanism:	by reference

AST service routine to be executed when $SNDJBC completes. The **astadr** argument is the address of this routine.

If specified, the AST routine executes at the same access mode as the caller of $SNDJBC.

astprm

OpenVMS 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.

Function Codes

This section describes the various function codes that are applicable to the $SNDJBC system service.

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 one of the following input value item codes:

SJC$_ENTRY_NUMBER
SJC$_QUEUE

You must specify the following input value item code for batch jobs:

SJC$_ENTRY_NUMBER

You can specify the following optional input value 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$_ADD_FILE request associates an additional file with the job. Finally, you make an SJC$_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 value item codes:

SJC$_FILE_IDENTIFICATION
SJC$_FILE_SPECIFICATION

You can specify the following input value 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 value item code:

SJC$_ENTRY_NUMBER

You can specify the following input value 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_DEFAULT_RETAIN	—
SJC$_JOB_ERROR_RETAIN	—
SJC$_JOB_NAME	—
SJC$_JOB_RETAIN	—
SJC$_JOB_RETAIN_TIME	—
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 value item code:

SJC$_QUEUE

You can specify the following input value 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 value item codes:

SJC$_DESTINATION_QUEUE
SJC$_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 value 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 can specify the following output value 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 request completes successfully.

You must specify the following input value item code:

SJC$_QUEUE

You can specify the following input value 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$_FILE_TRAILER_ONE	—
SJC$_FIRST_PAGE	SJC$_NO_FIRST_PAGE
SJC$_FORM_NAME	—
SJC$_FORM_NUMBER	—
SJC$_HOLD	SJC$_NO_HOLD
SJC$_JOB_COPIES	—
SJC$_JOB_DEFAULT_RETAIN	—

SJC$_JOB_ERROR_RETAIN	—
SJC$_JOB_NAME	—
SJC$_JOB_RETAIN	—
SJC$_JOB_RETAIN_TIME	—
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$_NO_OPERATOR_REQUEST
SJC$_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 can specify the following output value 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 value item code:

SJC$_QUEUE

You can specify the following input value or Boolean item codes:

SJC$_AUTOSTART_ON	—
SJC$_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$_QUEUE_MANAGER_NAME	—
SJC$_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 value item code:

SJC$_QUEUE

SJC$_DEFINE_CHARACTERISTIC
This request defines a characteristic name and number and inserts this definition into 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 can possess additional characteristics and the job will still execute.

You must specify the following input value 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 9999. 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 value item codes:

SJC$_FORM_NAME
SJC$_FORM_NUMBER

You can specify the following input value 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 value 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 value item code:

SJC$_FORM_NAME

SJC$_DELETE_JOB
This request deletes a job from the system job queue file. If the job is currently executing, it is aborted. If you specify the SJC$_QUEUE item code, the $SNDJBC service verifies that the selected job entry exists on the specified queue before deleting the job.

You must specify the following input value item code:

SJC$_ENTRY_NUMBER

You can specify the following input value 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 value item code:

SJC$_QUEUE

SJC$_DELETE_QUEUE_MANAGER
This request removes all references to the specified queue manager from the shared master file. It also deletes the queue and journal files associated with the queue manager. A queue manager must be stopped to be deleted.

You must specify the following input value item code:

SJC$_QUEUE_MANAGER_NAME

SJC$_DISABLE_AUTOSTART
This request disables autostart on a node. By default, SJC$_DISABLE_AUTOSTART affects the requesting node. To disable autostart on a node other than the node from which the $SNDJBC request is sent, use the SJC$_SCSNODE_NAME item code to specify the affected node.

Disabling autostart on a node forces the appropriate queue manager to perform these tasks:

- Prevent autostart queues from failing over to the node.
- Mark all of that queue manager's autostart queues on the node as "stop pending" in preparation for a planned shutdown, allowing jobs currently executing on the queues to complete.
- Force all autostart queues with failover lists to fail over to the next available node in the queue manager's failover list on which autostart is enabled. Each queue fails over when all jobs currently executing on any of that queue manager's queues on the node have completed.

You can specify the following input value item codes:

SJC$_QUEUE_MANAGER_NAME
SJC$_SCSNODE_NAME

For more information, see the *OpenVMS System Manager's Manual.*

SJC$_ENABLE_AUTOSTART
This request notifies the appropriate queue manager process that a node has progressed sufficiently in its startup procedure that batch and print jobs should execute. By default, SJC$_ENABLE_AUTOSTART affects the requesting node. To enable autostart on a node other than the node from which the $SNDJBC request is sent, use the SJC$_SCSNODE_NAME item code to specify the affected node. Once autostart is enabled, the queue manager starts all autostart-active queues on the appropriate node.

When a node reboots, autostart is disabled until the SJC$_ENABLE_AUTOSTART request is entered.

You can specify the following input value item codes:

SJC$_QUEUE_MANAGER_NAME
SJC$_SCSNODE_NAME

For more information, see the *OpenVMS System Manager's Manual.*

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 value item code:

SJC$_QUEUE

You must specify one of the following input value item codes:

SJC$_FILE_IDENTIFICATION

SJC$_FILE_SPECIFICATION

You can specify the following input value 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$_JOB_DEFAULT_RETAIN	—
SJC$_JOB_ERROR_RETAIN	—
SJC$_JOB_NAME	—
SJC$_JOB_RETAIN	—
SJC$_JOB_RETAIN_TIME	—
SJC$_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 can specify the following output value 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 value item codes:

SJC$_DESTINATION_QUEUE
SJC$_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 value 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 value 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 can specify the following input value 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 value item code:

SJC$_QUEUE

You can specify the following input value or Boolean item codes:

SJC$_ALIGNMENT_MASK	—
SJC$_ALIGNMENT_PAGES	—
SJC$_AUTOSTART_ON	—
SJC$_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 clusterwide queue manager for the batch and print queuing system. It also opens the queue database.

The SJC$_START_QUEUE_MANAGER request has the following five uses:

- To create a queue database and initially start the queue manager, issue a SJC$_START_QUEUE_MANAGER request with the SJC$_NEW_VERSION item code. See the description of the SJC$_NEW_VERSION item code for more information. Once the queue manager has been started, it will remain running unless it is explicitly stopped with an SJC$_STOP_QUEUE_MANAGER request.
- If an SJC$_STOP_QUEUE_MANAGER request has been specified, issue a SJC$_START_QUEUE_MANAGER request to restart the queue manager.
- In a VMScluster environment, issue an SJC$_START_QUEUE_MANAGER request with the SJC$_QUEUE_MANAGER_NODES item code to modify the list of preferred nodes on which the queue manager can run. See the description of the SJC_QUEUE_MANAGER_NODES item code for more information.
- In a cluster, issue an SJC$_START_QUEUE_MANAGER request to ensure that the queue manager process is executing on the most preferred, available node. If the queue manager is not running on the most preferred, available node, the queue manager will be moved to that node without interruption of service. If you are using the default node list (*), the queue manager will not move. For more information, see the description of the SJC$_QUEUE_MANAGER_NODES item code.
- To create additional queue managers, issue an SJC$_START_QUEUE_MANAGER request with the SJC$_ADD_QUEUE_MANAGER and SJC$_QUEUE_MANAGER_NAME item codes. Note that the queue manager name must be different from the name of any queue manager currently defined. For more information about creating multiple queue managers, see the *OpenVMS System Manager's Manual*.

You can specify the following input value or Boolean item codes:

SJC$_ADD_QUEUE_MANAGER
SJC$_NEW_VERSION

SJC$_QUEUE_DIRECTORY
SJC$_QUEUE_MANAGER_NAME
SJC$_QUEUE_MANAGER_NODES

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 can specify the following input value item code:

SJC$_ACCOUNTING_TYPES

SJC$_STOP_ALL_QUEUES_ON_NODE
This request stops all queues on a specific node. By default, all queues on the requesting node are stopped. To stop all queues on a node other than the node from which the $SNDJBC request is sent, use the SJC$_SCSNODE_NAME item code to specify the affected node.

Besides stopping all queues on a specific node, this request aborts each job that is currently executing. Aborted jobs that are restartable are requeued. Queues for which an autostart list has been specified fail over to the first available node in the list for which autostart is enabled.

You can specify the following input value item codes:

SJC$_QUEUE_MANAGER_NAME
SJC$_SCSNODE_NAME

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 value item code:

SJC$_QUEUE

SJC$_STOP_QUEUE_MANAGER
This request shuts down the appropriate queue manager. It disables autostart on all nodes; stops all queues; aborts each job that is currently executing, requeuing those jobs that are restartable; and closes the files of the queue database.

You can specify the following input value item code:

SJC$_QUEUE_MANAGER_NAME

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 one of the following input value item codes:

SJC$_ENTRY_NUMBER
SJC$_QUEUE

If SJC$_QUEUE queue is specified, then you must also specify one of the following:

SJC$_ENTRY_NUMBER

SJC$_JOB_NAME

SJC$_WRITE_ACCOUNTING
This request writes an accounting record.

You must specify the following input value item code:

SJC$_ACCOUNTING_MESSAGE

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 1 to 8 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 0.

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

Accounting-Record Type	Type of Image or 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$_ADD_QUEUE_MANAGER
The SJC$_ADD_QUEUE_MANAGER item code is a Boolean item code. It specifies that a new queue manager process should be defined in the master file. The operating system allows no more than five queue managers in a cluster.

(Valid for SJC$_START_QUEUE_MANAGER function code)

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 time value specified must be in the future, or it will be set to the current 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$_AUTOSTART_ON
The SJC$_AUTOSTART_ON item code is an input value item code. For a batch queue, it uses as its value a comma-separated list of the nodes on which the specified queue can be located. Each node name must be followed by a double colon (::).

For an output queue, it uses as its value a comma-separated list of the names of the nodes and devices to which the specified queue's output can be sent. Each node name must be followed by a double colon, and each device name may be followed by the optional colon [:].

By specifying this item code, you designate a queue as an autostart queue. If you specify more than one node name (within a VMScluster environment), the queue can automatically fail over if the node on which the queue is running is removed from the cluster.

This item code cannot be used with the SJC$_SCSNODE_NAME and SJC$_DEVICE_NAME item codes.

For more information, see the *OpenVMS System Manager's Manual.*

(Valid for SJC$_CREATE_QUEUE, SJC$_START_QUEUE function codes)

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.

(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 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$_CHARACTERISTIC_NAME
SJC$_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.

(The following function codes are valid for SJC_CHARACTERISTIC_NAME item code:

SJC$_ALTER_JOB
SJC$_ALTER_QUEUE
SJC$_CREATE_JOB
SJC$_CREATE_QUEUE
SJC$_DEFINE_CHARACTERISTIC
SJC$_DELETE_CHARACTERISTIC
SJC$_ENTER_FILE
SJC$_START_QUEUE)

(The following function codes are valid for SJC$_CHARACTERISTIC_NUMBER item code:

SJC$_ALTER_JOB
SJC$_ALTER_QUEUE
SJC$_CREATE_JOB
SJC$_CREATE_QUEUE
SJC$_DEFINE_CHARACTERISTIC
SJC$_ENTER_FILE
SJC$_START_QUEUE)

SJC$_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.

(Valid for SJC$_BATCH_CHECKPOINT function code)

SJC$_NO_CHECKPOINT_DATA
The SJC$_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$_ALTER_JOB 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 OpenVMS 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 SYS2–5 shows the action taken when you specify a value for SJC$_CPU_DEFAULT.

Table SYS2–5 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 SYS2–5.

(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 VMScluster environment, the SJC$_SCSNODE_NAME item code is used to specify the name of the node on which the device is located.

This item code cannot be used with the SJC$_AUTOSTART_ON item code.

(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$_FILE_BURST
SJC$_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.

(The following function codes are valid for SJC$_FILE_BURST item code:

SJC$_ADD_FILE
SJC$_ALTER_JOB
SJC$_ALTER_QUEUE
SJC$_CREATE_JOB
SJC$_CREATE_QUEUE
SJC$_ENTER_FILE
SJC$_START_QUEUE)

(The following function codes are valid for SJC$_FILE_BURST_ONE item code:

SJC$_ALTER_QUEUE
SJC$_CREATE_JOB
SJC_CREATE_QUEUE
SJC_START_QUEUE)

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.

(The following function codes are valid for SJC$_FILE_FLAG item code:

SJC$_ADD_FILE
SJC$_ALTER_JOB
SJC$_ALTER_QUEUE
SJC$_CREATE_JOB
SJC$_CREATE_QUEUE
SJC$_ENTER_FILE

SJC$_START_QUEUE)

(The following function codes are valid for SJC$_FLAG_ONE item code:

SJC$_ALTER_QUEUE
SJC$_CREATE_JOB
SJC$_CREATE_QUEUE
SJC$_START_QUEUE)

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 OpenVMS 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.

(The following function codes are valid for SJC$_FILE_TRAILER item code:

SJC$_ADD_FILE
SJC$_ALTER_JOB
SJC$_ALTER_QUEUE
SJC$_CREATE_JOB
SJC$_CREATE_QUEUE
SJC$_ENTER_FILE
SJC$_START_QUEUE)

(The following function codes are valid for SJC$_FILE_TRAILER_ONE item code:

SJC$_ALTER_QUEUE
SJC$_CREATE_JOB
SJC$_CREATE_QUEUE
SJC$_START_QUEUE)

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, respectively, a specified form name and number 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 *OpenVMS System Manager's Manual*.

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.

(The following function codes are valid for SJC$_FORM_NAME item code:

SJC$_ALTER_JOB
SJC$_ALTER_QUEUE
SJC$_CREATE_JOB
SJC$_CREATE_QUEUE
SJC$_DEFINE_FORM

SJC$_DELETE_FORM
SJC$_ENTER_FILE
SJC$_START_QUEUE)

(The following function codes are valid for SJC$_FORM_NUMBER item code:

SJC$_ALTER_JOB
SJC$_ALTER_QUEUE
SJC$_CREATE_JOB
SJC$_CREATE_QUEUE
SJC$_DEFINE_FORM
SJC$_ENTER_FILE
SJC$_START_QUEUE)

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.

(Valid for SJC$_DEFINE_FORM function code)

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$_JOB_DEFAULT_RETAIN
The SJC$_JOB_DEFAULT_RETAIN item code is a Boolean item code. It specifies that you want the job to be held in the queue as specified by the queue's retention policy.

For more information about user-specified job retention, see the /RETAIN qualifier for the PRINT or SUBMIT command in the *OpenVMS DCL Dictionary*.

(Valid for SJC$_ALTER_JOB, SJC$_CREATE_JOB, SJC$_ENTER_FILE function codes)

SJC$_JOB_ERROR_RETAIN
The SJC$_JOB_ERROR_RETAIN item code is a Boolean item code. It specifies that you want the job to be retained in the queue if the job completes unsuccessfully. However, the job might be held in the queue even if it completes successfully if the queue is set to retain all jobs because the QUI$V_QUEUE_RETAIN_ALL bit is set in the QUI$_QUEUE_FLAGS item code.

For more information about user-specified job retention, see the /RETAIN qualifier for the PRINT or SUBMIT command in the *OpenVMS DCL Dictionary*.

(Valid for SJC$_ALTER_JOB, SJC$_CREATE_JOB, SCJ$_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$_JOB_RETAIN
The SJC$_JOB_RETAIN item code is a Boolean item code. It specifies that you want the job to be retained in the queue after it has executed, regardless of the job's completion status.

For more information about user-specified job retention, see the /RETAIN qualifier for the PRINT or SUBMIT command in the *OpenVMS DCL Dictionary*.

(Valid for SJC$_ALTER_JOB, SJC$_CREATE_JOB, SJC$_ENTER_FILE function codes)

SJC$_JOB_RETAIN_TIME
The SJC$_JOB_RETAIN_TIME item code is an input value item code. It specifies a quadword time value representing the length of time you want the job to be retained in the queue.

If a delta time is provided, the delta begins when the job completes. However, depending on the queue's job retention policy, the job may be retained indefinitely.

For more information about user-specified job retention, see the /RETAIN qualifier for the PRINT or SUBMIT command in the *OpenVMS DCL Dictionary*.

(Valid for SJC$_ALTER_JOB, SJC$_CREATE_JOB, SJC$_ENTER_FILE 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 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 OpenVMS 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 can 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.

If your system uses multiple queue managers to run batch queues on a separate queue manager from output queues, certain checks that would otherwise be performed for the SJC$_LOG_QUEUE item code of the $SNDJBC system service are not performed.

When batch and print queues are managed by the same queue manager, the queue manager checks to ensure that the queue specified with the SJC$_LOG_QUEUE is an output queue and that the user has access to the output queue. These checks are not made if the batch queue specified by the $SNDJBC service and the output queue specified by the SJC$_LOG_QUEUE item code are managed by different queue managers. If you explicitly specify an output queue for the log file when submitting a batch job, be sure the queue you specify with the SJC$_LOG_QUEUE is an output queue and not a batch queue. Also, be sure that you have access to the printer queue.

(Valid for SJC$_ALTER_JOB, SJC$_CREATE_JOB, SJC$_ENTER_FILE function codes)

SJC$_LOG_SPECIFICATION
SJC$_NO_LOG_SPECIFICATION
The SJC$_LOG_SPECIFICATION item code is an input value item code. It is meaningful only for batch jobs. It specifies the file specification of the log file produced for a batch job. The buffer must contain this OpenVMS RMS file specification. Omitted fields in the file specification are supplied from the default file specification SYS$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.

When used with the SJC$_START_QUEUE_MANAGER function code, it specifies that a new (empty) version of the queue database is to be created, whether or not the database files already exist. This item code is required when initially creating and starting the queuing system, but it should be used with caution thereafter.

Caution

If you specify this item code and a queue database already exists, the new master and queue files of the queue database supersede existing version of those files. However, the journal files of the queue database are deleted. Thus, jobs and other information are lost.

When used with the SJC$_START_ACCOUNTING function code, the SJC$_NEW_VERSION item code specifies that a new version of the system accounting file is to be created, whether or not the file already exists.

(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 paused 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 for batch and 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 OpenVMS 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 you specify a value for SJC$_PRIORITY that is greater than the SYSGEN parameter MAXQUEPRI and you do not have either ALTPRI or OPER privilege, the system uses the greater of the following two values: DEFQUEPRI or MAXQUEPRI. If you have either ALTPRI or OPER privilege, the system uses any value you specify for SJC$_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 OpenVMS 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.

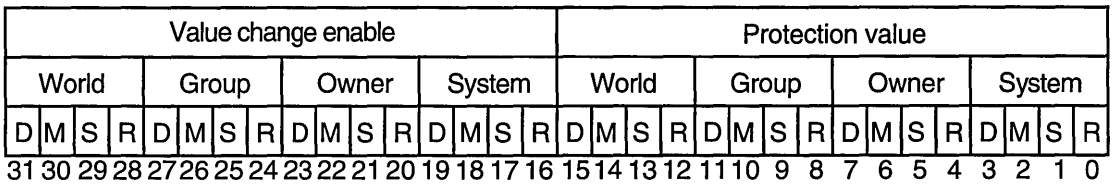
When specified for a generic output queue, SJC$_PROCESSOR specifies that the generic queue can place jobs only in server queues that are executing the symbiont image SYS$SYSTEM:name.EXE, where *name* is the RMS file name contained in the buffer.

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$_PROTECTION
The SJC$_PROTECTION item code 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–GE

Bits 0 through 15 specify the protection value: the four types of access (read, submit, manage, delete) to be granted to the four categories of user (System, Owner, Group, World). Set bits deny access and clear bits allow access.

Bits 16 through 31 specify the protection enable mask: they identify which part of the protection value (bits 0 through 15) is to be applied to queue protection. If all bits are set in the enable mask, it means that all of the protection values are to be applied. A value other than –1 in the protection enable mask means that only those bits set will affect the corresponding bits in the protection value. When a bit in the protection enable mask is clear, the corresponding bit in the existing queue protection value is unchanged.

By default, the queue protection is (S:M,O:D,G:R,W:S).

Note that on VAX systems you can assign ACLs to queues using the $SET_SECURITY system service.

(Valid for SJC$_ALTER_QUEUE, SJC$_CREATE_QUEUE, SJC$_START_QUEUE function codes)

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.

(The following function codes are valid for SJC$_QUEUE item code:

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)

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$_QUEUE_DIRECTORY
The SJC$_QUEUE_DIRECTORY item code is an input value item code. SJC$_QUEUE_DIRECTORY specifies the directory location that contains the system queue and journal files for the queue manager. The queue file has a file type of QMAN$QUEUES and contains queue definitions. The journal file has a file type of .QMAN$JOURNAL and contains job and other information allowing the queue manager to return to its last known state should a system be stopped unexpectedly. These files must reside together in the same directory.

The default location of the queue and journal files is SYS$COMMON:[SYSEXE]. The optional use of SJC$_QUEUE_DIRECTORY is for specifying an alternate location for the queue and journal files. The specification must include at least the device and directory name; wildcard characters are not allowed in the directory specification. The directory specified must be available to all nodes that can run the queue manager. If the directory specification is a concealed logical name, it must be defined identically on all nodes in the cluster.

The location of the queue and journal files is stored in the master file of the queue database. You do not have to respecify the directory location with subsequent use of SJC$_START_QUEUE_MANAGER.

For more information, see the *OpenVMS System Manager's Manual*.

(Valid for SJC$_START_QUEUE_MANAGER function code)

SJC$_QUEUE_MANAGER_NAME
The SJC$_QUEUE_MANAGER_NAME item code is an input value item code. It uniquely identifies the queue manager process that manages some segment of the queues and jobs in the system. If it is not present, a default name of SYS$QUEUE_MANAGER is used.

The maximum length of the final character string is 31 characters. As with queue names, this may be a logical and will be resolved by the system. Once resolved, the name provided will serve as the file name for the queue and journal files, the process name, and the user name for the active process. Only the first 15 and 12 characters of the name are used for the process and user names, respectively.

(Valid for SJC$_CREATE_QUEUE, SJC$_DELETE_QUEUE_MANAGER, SJC$_DISABLE_AUTOSTART, SJC$_ENABLE_AUTOSTART, SJC$_START_QUEUE_MANAGER, SJC$_STOP_ALL_QUEUES_ON_NODE, SJC$_STOP_QUEUE_MANAGER function codes)

SJC$_QUEUE_MANAGER_NODES
The SJC$_QUEUE_MANAGER_NODES item code is an input value item code. In a VMScluster, SJC$_QUEUE_MANAGER_NODES specifies a list of nodes that can run the queue manager. It also gives the explicit order of failover if the node running the queue manager exits the cluster. The specified node list is stored in the queue database.

The default value for the node list is an asterisk (*); it specifies that all nodes in the cluster are eligible to run the queue manager. The asterisk may also be specified as an element of the list. For example, a list may be specified as nodes A, B, C, *. If the node on which the queue manager is running leaves the cluster, the queue manager automatically fails over to any available node in the cluster; that is, if nodes A, B, and C are unavailable, then the queue manager may run on any other node. When establishing the node list, there is no validation of the individual nodes. If, for example, a node name is misspelled, there is no error status returned.

Anytime the SJC$_START_QUEUE_MANAGER function code is used, the job controller checks the queue database to see if the node list is other than the default (*). If the node list is other than the default and the queue manager is running on a node other than the first available node of those specified, then the queue manager process is moved from its current node and restarted on the first available preferred node. When a current call includes the SJC$_QUEUE_MANAGER_NODES item code, the job controller also updates the node list stored in the database. Despite this transition, queues on the running nodes are not stopped, and all requests to the queuing system complete as expected.

Note that because the specified node list is saved in the database, it is used every time the SJC$_START_QUEUE_MANAGER function code is used, unless the node list has been changed by a more recent call to $SNDJBC with the SJC_$QUEUE_MANAGER_NODES item code.

For more information, see the *OpenVMS System Manager's Manual.*

(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 OpenVMS 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 specifies the name of the node for which the command is to execute. The buffer must specify a 1- to 6-character string that matches the value of the SYSGEN parameter SCSNODE in effect on the target node.

When used with the function codes of SJC$_STOP_ALL_QUEUES_ON_NODE, SJC$_DISABLE_AUTOSTART, and SJC$_ENABLE_AUTOSTART, this item code requests a function on a node other than the node from which the $SNDJBC request is sent.

SJC$_SCSNODE_NAME is meaningful only for execution queues in a cluster environment. By default, the queue executes on the node from which the queue is first started. For an output execution queue, you use the SJC$_DEVICE_NAME item code to specify the name of the device managed by the queue.

(Valid for SJC$_CREATE_QUEUE, SJC$_DISABLE_AUTOSTART, SJC$_ENABLE_AUTOSTART, SJC$_START_QUEUE, SJC$_STOP_ALL_QUEUES_ON_NODE 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$_NO_TERMINAL are supported for compatibility with VAX/VMS Version 4.n, but may not be supported in the future. For SJC$_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, in pages (on VAX systems) or pagelets (on Alpha systems), 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 SYS2–6 shows the action taken when you specify a value for SJC$_WSDEFAULT.

Table SYS2–6 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, in pages (on VAX systems) or pagelets (on Alpha systems), 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 SYS2–6.

(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, in pages (on VAX systems) or pagelets (on Alpha systems), 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 SYS2–6.

(Valid for SJC$_ALTER_JOB, SJC$_ALTER_QUEUE, SJC$_CREATE_JOB, SJC$_CREATE_QUEUE, SJC$_ENTER_FILE, SJC$_START_QUEUE function codes)

Description

The Send to Job Controller service creates, stops, and manages queues and the batch and print jobs in those queues. The $SNDJBC and $GETQUI (Get Queue Information) services together provide the user interface to the queue manager and job controller processes. See the description of the $GETQUI service for a discussion of queues and jobs initiated from those queues.

$SNDJBC 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.

Types of Queues The VMS batch and print queuing system supports several types of queues, which aid in the processing of batch and print jobs. The different types of queues can be divided into three major categories according to the way the system processes the jobs assigned to the queue. The three types of queues 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 queues can accept for processing. Some types of execution and generic queues 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:

- **Execution queue.** An execution queue schedules jobs for processing. In a VMScluster environment, jobs are processed on the node that manages the execution queue. There are two types of execution queues:
  - **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.
  - **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, such as a printer or terminal, that it controls.

    The standard print symbiont image provided by the operating system 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 batch and print queuing system. 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 print symbiont image, PRTSMB, is associated with the queue.

    There are three types of output execution queue:

    a. **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.

    b. **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.

    c. **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:

a. When an output execution queue associated with the standard 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.

b. 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.

- **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 cluster environment, a generic queue can direct jobs to execution queues that are located on other nodes in the cluster.

  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:

  – **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.

  – **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.

- **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.

There are two aspects to UIC-based queue protection:

- When you create a queue, you assign it a UIC by using the SJC$_OWNER_UIC item code. If you do not specify this item code, the queue is given the default UIC [1,4].
- You can assign a queue a protection mask by specifying the SJC$_PROTECTION item code. This protection mask specifies read, submit, manage, 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 in the Privileges and Restrictions section.

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 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 read and delete protection specified for the queue in which the job is entered and the owner UIC of the job. In general, read access to a job allows you to determine that the job exists; delete access to a job allows you to affect the job.

Operations that apply to queues are checked against the submit and manage protection specified for the queue and the owner UIC of the queue. In general, submit access to a queue allows you to submit jobs to the queue; manage 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 manage access to all queues.

Privileges and Restrictions To specify the following function codes, the caller must have both OPER and SYSNAM privilege:

SJC$_DELETE_QUEUE_MANAGER
SJC$_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 can have OPER privilege or manage access:

SJC$_DELETE_QUEUE

To specify the following function code, the caller must have OPER privilege, execute access to the queue containing the specified job, or read access to the specified job:

SJC$_SYNCHRONIZE_JOB

To specify the following function codes, the caller must have OPER privilege, manage access to the specified queue, or submit 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 manage access to the specified queue or queues:

SJC$_ALTER_QUEUE
SJC$_ASSIGN_QUEUE
SJC$_DEASSIGN_QUEUE
SJC$_DISABLE_AUTOSTART
SJC$_ENABLE_AUTOSTART
SJC$_MERGE_QUEUE
SJC$_PAUSE_QUEUE
SJC$_RESET_QUEUE
SJC$_START_QUEUE
SJC$_STOP_ALL_QUEUES_ON_NODE
SJC$_STOP_QUEUE

To specify the following function codes, the caller must have OPER privilege, manage access to the queue containing the specified job, or delete 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 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$_OWNER_UIC
SJC$_PROTECTION

To specify the following item codes, the caller must have CMKRNL privilege:

SJC$_ACCOUNT_NAME
SJC$_UIC
SJC$_USERNAME

Required Quota

To specify the **astadr** argument, the process must have sufficient ASTLM quota.

Related Services

$ALLOC, $ASSIGN, $BRKTHRU, $BRKTHRUW, $CANCEL, $CREMBX, $DALLOC, $DASSGN, $DELMBX, $DEVICE_SCAN, $DISMOU, $GETDVI, $GETDVIW, $GETMSG, $GETQUI, $GETQUIW, $INIT_VOL, $MOUNT, $PUTMSG, $QIO, $QIOW, $SNDERR, $SNDJBCW, $SNDOPR, $TRNLNM

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$_BADPARAM	The function code is invalid; the item descriptor contains an invalid buffer length value; a buffer descriptor has an invalid length; or the reserved parameter has a nonzero value.
SS$_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$_INSFMEM	Insufficient space exists for completing the request.
SS$_IVLOGNAM	Queue form or characteristic name is not a valid logical name.
SS$_MBFULL	The job controller mailbox is full.

SS$_MBTOOSML	The mailbox message is too large for the job controller mailbox.
SS$_SHELVED	The job controller attempted to access a shelved file. The service does not automatically unshelve files.
SS$_UNASEFC	The **efn** argument specifies an unassociated event flag cluster.

Condition Values Returned in the I/O Status Block

JBC$_NORMAL	The service completed successfully.
JBC$_AUTONOTSTART	The queue is autostart active, but not started. You have tried to start an autostart queue when none of its available nodes has autostart enabled.
JBC$_BUFTOOSMALL	The request could not be completely satisfied due to limited buffer size. The amount of information retrieved in response to the query exceeds the amount of data the queue manager can return in response to a single request.
JBC$_DELACCESS	The file protection of the specified file, which was entered with the delete option, does not allow delete access to the caller.
JBC$_DUPCHARNAME	The command specified a duplicate characteristic name. Each characteristic must have a unique name.
JBC$_DUPCHARNUM	The command specified a duplicate characteristic number. Each characteristic must have a unique number.
JBC$_DUPFORM	The specified form number is invalid because it is already defined; each form must have a unique form number.
JBC$_DUPFORMNAME	The command specified a duplicate form name. Each form must have a unique name.
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$_INTERNALERROR	An internal error caused loss of process status. A system error prevented the queue manager from obtaining the completion status of a process.
JBC$_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$_INVQUENAM	The queue name is not syntactically valid.
JBC$_ITMREMOVED	The meaningless items were removed from the request. One or more item codes not meaningful to this command were specified. The command is processed and the meaningless items are ignored.
JBC$_JOBNOTEXEC	The specified job is not executing.
JBC$_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$_NOAUTOSTART	The node does not have the autostart feature enabled.
JBC$_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$_NORESTART	The specified job cannot be requeued because it was not defined as restartable.
JBC$_NOSUCHCHAR	The specified characteristic does not exist.
JBC$_NOSUCHENT	There is no job with the specified entry number.
JBC$_NOSUCHFORM	The specified form does not exist.

JBC$_NOSUCHJOB	The specified job does not exist.
JBC$_NOSUCHMGR	The specified queue manager does not exist.
JBC$_NOSUCHNODE	The specified node does not exist.
JBC$_NOSUCHQUE	The specified queue does not exist.
JBC$_NOTALLREQUE	Not all jobs in the source queue could be requeued to the target queue. Some of the jobs specified were not suitable for execution on the specified target queue.
JBC$_NOTASSIGN	The specified queue cannot be deassigned because it is not assigned.
JBC$_NOTMEANINGFUL	The specified item code is no longer meaningful.
JBC$_NOTSUPPORTED	The specified item code or function code is not supported.
JBC$_PRIOSMALL	The scheduling priority has a smaller value than requested. A user without ALTPRI or OPER privilege specified a value for a job's priority that exceeded the queue's maximum priority for nonprivileged jobs. The job is entered in the queue, but its scheduling priority is lower than the value requested by the user.
JBC$_QMANNOTSTARTED	The queue manager could not be started.
JBC$_QUEDISABLED	The disabled queue cannot be modified, nor can jobs be submitted to it.
JBC$_QUENOTMOD	The modifications were not made because the queue was not stopped.
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$_STARTED	The specified queue cannot be started because it is already running.
JBC$_STKNOTCHANGE	The stock associated with a form cannot be changed.
JBC$_TOOMUCHINFO	The size of the data in request exceeds system constraints. The amount of data specified for a record within the queue manager's database is too large.

When you use the SJC$_SYNCHRONIZE_JOB function code, the return value is the exit status of the specified job.

When you start a symbiont queue with the SJC$_START_QUEUE function code or the SJC$_CREATE_QUEUE function code with the SJC$_CREATE_START item code, any error encountered by the symbiont process will be returned in the IOSB.

Example

```
! 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
```

This 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.

$SNDJBCW
Send to Job Controller and Wait

The Send to Job Controller and Wait 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 can be generic, batch, execution, or output queues. Jobs can 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 $SNDJBCW, refer to the documentation of $SNDJBC.

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

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]

Arguments

msgbuf
OpenVMS 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.

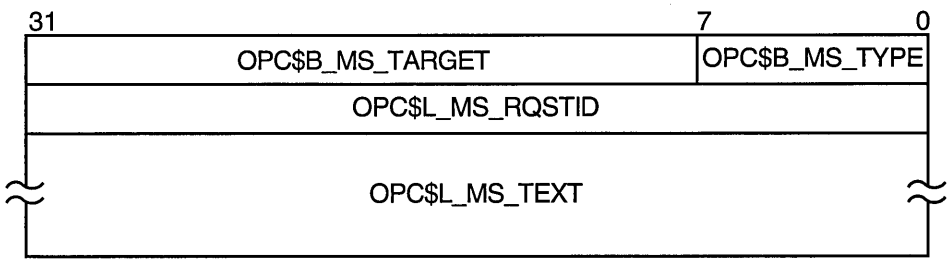
The format and contents of the buffer vary with the requested operation; however, the first byte in any buffer is the request code, which specifies the operation to be performed. The $OPCMSG macro defines the symbolic names for these request codes. The following table shows each operation that $SNDOPR performs and the request code that specifies that operation.

Request Code	Corresponding Operation
OPC$_RQ_CANCEL	Sends a user cancellation request to specified operator terminals. You use this request code to notify one or more operators that a previous request is to be canceled. To specify OPC$_RQ_CANCEL, you must also specify the **chan** argument.
OPC$_RQ_LOGI	Initializes the operator log file.
OPC$_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.

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_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_TERME	Enables an operator terminal. You use this request to enable a specified terminal to receive operator messages.

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 can be specified as the contents of a field.

Message Buffer Format for OPC$_RQ_RQST



ZK-1725-GE

OPC$B_MS_TYPE — This 1-byte field contains the request code OPC$_RQ_RQST.

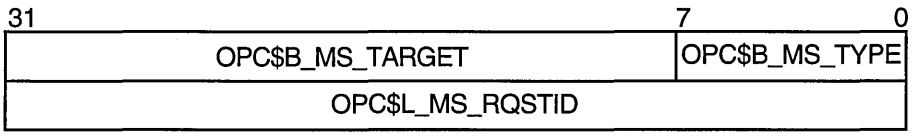
OPC$B_MS_TARGET — This 3-byte field contains a 24-bit bit vector that specifies which operator terminal types are to receive the request. The $OPCDEF macro defines symbolic names for the operator terminal types. You construct the bit vector by specifying the desired symbolic names in a logical OR operation. Following is the symbolic name of each operator terminal type:

	OPC$M_NM_CARDS	Card device operator
	OPC$M_NM_CENTRL	Central operator
	OPC$M_NM_CLUSTER	VMScluster operator
	OPC$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 through 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. $SNDOPR uses the buffer size of the device to which the message is being sent.

Message Buffer Format for OPC$_RQ_CANCEL



ZK-1726-GE

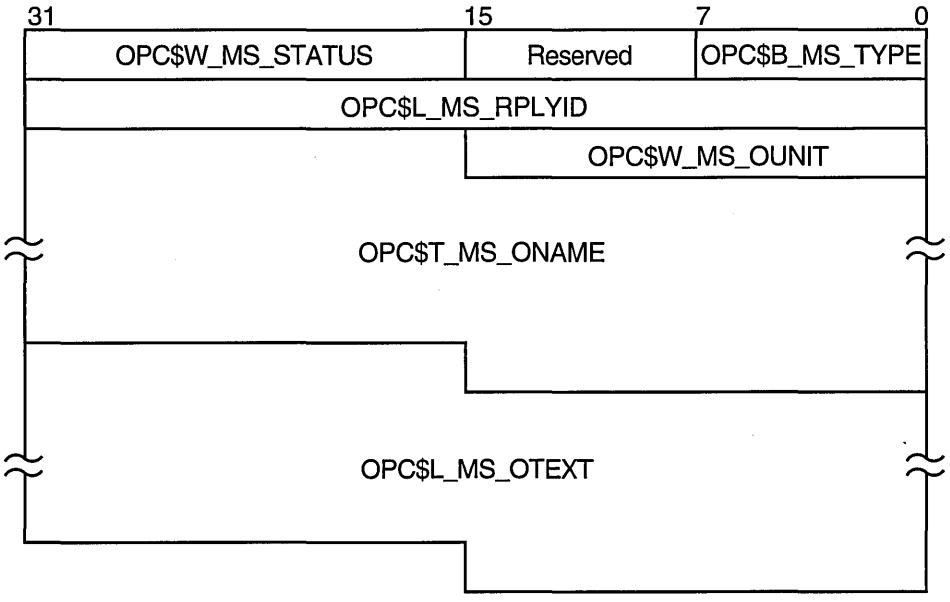
OPC$B_MS_TYPE — This 1-byte field contains the request code OPC$_RQ_CANCEL.

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	VMScluster 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 through OPC$M_NM_OPER12	System-manager-defined operator functions

OPC$L_MS_RQSTID — This longword field contains a user-supplied longword message code.

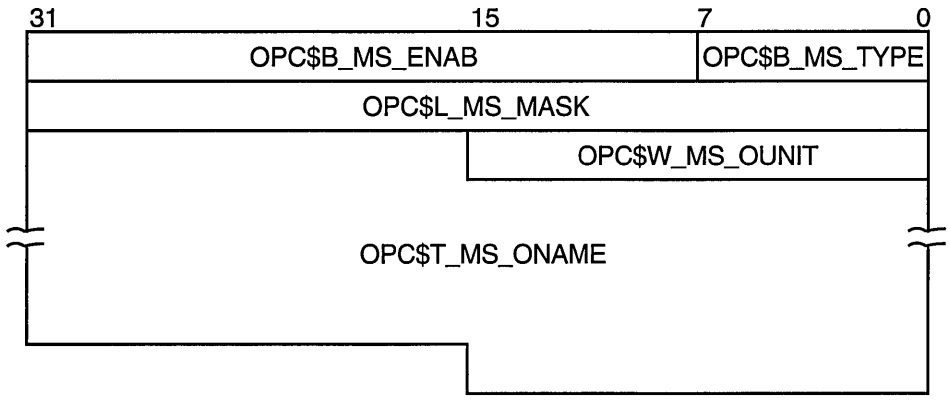
Message Buffer Format for OPC$_RQ_REPLY



ZK-1727-GE

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$W_MS_OUNIT) to learn how to obtain the device name.
OPC$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



ZK–1728–GE

OPC$B_MS_TYPE — This 1-byte field contains the request code OPC$_RQ_TERME.

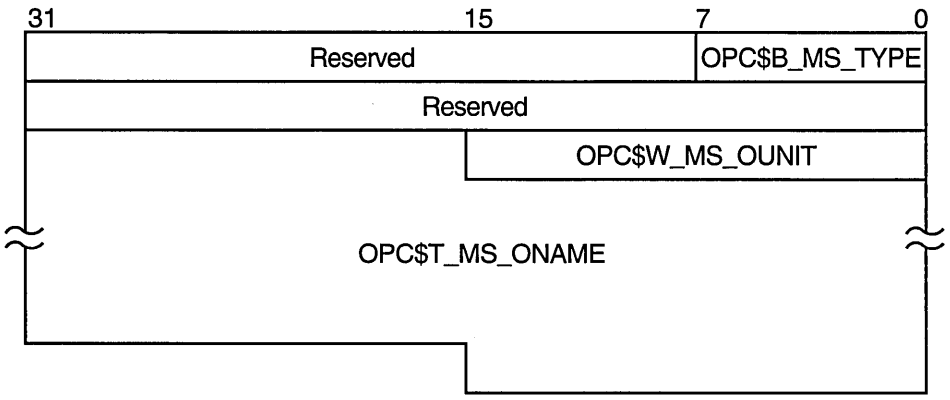
OPC$B_MS_ENAB — This 3-byte field contains a user-supplied value. The value 0 indicates that the specified terminal is to be disabled for the specified operator classes. Any nonzero value indicates that the specified terminal is to be enabled for the specified operator classes.

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	VMScluster 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 through 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$W_MS_OUNIT) to learn how to obtain the device name.

Message Buffer Format for OPC$_RQ_STATUS



ZK-1729-GE

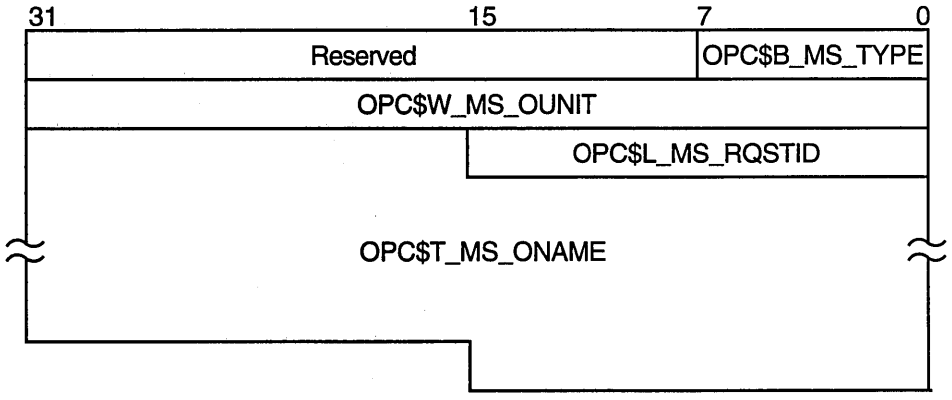
OPC$B_MS_TYPE	This 1-byte field contains the request code OPC$_RQ_STATUS.
Reserved	This 3-byte field is reserved for future use.
Reserved	This 4-byte field is reserved for future use.

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$T_MS_ONAME — This variable-length field contains a counted ASCII string specifying the device name of the operator terminal whose status is requested. The maximum total length of the string is 14 bytes. See the preceding field description (OPC$W_MS_OUNIT) to learn how to obtain the device name.

Message Buffer Format for OPC$_RQ_LOGI



ZK-1730-GE

OPC$B_MS_TYPE — This 1-byte field contains the request code OPC$_RQ_LOGI.

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	VMScluster 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 through OPC$M_NM_OPER12	System-manager-defined operator functions

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 with all classes enabled (OPC$B_MS_TARGET is ignored).

The value 1 specifies that the current operator log file is to be closed but no new log file is to be opened.

The value 2 specifies that the classes in OPC$B_MS_TARGET are added to the current operator log file classes. A log file is opened if necessary.

The value 3 specifies that the operator classes in OPCB_MS_TARGET are to be removed from the operator log file classes. If all classes are removed, the log file is closed.

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$W_MS_OUNIT) to learn how to obtain the device name.

chan

OpenVMS 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 the value 0 (the default), no reply is sent.

If a reply from the operator is desired, you must specify the **chan** argument.

Description

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

This system service requires system dynamic memory; it cannot be called from kernel mode.

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 $SNDOPR service.
3. Check the condition value returned in R0 to make sure the request was successfully made.
4. Issue a read request to the mailbox specified, if any.
5. When the read operation completes, check the 2-byte condition value in the OPC$W_MS_STATUS field to make sure 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-1994 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-1994 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-1994 12:57:32.25
Request 1285, from user ROSS on NODE_NAME
Please mount device _NODE$DMA0:
```

Required Access or Privileges

OPER privilege is required for the following functions:

- Enabling a terminal as an operator's terminal
- Replying to or canceling a user's request
- Initializing the operator communication log file

In addition, the operator must have SECURITY privilege to affect security functions.

Required Quota

None

Related Services

$ALLOC, $ASSIGN, $BRKTHRU, $BRKTHRUW, $CANCEL, $CREMBX, $DALLOC, $DASSGN, $DELMBX, $DEVICE_SCAN, $DISMOU, $GETDVI, $GETDVIW, $GETMSG, $GETQUI, $GETQUIW, $INIT_VOL, $MOUNT, $PUTMSG, $QIO, $QIOW, $SNDERR, $SNDJBC, $SNDJBCW

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$_INSFMEM	The service was called from kernel mode or the system dynamic memory is insufficient for completing the service.
SS$_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$_MBFULL	The mailbox used to support communication is full. Retry at a later time.
OPC$_NOPERATOR	The service completed successfully; the Operator Communications Manager (OPCOM) is not running and the message will not be sent. Note that OPC$_NOPERATOR is a success status and must be tested for explicitly.
SS$_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.
```
#include <ssdef.h>
#include <opcdef.h>
#include <string.h>
#include <descrip.h>
#include <starlet.h>
#include <lib$routines.h>

char    input_buffer[256];      /* Input buffer, if needed */

/* VMS descriptors: */
$DESCRIPTOR(input_desc, input_buffer);
$DESCRIPTOR(prompt_desc, "Request> ");
struct dsc$descriptor req_desc;
```

```
main(int argc, char *argv[])
{
    int status,                 /* Status of system calls */
        length = 0;             /* Length of message text */
    struct OPC request;         /* Request message buffer */

    /* Check for too many arguments on command line */
    if (argc > 2)
        return (SS$_OVRMAXARG);

    /* See if request string present on command line... */
    if (argc > 1)
    {
        /* It is.  Compute length and copy pointer */
        length = strlen(argv[1]);
        input_desc.dsc$a_pointer = argv[1];
    }

    /* If no message present, prompt user for message text */
    while (length == 0)
    {
        status = lib$get_input(&input_desc, &prompt_desc, &length);
        if (status != SS$_NORMAL)
            return (status);
    };

    if (length > 128)           /* Limit message length */
        length = 128;           /*    to 128 characters */

    /* Set up request buffer... */
    request.opc$b_ms_type = OPC$_RQ_RQST;
    request.opc$b_ms_target = OPC$M_NM_CENTRL;
    request.opc$l_ms_rqstid = 0;
    memcpy(&request.opc$l_ms_text, input_desc.dsc$a_pointer, length);

    /* Set up request buffer descriptor and send message */
    req_desc.dsc$w_length = length + 8;
    req_desc.dsc$a_pointer = (char *) &request;
    return (sys$sndopr(&req_desc, 0));
}
```

This 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 DEC Fortran for OpenVMS program enables the current terminal to receive OPER1 operator messages.

$START_ALIGN_FAULT_REPORT (Alpha Only) Start Alignment Fault Reporting

On Alpha systems, initializes user image alignment fault reporting.

Format

SYS$START_ALIGN_FAULT_REPORT report_method ,report_buffer ,buffer_length

Arguments

report_method
OpenVMS usage: longword_signed
type: longword (signed)
access: read
mechanism: by value

Method by which image alignment faults are to be reported. The following table shows valid values for the **report_method** argument.

Value	Meaning
AFR$C_BUFFERED	Alignment fault PCs and fault addresses are saved in a user-supplied buffer.
AFR$C_EXCEPTION	Alignment faults are elevated to user mode exceptions.

report_buffer
OpenVMS usage: address
type: longword (unsigned)
access: read
mechanism: by reference

The 32-bit address of the buffer into which to write the fault data. The **report_buffer** argument is needed only if the value of the **report_method** argument is AFR$C_BUFFERED.

buffer_length
OpenVMS usage: byte count
type: longword (signed)
access: read
mechanism: by value

Length of the buffer specified in the **report_buffer** argument. The buffer must have a minimum size of AFR$K_USER_LENGTH + 32. However, a larger buffer allows for more information to be collected.

Description

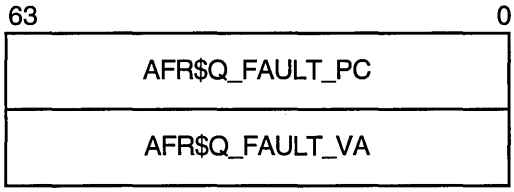
The Start Alignment Fault Reporting service initializes user image alignment fault reporting.

The $START_ALIGN_FAULT_REPORT service allows the user to gather alignment fault data for one image. Reporting is enabled for the life of the image. When the image terminates, the alignment fault reporting is disabled.

User alignment fault data can be written to a buffer or broadcast as an informational exception message.

If the AFR$C_BUFFERED value is given in the **report_method** buffer, alignment fault PCs and fault addresses are saved in a user-supplied buffer.

The following diagram illustrates the format in which user alignment fault data is stored in the buffer.



ZK–4983A–GE

If the AFR$C_EXCEPTION value is given in the **report_method** argument, alignment faults are elevated to user mode exceptions. These exceptions can be trapped in a handler. Otherwise, an informational exception message may be broadcast and the program continues to execute.

Required Access or Privileges

None

Required Quota

None

Related Services

$GET_ALIGN_FAULT_DATA, $GET_SYS_ALIGN_FAULT_DATA, $INIT_SYS_ALIGN_FAULT_REPORT, $PERM_DIS_ALIGN_FAULT_REPORT, $PERM_REPORT_ALIGN_FAULT, $STOP_ALIGN_FAULT_REPORT, $STOP_SYS_ALIGN_FAULT_REPORT

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The buffer specified in the **report_buffer** argument is not accessible.
SS$_AFR_ENABLED	The service has already been called for this image.
SS$_ARG_GTR_32_BITS	The report buffer's virtual address lies in 64-bit virtual address space.
SS$_ALIGN	The buffer specified in the **report_buffer** argument is not quadword aligned.
SS$_BADPARAM	The buffer size is smaller than that defined by the AFR$K_USER_LENGTH + 32 symbol.

Example

```
#include <afrdef>
#include <stdio>
#include <ssdef>

#define USER_BUFFER_ITEMS  10
#define GET_BUFFER_SIZE    USER_BUFFER_ITEMS*AFR$K_USER_LENGTH
#define SAVE_BUFFER_SIZE   128+64

#define fault_pc afr$l_fault_pc_l
#define fault_va afr$l_fault_va_l

static int usr_buff_len;
static char *usr_buff;
static int rep_method;

void
cause_af()
{
  int     addr;
  int     *ptr;
  int    arr[2];

  addr = (int) &arr[0];
  ptr = (int *) ++addr;
  *ptr = 1;    /* cause alignment fault */
}

main()
{
  int            i;
  char           get_buffer[GET_BUFFER_SIZE];
  struct afrdef  *data_item;
  int            offset;
  int            status;
  int            return_size;

  rep_method = AFR$C_BUFFERED;
  usr_buff_len = SAVE_BUFFER_SIZE;
  usr_buff = (char *)malloc (usr_buff_len);
  if(( status = sys$start_align_fault_report(rep_method, usr_buff,
              usr_buff_len))
      != SS$_NORMAL) return(status);

  for (i=0;i<USER_BUFFER_ITEMS;i++)
    cause_af();

  while (((status = sys$get_align_fault_data (get_buffer,
               GET_BUFFER_SIZE,
               &return_size)) > 0) &&
         (return_size > 0)) {
    /* got some data, print it */
    offset = 0;
    while (offset < return_size) {
      data_item = (struct afrdef *)(&get_buffer[offset]);
      printf ("Alignment fault at PC = %8.8X, VA = %8.8X\n",
        data_item->fault_pc, data_item->fault_va);
      offset += AFR$K_USER_LENGTH;
    }
  }

  return (status);
}
```

This example shows how to use the $START_ALIGN_FAULT_REPORT service to initialize user image alignment fault reporting on Alpha systems.

$START_TRANS
Start Transaction

Starts a new transaction.

Format

SYS$START_TRANS [efn] ,[flags] ,iosb [,[astadr] ,[astprm] ,[tid] ,[timout] ,[acmode]]

Arguments

efn

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

Number of the event flag that is set when the service completes. If this argument is omitted, event flag 0 is set.

flags

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

Flags specifying options for the service. The **flags** argument is a longword bit mask in which each bit corresponds to an option flag. The $DDTMDEF macro defines symbolic names for these option flags. The flags currently defined are shown in the following table. All undefined bits must be 0. If this argument is omitted, no flags are set.

Flag	Description
DDTM$M_NONDEFAULT	Set this flag if you do not want the new transaction to be the default transaction of the calling process. If this flag is clear, the new transaction becomes the default transaction of the calling process. An error is returned if this flag is clear and the calling process already has a default transaction.
DDTM$M_PROCESS	Set this flag if you do not want the DECdtm transaction manager to try to abort the transaction if the current image terminates. If this flag is clear, when the current image terminates (normally or abnormally), the DECdtm transaction manager will abort the transaction if it has not already committed. An error is returned if this flag is set and the caller is in user mode.

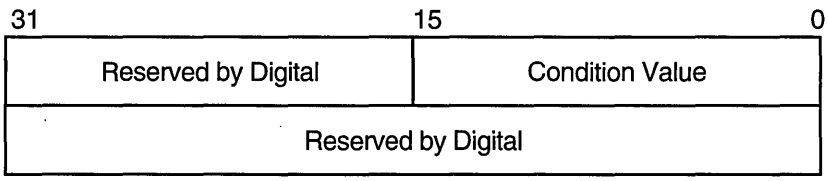
Flag	Description
DDTM$M_SYNC	Set this flag to specify that successful synchronous completion is to be indicated by returning SS$_SYNCH. When SS$_SYNCH is returned, the AST routine is not called, the event flag is not set, and the I/O status block is not filled in.

iosb

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

I/O status block in which the completion status of the service is returned as a condition value. See the Condition Values Returned section.

The following diagram shows the structure of the I/O status block.



ZK–1224A–GE

astadr

OpenVMS usage: ast_procedure
type: procedure value
access: call without stack unwinding
mechanism: by reference

AST routine that is executed when the service completes. The **astadr** argument is the address of this routine. This routine is executed in the access mode of the caller.

astprm

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

AST parameter that is passed to the AST routine specified by the **astadr** argument.

tid

OpenVMS usage: transaction_id
type: octaword (unsigned)
access: write only
mechanism: by reference

Address of an octaword in which the service returns the identifier of the new transaction.

timout

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

Timeout for the new transaction. This is the time at which the DECdtm transaction manager is to abort the transaction if the transaction has not already committed.

The time value is a binary number, in units of 100 nanoseconds.

A positive time value specifies an offset from the system base time. The system base time is 00:00 hours November 17, 1858.

A negative time value specifies an offset from the current time to some time in the future.

The transaction is aborted at the next timer interval if you specify either a zero time value or any time in the past.

If this argument is omitted, the new transaction has no timeout.

acmode

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

The least privileged access mode that the calling process must be in to end the transaction by calling $END_TRANS. Note that the calling process can end the transaction by calling $ABORT_TRANS from any access mode.

The access mode that the calling process must be in to end the transaction by calling $END_TRANS is whichever is the least privileged of the following:

- The access mode of the caller.
- The access mode specified by the **acmode** argument.

If the **acmode** argument is omitted, it defaults to the access mode of the caller.

Description

The Start Transaction service starts a new transaction that is to be coordinated by the DECdtm distributed transaction manager.

$START_TRANS creates a unique transaction identifier for the new transaction. The same identifier can never be created by any other call to $START_TRANS on any node.

Each process can have a default transaction. This is the transaction that is assumed when the process:

- Invokes resource manager operations without specifying a transaction identifier, for resource managers such as RMS Journaling that support default transactions.
- Calls $END_TRANS or $ABORT_TRANS without specifying a **tid** argument.

By default, the new transaction becomes the default transaction of the calling process. If you want to start a new transaction and the calling process already has a default transaction, set the DDTM$M_NONDEFAULT flag.

Required Access or Privileges

None

Required Quotas

ASTLM, BYTLM

Related Services

$ABORT_TRANS, $ABORT_TRANSW, $END_TRANS, $END_TRANSW, $START_TRANSW

Condition Values Returned

SS$_NORMAL	If this was returned in R0, the request was successfully queued. If it was returned in the I/O status block, the service completed successfully.
SS$_SYNCH	The service completed successfully and synchronously (returned only if the DDTM$M_SYNC flag is set).
SS$_ACCVIO	An argument was not accessible by the caller.
SS$_ALCURTID	An attempt was made to start a default transaction (the DDTM$M_NONDEFAULT flag was clear) when the calling process already had a default transaction.
SS$_BADPARAM	Either the DDTM$M_NONDEFAULT flag was set and the **tid** argument was omitted, or the options flags were invalid.
SS$_CURTIDCHANGE	The DDTM$M_NONDEFAULT flag was clear and a call to change the default transaction of the calling process was in progress.
SS$_EXASTLM	The process AST limit (ASTLM) was exceeded.
SS$_EXQUOTA	The job buffered I/O byte limit quota (BYTLM) was exceeded.
SS$_ILLEFC	The event flag number was invalid.
SS$_INSFARGS	Not enough arguments were supplied.
SS$_INSFMEM	There was insufficient system dynamic memory for the operation.
SS$_NOLOG	The local node did not have a transaction log.
SS$_TPDISABLED	The TP_SERVER process was not running on the local node.
SS$_WRONGACMODE	The DDTM$M_PROCESS flag was set and the caller was in user mode.

$START_TRANSW
Start Transaction and Wait

Starts a new transaction.

$START_TRANSW always waits for the request to complete before returning to the caller. Other than this, it is identical to $START_TRANS.

Format

SYS$START_TRANSW [efn] ,[flags] ,iosb [,[astadr] ,[astprm] ,[tid] ,[timout] ,[acmode]]

$STOP_ALIGN_FAULT_REPORT (Alpha Only) Stop Alignment Fault Reporting

On Alpha systems, disables user image alignment fault reporting.

Format

SYS$STOP_ALIGN_FAULT_REPORT

Description

The Stop Alignment Fault Reporting service disables user image alignment fault reporting.

The service returns SS$_AFR_NOT_ENABLED if user image alignment fault reporting is not enabled. Otherwise, it returns success.

Required Access or Privileges

None

Required Quota

None

Related Services

$GET_ALIGN_FAULT_DATA, $GET_SYS_ALIGN_FAULT_DATA, $INIT_SYS_ALIGN_FAULT_REPORT, $PERM_DIS_ALIGN_FAULT_REPORT, $PERM_REPORT_ALIGN_FAULT, $START_ALIGN_FAULT_REPORT, $STOP_SYS_ALIGN_FAULT_REPORT

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_AFR_NOT_ENABLED	The $START_ALIGN_FAULT_REPORT service has not been called.

$STOP_SYS_ALIGN_FAULT_REPORT (Alpha Only) Stop System Alignment Fault Reporting

On Alpha systems, disables systemwide alignment fault reporting.

Format

SYS$STOP_SYS_ALIGN_FAULT_REPORT

Description

The Stop System Alignment Fault Reporting service disables systemwide alignment fault reporting.

The service returns SS$_AFR_NOT_ENABLED if systemwide alignment fault reporting is not enabled. Otherwise, it returns success.

Required Access or Privileges

CMKRNL privilege is required.

Required Quota

None

Related Services

$GET_ALIGN_FAULT_DATA, $GET_SYS_ALIGN_FAULT_DATA, $INIT_SYS_ALIGN_FAULT_REPORT, $PERM_DIS_ALIGN_FAULT_REPORT, $PERM_REPORT_ALIGN_FAULT, $START_ALIGN_FAULT_REPORT

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_NOPRIV	The caller lacks sufficient privilege.
SS$_AFR_NOT_ENABLED	The $START_ALIGN_FAULT_REPORT service has not been called.

$SUBSYSTEM
Subsystem

Saves or restores the process image rights for the current protected subsystem.

Format

SYS$SUBSYSTEM enbflg

Argument

enbflg

OpenVMS usage:	boolean
type:	longword (unsigned)
access:	read only
mechanism:	by value

Value specifying whether the protected subsystem identifiers are to be saved or restored. If the **enbflg** argument is set to 0, the active subsystem is saved. If it is set to 1, the subsystem is restored.

Description

A protected subsystem image is a main image that has in its access control list a special type of ACE that names a set of identifiers and their attributes. Whenever the operating system activates a main image that has protected subsystem identifiers associated with it, these identifiers are automatically granted to the process for the duration of the image.

In essence, a protected subsystem provides the same behavior as if the image had been installed with the identifiers. Subsystem identifiers are sometimes referred to as image rights, in contrast to process rights and system rights.

The Subsystem service provides an easy way for a protected subsystem image to dynamically save and restore its subsystem identifiers. A protected subsystem may choose to turn off its subsystem identifiers at certain times to temporarily revoke the user's access to the objects comprising the protected subsystem. For example, DCL uses the $SUBSYSTEM service to temporarily remove any image identifiers from the process during Ctrl/Y interrupt processing.

The image rights are saved in the process control region and automatically deleted on image rundown ($RMSRUNDWN).

For more information about protected subsystems, see the *OpenVMS Guide to System Security*.

Required Access or Privileges

None

Required Quota

None

Related Services

None

Condition Values Returned

SS$_WASCLR	The service completed successfully; protected subsystem was not active.
SS$_WASSET	The service completed successfully; protected subsystem was active.

$SUSPND
Suspend Process

Allows a process to suspend itself or another process.

Format

SYS$SUSPND [pidadr] ,[prcnam] ,[flags]

Arguments

pidadr

OpenVMS 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. The **pidadr** argument can refer to a process running on the local node or a process running on another node in the VMScluster system.

You must specify the **pidadr** argument to suspend a process whose UIC group number is different from that of the calling process.

prcnam

OpenVMS 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 the process name. A process running on the local node can be identified with a 1- to 15-character string. To identify a process on a particular node on a cluster, specify the full process name, which includes the node name as well as the process name. The full process name can contain up to 23 characters.

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.

flags

OpenVMS 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 is 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. 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

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. If you specify neither the **pidadr** nor the **prcnam** argument, the caller process is suspended.

If the longword value at address **pidadr** is 0, the PID of the target process is returned.

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.

Required Access or Privileges

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

Required Quota

None

Related Services

$CANEXH, $CREPRC, $DCLEXH, $DELPRC, $EXIT, $FORCEX, $GETJPI, $GETJPIW, $HIBER, $PROCESS_SCAN, $RESUME, $SETPRI, $SETPRN, $SETPRV, $SETRWM, $WAKE

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$_INCOMPAT	The remote node is running an incompatible version of the operating system.
SS$_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$_NOPRIV	The target process was not created by the caller and the calling process does not have GROUP or WORLD privilege, or flag bit 0 was set from outer mode.
SS$_NOSUCHNODE	The process name refers to a node that is not currently recognized as part of the VMScluster system.
SS$_NOSUSPEND	The process was previously marked as not suspendable by the PCB$V_NOSUSPEND flag.
SS$_REMRSRC	The remote node has insufficient resources to respond to the request. (Bring this error to the attention of your system manager.)
SS$_UNREACHABLE	The remote node is a member of the cluster but is not accepting requests. (This is normal for a brief period early in the system boot process.)
SS$_WAIT_CALLERS_MODE	Bit 1 was used in the **flags** argument.

$SYNCH
Synchronize

Checks the completion status of a system service that completes asynchronously.

On Alpha systems, this service accepts 64-bit addresses.

Format

SYS$SYNCH [efn] ,[iosb]

Arguments

efn

OpenVMS 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

OpenVMS usage:	io_status_block
type:	quadword (unsigned)
access:	read only
mechanism:	by 32-bit or 64-bit reference (Alpha) by 32-bit reference (VAX)

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 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.

This service performs a true test for the completion of an asynchronous service, such as $GETJPI. $SYNCH operates in the following way:

1. When called, $SYNCH waits (by calling $WAITFR) 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 the value 0, 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 $CLREF) 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 conflicting. 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.

Required Access or Privileges

None

Required Quota

None

Condition Values Returned

SS$_NORMAL	The service completed successfully. The asynchronous service has completed, and the I/O status block contains the condition value describing the completion status of the asynchronous service.
SS$_ACCVIO	The I/O status block cannot be read by the caller.
SS$_ILLEFC	An illegal event flag was specified.
SS$_UNASEFC	The process is not associated with the cluster containing the specified event flag.

$TIMCON
Time Converter

Converts 128-bit Coordinated Universal Time (UTC) format to 64-bit system format or 64-bit system format to 128-bit UTC format based on the value of the convert flag.

Format

SYS$TIMCON [smnadr] ,[utcadr] ,cvtflg

Arguments

smnadr

OpenVMS usage:	date_time
type:	quadword (unsigned)
access:	read/write
mechanism:	by reference

The 64-bit system format value that $TIMCON will use in the conversion. The **smnadr** argument will be read from or written to based on the value of the **cvtflg** argument. The **smnadr** is required when converting UTC time to 64-bit system format.

utcadr

OpenVMS usage:	coordinated universal time
type:	utc_date_time
access:	read/write
mechanism:	by reference

UTC time value that $TIMCON will use in the conversion. The **utcadr** argument will be read from or written to based on the value of the **cvtflg** argument. The **utcadr** argument is required when converting 64-bit system format to UTC time.

cvtflg

OpenVMS usage:	conversion flag
type:	longword (unsigned)
access:	read only
mechanism:	by value

A longword indicating the direction of the conversion. If the **cvtflg** value is 0, UTC time is converted to 64-bit system value. If the **cvtflg** value is 1, 64-bit system format is converted to UTC time.

Description

The Time Converter service converts 64-bit system format time to UTC format, and vice versa.

When converting a 64-bit system format time to 128-bit UTC format time, the time zone of the local system is used.

When converting a 128-bit UTC format time to a 64-bit system time, the time zone differential factor encoded in the 128-bit buffer is used.

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_INVTIME	The input time cannot be converted because its value is out of the legal range or is a delta time, or the UTC is of an illegal format.

$TRNLNM
Translate Logical Name

Returns information about a logical name.

Format

SYS$TRNLNM [attr] ,tabnam ,lognam ,[acmode] ,[itmlst]

Arguments

attr
OpenVMS 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. 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, use its symbolic name or set its corresponding bit. All undefined bits in the longword have the value 0.

If you do not specify this argument or specify it as the value 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
OpenVMS 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
OpenVMS 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

OpenVMS 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.

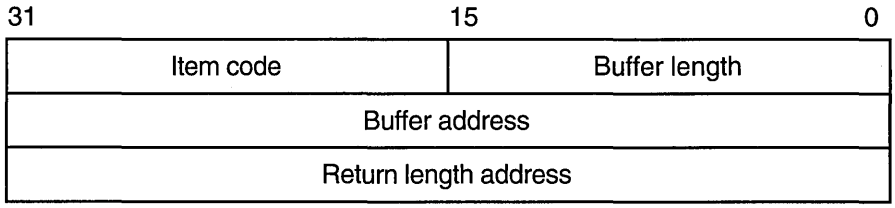
If you do not specify **acmode**, $TRNLNM performs the translation without regard to access mode; however, the translation process proceeds from the outermost to the innermost access modes. Thus, if two logical names with the same name but at different access modes exist in the same table, $TRNLNM translates the name with the outermost access mode.

itmlst

OpenVMS 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–5186A–GE

The following table defines the item descriptor fields.

Descriptor Field	Definition
Buffer length	A word specifying the number of bytes in the buffer pointed to by the buffer address field.

Descriptor Field	Definition
Item code	A word containing a symbolic code describing the nature of the information currently in the buffer, to be returned in the buffer, or to be returned by the buffer pointed to by the buffer address field.
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 specifying the actual length (in bytes) of the information returned by $TRNLNM in the buffer pointed to by the buffer address field.

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 $TRNLNM sets this bit, the equivalence name at the current index value for the logical name is a concealed logical name, as interpreted by OpenVMS RMS.
LNM$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.

Attribute	Description
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 can 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 the value 0 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 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

The Translate Logical Name service returns information about a logical name. You need read access to a shareable logical name table to translate a logical name located in that shareable logical name table.

Required Access or Privileges

Read access is required.

Required Quota

None

Related Services

$ADJSTK, $ADJWSL, $CRETVA, $CRMPSC, $DELTVA, $DGBLSC, $EXPREG, $LCKPAG, $LKWSET, $MGBLSC, $PURGWS, $SETPRT, $SETSTK, $SETSWM, $ULWSET, $UPDSEC, $UPDSECW

Condition Values Returned

SS$_NORMAL	The service completed successfully. An equivalence name for the logical name has been found.
SS$_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$_TOOMANYLNAM	Logical name translation of the table name exceeded the allowable depth (10 translations).

$TSTCLUEVT (Alpha Only) Test Cluster Event

On Alpha systems, simulates the occurrence of a cluster configuration event to test the functionality of the notification AST.

Format

SYS$TSTCLUEVT [handle] ,[acmode] ,[event]

Arguments

handle

OpenVMS usage: identifier
type: quadword (unsigned)
access: read only
mechanism: by reference

Identification of the AST to be canceled. The **handle** argument uniquely identifies the request and is returned when the $SETCLUEVT service is called.

acmode

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

Access mode for which a configuration event AST is to be triggered. The **acmode** argument is a longword containing the access mode.

Each access mode has a symbolic name. 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

event

OpenVMS usage: event_code
type: longword (unsigned)
access: read only
mechanism: by value

Event code indicating the type of configuration for which an AST is to be triggered.

Each event type has a symbolic name. The $CLUEVTDEF macro defines the following symbolic names.

Symbolic Name	Description
CLUEVT$C_ADD	One or more OpenVMS nodes have been added to the VMScluster system.
CLUEVT$C_REMOVE	One or more OpenVMS nodes have been removed from the VMScluster system.

Description

The Test Cluster Event service simulates the occurrence of a cluster configuration event to test the functionality of the notification ASTs. The service allows an application to test itself and must be issued from within the same process as the application being tested. $TSTCLUEVT does not affect other processes in the cluster.

The service will allow one specific AST to be fired via the **handle** argument, or all ASTs for a specific configuration event via the **event** argument. Specifying both the **event** and the **handle** arguments will return an error.

If the **handle** argument is specified, the value of the **acmode** argument must not be greater than the access mode of the caller and must match the mode specified when the $SETCLUEVT service was called.

If the **event** argument is specified, those ASTs that match the value specified in the **acmode** argument, or that match the caller's mode, will be triggered.

Required Access or Privileges

None

Required Quota

None

Related Services

$CLRCLUEVT, $SETCLUEVT

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_BADPARAM	There is an unsatisfactory combination of event and handle parameters, or the event was specified incorrectly.
SS$_NOSUCHOBJ	No request was found that matches the description supplied.

$ULKPAG
Unlock Pages from Memory

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]

Arguments

inadr

OpenVMS 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 byte-within-page 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 $ULKPAG. The condition value returned by $ULKPAG indicates whether the page was previously unlocked.

retadr

OpenVMS 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

OpenVMS 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

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.

Alpha

On Alpha systems, if you are attempting to unlock executable code, you should issue multiple $ULKPAG calls: one to unlock the code pages and others to unlock the linkage section references to these pages.♦

Required Access or Privileges

To call the $ULKPAG service, a process must have PSWAPM privilege.

Required Quota

None

Related Services

For more information, see the chapter on memory management in the *OpenVMS Programming Concepts Manual*.

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.

$ULKPAG_64 (Alpha Only) Unlock Pages from Memory

On Alpha systems, unlocks pages that were previously locked in memory by the Lock Pages in Memory ($LCKPAG_64) service.

This service accepts 64-bit addresses.

Format

SYS$ULKPAG_64 start_va_64 ,length_64 ,acmode ,return_va_64 ,return_length_64

Arguments

start_va_64

OpenVMS usage:	address
type:	quadword address
access:	read only
mechanism:	by value

The starting virtual address of the pages to be unlocked. The specified virtual address will be rounded down to a CPU-specific page boundary.

length_64

OpenVMS usage:	byte count
type:	quadword (unsigned)
access:	read only
mechanism:	by value

Length of the virtual address space to be unlocked. The specified length will be rounded up to a CPU-specific page boundary so that it includes all CPU-specific pages in the requested range.

acmode

OpenVMS 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 $PSLDEF macro in STARLET.MLB and the file PSLDEF.H in SYS$STARLET_C.TLB define the following symbols and their values for the four access modes:

Value	Symbolic Name	Access Mode
0	PSL$C_KERNEL	Kernel
1	PSL$C_EXEC	Executive
2	PSL$C_SUPER	Supervisor
3	PSL$C_USER	User

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.

return_va_64
OpenVMS usage: address
type: quadword address
access: write only
mechanism: by 32-bit or 64-bit reference

The lowest process virtual address of the unlocked virtual address range. The **return_va_64** argument is the 32-bit or 64-bit virtual address of a naturally aligned quadword into which the service returns the virtual address.

return_length_64
OpenVMS usage: byte count
type: quadword (unsigned)
access: write only
mechanism: by 32-bit or 64-bit reference

The length of the virtual address range unlocked. The **return_length_64** argument is the 32-bit or 64-bit virtual address of a naturally aligned quadword into which the service returns the length of the virtual address range in bytes.

Description

The Unlock Pages from Memory service unlocks pages that were previously locked in memory by the Lock Pages in Memory ($LCKPAG_64) service.

If the condition value SS$_ACCVIO is returned by this service, a value *cannot* be returned in the memory locations pointed to by the **return_va_64** and **return_length_64** arguments.

If a condition value other than SS$_ACCVIO is returned, the returned address and returned length indicate the pages that were successfully unlocked before the error occurred. If no pages were unlocked, the **return_va_64** argument will contain the value -1, and a value *cannot* be returned in the memory location pointed to by the **return_length_64** argument.

Required Privileges

To call the $ULKPAG_64 service, a process must have PSWAPM privilege.

Required Quota

None.

Related Services

$LCKPAG_64, $ULKPAG

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 **return_va_64** or **return_length_64** argument cannot be written by the caller, or an attempt was made to unlock pages by a caller whose access mode is less privileged than the access mode associated with the pages.

$ULWSET
Unlock Pages from Working Set

Unlocks pages that were previously locked in the working set by the Lock Pages in Working Set ($LKWSET) service.

Format

SYS$ULWSET inadr ,[retadr] ,[acmode]

Arguments

inadr

OpenVMS 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 byte-within-page 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

OpenVMS 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

OpenVMS 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

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.

Alpha

On Alpha systems, if you are attempting to unlock executable code, you should issue multiple $ULKWSET calls: one to unlock the code pages and others to unlock the linkage section references to these pages.♦

Required Access or Privileges

None

Required Quota

None

Related Services

$ADJSTK, $ADJWSL, $CRETVA, $CRMPSC, $DELTVA, $DGBLSC, $EXPREG, $LCKPAG, $LKWSET, $MGBLSC, $PURGWS, $SETPRT, $SETSTK, $SETSWM, $ULKPAG, $UPDSEC, $UPDSECW

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.

$ULWSET_64 (Alpha Only) Unlock Pages in Working Set

On Alpha systems, unlocks a virtual address range that was previously locked in the working set by the Lock Pages in Working Set ($LKWSET_64) service.

This service accepts 64-bit addresses.

Format

SYS$ULWSET_64 start_va_64 ,length_64 ,acmode ,return_va_64 ,return_length_64

Arguments

start_va_64
OpenVMS usage: address
type: quadword address
access: read only
mechanism: by value

The starting virtual address of the pages to be unlocked from the working set. The specified virtual address will be rounded down to a CPU-specific page boundary.

length_64
OpenVMS usage: byte count
type: quadword (unsigned)
access: read only
mechanism: by value

Length of the virtual address space to be unlocked from the working set. The specified length will be rounded up to a CPU-specific page boundary so that it includes all CPU-specific pages in the requested range.

acmode
OpenVMS 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 $PSLDEF macro in STARLET.MLB and the file PSLDEF.H in SYS$STARLET_C.TLB define the following symbols and their values for the four access modes:

Value	Symbolic Name	Access Mode
0	PSL$C_KERNEL	Kernel
1	PSL$C_EXEC	Executive
2	PSL$C_SUPER	Supervisor
3	PSL$C_USER	User

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.

return_va_64

OpenVMS usage:	address
type:	quadword address
access:	write only
mechanism:	by 32-bit or 64-bit reference

The lowest process virtual address of the unlocked virtual address range. The **return_va_64** argument is the 32-bit or 64-bit virtual address of a naturally aligned quadword into which the service returns the virtual address.

return_length_64

OpenVMS usage:	byte count
type:	quadword (unsigned)
access:	write only
mechanism:	by 32-bit or 64-bit reference

The length of the virtual address range unlocked. The **return_length_64** argument is the 32-bit or 64-bit virtual address of a naturally aligned quadword into which the service returns the length of the virtual address range in bytes.

Description

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_64) service. Unlocked pages become candidates for replacement within the working set of the process.

If the condition value SS$_ACCVIO is returned by this service, a value *cannot* be returned in the memory locations pointed to by the **return_va_64** and **return_length_64** arguments.

If a condition value other than SS$_ACCVIO is returned, the returned address and returned length indicate the pages that were successfully unlocked before the error occurred. If no pages were unlocked, the **return_va_64** argument will contain the value -1, and a value *cannot* be returned in the memory location pointed to by the **return_length_64** argument.

Required Privileges

None.

Required Quota

None.

Related Services

$LKWSET_64, $PURGE_WS, $ULWSET

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$_ACCVIO	The **return_va_64** or **return_length_64** argument cannot be written by the caller, or an attempt was made to unlock pages by a caller whose access mode is less privileged than the access mode associated with the pages.
SS$_PAGNOTINREG	A page in the specified range is not within process private address space.

$UNWIND
Unwind Call Stack

Unwinds the procedure call stack.

Format

SYS$UNWIND [depadr] ,[newpc]

Arguments

depadr

OpenVMS 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); the value 1 specifies the caller of that frame; the value 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

OpenVMS usage:	address
type:	longword (unsigned)
access:	read only
mechanism:	by value

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 Unwind Call Stack service unwinds the procedure call stack; that is, it removes a specified number of call frames from the stack. Optionally, it can 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.

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 cleanup operations that might be required. After the condition handler returns, the call frame is removed from the stack.

Required Access or Privileges

None

Required Quota

None

Related Services

$DCLCMH, $SETEXV, $SETSFM

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

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]

Arguments

inadr

OpenVMS 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. Addresses are adjusted up or down to CPU-specific pages. Only the virtual page number portion of each virtual address is used; the low-order byte-within-page bits are ignored.

$UPDSEC 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 might be smaller than the address specified by the first longword.

retadr

OpenVMS 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. Addresses always are adjusted up or down to fall on CPU-specific boundaries.

If $UPDSEC returns an error condition value in R0, each longword specified by **retadr** contains 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

OpenVMS 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

OpenVMS 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 or not they have been modified. 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

OpenVMS 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

OpenVMS 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 remaining bits of the second word are zeros.
- 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 $SYNCH 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 $UPDSEC. 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

OpenVMS usage:	ast_procedure
type:	procedure value
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 this routine.

If you specify **astadr**, the AST routine executes at the access mode from which the section file update was requested.

astprm

OpenVMS 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 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.

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.

VAX

On VAX systems, 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 $UPDSEC. Processes on another processor that attempt to update the section file receive an error condition value indicating that the request was not performed.♦

Required Access or Privileges

None

Required Quota

$UPDSEC 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.

Related Services

$ADJSTK, $ADJWSL, $CRETVA, $CRMPSC, $DELTVA, $DGBLSC, $EXPREG, $LCKPAG, $LKWSET, $MGBLSC, $PURGWS, $SETPRT, $SETSTK, $SETSWM, $ULKPAG, $ULWSET, $UPDSECW

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 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 multiport memory as shared at system generation time.

†VAX specific

SS$_UNASCEFC	The process is not associated with the cluster containing the specified event flag.

$UPDSEC_64 (Alpha Only) Update Global Section File on Disk

On Alpha systems, writes all pages (or only those pages modified by the current process) in an active private or global disk file section back into the section file on disk. One or more I/O requests are queued to perform the write operation.

The $UPDSEC_64 service completes asynchronously. For synchronous completion, use the Update Global Section File on Disk and Wait ($UPDSEC_64W) service.

This service accepts 64-bit addresses.

Format

SYS$UPDSEC_64 start_va_64 ,length_64 ,acmode ,updflg ,efn ,iosa_64 ,return_va_64 ,return_length_64 [,astadr_64 [,astprm_64]]

Arguments

start_va_64

OpenVMS usage: address
type: quadword address
access: read only
mechanism: by value

The starting virtual address of the pages to be written to the section file. The specified virtual address is rounded down to a CPU-specific page boundary.

length_64

OpenVMS usage: byte count
type: quadword (unsigned)
access: read only
mechanism: by value

Length of the virtual address range to be written to the section file. The length specified is rounded up to a CPU-specific page boundary so that it includes all CPU-specific pages in the requested range.

acmode

OpenVMS 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 $PSLDEF macro in STARLET.MLB and the file PSLDEF.H in SYS$STARLET_C.TLB define the following symbols and their values for the four access modes:

Value	Symbolic Name	Access Mode
0	PSL$C_KERNEL	Kernel

Value	Symbolic Name	Access Mode
1	PSL$C_EXEC	Executive
2	PSL$C_SUPER	Supervisor
3	PSL$C_USER	User

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_64 is equal to or more privileged than the access mode of the owner of the page to be written.

updflg

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

The 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 or not they have been modified. The value UPDFLG$M_WRT_MODIFIED specifies that the caller is the only process actually writing the global section and that only those pages that were actually modified by the caller are to be written to the section file on disk.

Definitions for this flag can be found in the file SECDEF.H in SYS$STARLET_C.TLB for C and in $SECDEF in STARLET.MLB for macro.

efn

OpenVMS usage: ef_number
type: longword (unsigned)
access: read_only
mechanism: by value

The 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, this service only uses the low-order byte. If you do not specify the **efn**, event flag 0 is used.

When you invoke $UPDSEC_64, the specified event flag or event flag 0 is cleared. When the update operation is complete, the event flag is set.

iosa_64

OpenVMS usage: io_status_area
type: IOSA structure
access: write only
mechanism: by 32-bit or 64-bit reference

The I/O status area to receive the final completion status of the updating operation. The **iosa_64** argument is the 32-bit or 64-bit virtual address of the I/O status area. The I/O status area structure is 32 bytes in length. The I/O status area structure definition can be found in $IOSADEF in STARLET.MLB for macro and in the file IOSADEF.H in SYS$STARLET_C.TLB for C.

When you call SYS$UPDSEC_64, the I/O status area 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:

- isoa$l_status (offset 0)

The first word contains the condition value return by SYS$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 remaining bits of the second word are zeros.

- iosa$l_resd (offset 4)

  This field is reserved for future use by Digital. The value in this field is unpredictable.

- iosa$q_count_q (offset 8)

  This field is reserved for future use by Digital. The value in this field is unpredictable.

- iosa$ph_upsec_nowrt_va (offset 16)

  This field contains the virtual address of the first byte in the first disk block that was not written. In the case of an I/O error, this virtual address indicates the disk block for which the error occurred.

- iosa$q_resq (offset 24)

  This field is reserved for future use by Digital. The value in this field is unpredictable.

return_va_64

OpenVMS usage:	address
type:	quadword address
access:	write only
mechanism:	by 32-bit or 64-bit reference

The process virtual address of the first page that was actually queued for writing (in the first I/O request) back to the section file on the disk. The **return_va_64** argument is the 32-bit or 64-bit virtual address of a naturally aligned quadword into which the service returns the virtual address.

return_length_64

OpenVMS usage:	byte count
type:	quadword (unsigned)
access:	write only
mechanism:	by 32-bit or 64-bit reference

The length of the first I/O request to write modified pages back to the section file on disk. The **return_length_64** argument is the 32-bit or 64-bit virtual address of a naturally aligned quadword into which the service returns the length of the virtual address range, in bytes, written by the first I/O request.

astadr_64

OpenVMS usage:	ast_procedure
type:	procedure value
access:	call without stack unwinding
mechanism:	by 32-bit or 64-bit reference

The AST routine to be executed when the section file has been updated. The **astadr_64** argument is the 32-bit or 64-bit address of this routine. If you specify the **astadr_64** argument, the AST routine executes at the access mode from which the section file update was requested.

astprm_64

OpenVMS usage:	user_arg
type:	quadword
access:	read only
mechanism:	by value

The AST parameter to be passed to the AST routine. The **astprm_64** argument is a quadword argument that is passed to the AST routine.

Description

The Update Global Section File on Disk service writes all pages in an active private or global section back into the section file on disk. If the **updflg** argument indicates that only modified pages are to be written back to the disk file, only those global pages modified by the current process are queued to be written back into the section file on disk.

Proper use of this service requires the caller to synchronize completion of the update request. To do this, first check the condition value returned. 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 I/O status for final completion status.

If any error is returned by this service, a value **cannot** be returned in the memory locations pointed to by the **iosb_64**, **return_va_64**, and **return_length_64** arguments.

Required Privileges

None

Required Quota

$UPDSEC_64 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_64** argument is specified.

Related Services

$CRMPSC, $CRMPSC_FILE_64, $CRMPSC_GFILE_64, $CRMPSC_GPFILE_64, $MGBLSC_64, $UPDSEC

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$_ACCVIO	The **return_va_64**, **return_length_64**, or **iosb_64** argument cannot be written by the caller.
SS$_EXASTLM	The process has exceeded its AST limit quota.
SS$_EXBYTLM	The process has exceeded the byte count quota.
SS$_ILLEFC	An illegal event flag number was specified.

SS$_PAGNOTINREG	A page in the specified range is not within the process private address space.
SS$_PAGOWNVIO	A page in the specified input address range is owned by a more privileged access mode.
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, 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 description of $UPDSEC.

For additional information about system service completion, refer to the Synchronize ($SYNCH) service.

Format

```
SYS$UPDSECW   inadr [,retadr] [,acmode] [,updflg] [,efn] [,iosb] [,astadr] [,astprm]
```

$UPDSEC_64W (Alpha Only)
Update Global Section File on Disk and Wait

On Alpha systems, writes all modified pages in an active private or global disk file section back into the section file on disk. Zero or more I/O requests are queued, based on the number of pages that have been modified.

The $UPDSEC_64W service completes synchronously; that is, it returns to the caller after writing all updated pages.

In all other respects, $UPDSEC_64W is identical to $UPDSEC_64. For all other information about the $UPDSEC_64W service, refer to the description of $UPDSEC_64 in this manual.

This service accepts 64-bit addresses.

Format

```
SYS$UPDSEC_64W  start_va_64 ,length_64 ,acmode ,updflg ,efn ,iosa_64
                ,return_va_64 ,return_length_64 [,astadr_64 [,astprm_64]]
```

$VERIFY_PROXY
Verify a Proxy

Verifies that a proxy exists and returns a valid local user for the caller to use to create a local login.

Format

SYS$VERIFY_PROXY rem_node ,rem_user ,[proposed_user] ,local_user ,local_user_length ,[flags]

Arguments

rem_node

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

Remote node name of the proxy to be verified. The **rem_node** argument is the address of a character-string descriptor pointing to the remote node name string.

A remote node name consists of 1 to 1024 characters. No specific characters, format, or case are required for a remote node name string. All node names are converted to their DECnet for OpenVMS full name unless the PRX$M_BYPASS_EXPAND flag is set with the **flags** argument.

Wildcards are not recognized. If you specify a wildcard character in the **rem_node** argument, it is ignored and assumed to be part of the requested node name.

rem_user

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

Remote user name of the proxy to be verified. The **rem_user** argument is the address of a character-string descriptor pointing to the user name string.

A remote user name consists of 1 to 32 alphanumeric characters, including dollar signs ($), underscores (_), and brackets ([ ]). Any lowercase characters specified are automatically converted to uppercase.

The **rem_user** argument can be specified in user identification code (UIC) format ([*group*, *member*]). Brackets are allowed only if the remote user name string specifies a UIC. Group and member are character-string representations of octal numbers with no leading zeros.

Wildcards are not allowed for the remote user specification. If wildcard characters are present in the string specified by the **rem_user** argument, the service returns SS$_BADPARAM.

proposed_user

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

Local user the caller suggests be used for the proxy login. The **proposed_user** argument is the address of a character-string descriptor pointing to the proposed local user name.

The proposed local user consists of 1 to 32 alphanumeric characters, including dollar signs ($) and underscores (_). Any lowercase characters specified are automatically converted to uppercase.

See the Description section for information about the interaction of this argument with the return value of the **local_user** argument.

local_user

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

Local user the caller must use for a proxy login. The **local_user** argument is the address of a 32-byte character-string descriptor pointer to receive the local user name the caller must use for a proxy login for the proxy with the remote node name specified by the **rem_node** argument and the remote user name specified by the **rem_user** argument.

A local user name is a 32-character blank padded string of alphanumeric characters, including dollar signs ($) and underscores (_).

local_user_length

OpenVMS usage:	output length
type:	word (unsigned)
access:	write only
mechanism:	by reference

Length of the returned local user name in the **local_user** argument. The **local_user_length** argument is the address of an unsigned word to receive the length, in bytes, of the character string returned in the **local_user** argument.

flags

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

Functional specification for the service and type of user the **local_user** argument represents. The **flags** argument is a longword bit mask wherein each bit corresponds to an option.

Each flag option has a symbolic name. The $PRXDEF macro defines the following symbolic name.

Symbolic Name	Description
PRX$M_BYPASS_EXPAND	The service should not convert the node name specified in the **rem_node** argument to its corresponding DECnet for OpenVMS full name. If this flag is set, it is the caller's responsibility to ensure that the fully expanded node name is passed into the service.

Description

The Verify Proxy service verifies the existence of a proxy in the proxy database and returns the local user name the caller must use for any proxy logins.

The following description shows how the service determines which local user name the caller must use for proxy logins.

Proxies that match the remote node and remote user specified by the **rem_node** and **rem_user** arguments, respectively, are searched in the following order if the remote user name *is not* a UIC:

1. rem_node::rem_user
2. *::rem_user
3. rem_node::*
4. *::*

Proxies that match the remote node and remote user specified by the **rem_node** and **rem_user** arguments, respectively, are searched for in the following order if the remote user name *is* a UIC:

1. rem_node::rem_user
2. *::rem_user
3. rem_node::[group,*]
4. rem_node::[*,member]
5. rem_node::[*,*]
6. *::*

The following table describes how the local user name the caller must use for any proxy logins is determined if a matching proxy record is found by the search.

Remote User	Proposed User	Proxy Default User	Proxy Local User Names	Returned Local User Name
rem_user	null	null	n/a	error
rem_user	null	default user	n/a	default user
rem_user	null	*	n/a	rem_user
rem_user	prop_user	default user	n/a	prop_user
rem_user	prop_user	default user	prop_user	prop_user

Remote User	Proposed User	Proxy Default User	Proxy Local User Names	Returned Local User Name
rem_user	prop_user	default user	local user	error
rem_user	prop_user	default user	*	rem_user if it equals prop_user
rem_user	prop_user	*	local user	rem_user if it equals prop_user

Required Access or Privileges

You must have SYSPRV privilege.

Required Quota

None

Related Services

$ADD_PROXY, $DELETE_PROXY, $DISPLAY_PROXY

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The **rem_node**, **rem_user**, or **proposed_user** argument cannot be read by the service; or the **local_user** or **local_user_length** argument cannot be written by the service.
SS$_BADBUFLEN	The length of the **rem_node**, **rem_user**, **proposed_user**, or **local_user** argument was out of range.
SS$_BADPARAM	The **rem_user** or **proposed_user** argument contains an invalid user name.
SS$_NOREADALL	The caller does not have access to the proxy database.

This service can also return any of the following messages passed from the security server, or any OpenVMS RMS error message encountered during operations on the proxy database:

SECSRV$_ BADLOCALUSERLEN	The local user name length is out of range.
SECSRV$_ BADNODENAMELEN	The node name length is out of range.
SECSRV$_ BADREMUSERLEN	The remote user name length is out of range.
SECSRV$_NOSUCHPROXY	The proxy specified by the **rem_node** and **rem_user** arguments does not exist in the proxy database.
SECSRV$_NOSUCHUSER	No valid user was found for the requested proxy.
SECSRV$_ PROXYNOTACTIVE	Proxy processing is currently stopped. Try the request again later.

SECSRV$_ SERVERNOTACTIVE	The security server is not currently active. Try the request again later.

$WAITFR
Wait for Single Event Flag

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

Argument

efn

OpenVMS 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 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. 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, the operating system repeats the $WAITFR request on behalf of the process. At this point, if the event flag has been set, the process resumes execution.

Required Access or Privileges

None

Required Quota

None

Related Services

$ASCEFC, $CLREF, $DACEFC, $DLCEFC, $READEF, $SETEF, $WFLAND, $WFLOR

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

Activates a process that has placed itself in a state of hibernation with the Hibernate ($HIBER) service.

Format

SYS$WAKE [pidadr] ,[prcnam]

Arguments

pidadr

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

Process identification (PID) of the process to be activated. The **pidadr** argument is the address of a longword containing the PID. The **pidadr** argument can refer to a process running on the local node or a process running on another node in the VMScluster system.

prcnam

OpenVMS 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 activated. The **prcnam** argument is the address of a character string descriptor pointing to the process name. A process running on the local node can be identified with a 1- to 15-character string. To identify a process on a particular node in a cluster, specify the full process name, which includes the node name as well as the process name. The full process name can contain up to 23 characters.

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 activated is in the same UIC group as the calling process. To activate a process in another UIC group, you must specify the **pidadr** argument.

Description

The Wake Process from Hibernation service activates a process that has placed itself in a state of hibernation with the Hibernate ($HIBER) service. If you specify neither the **pidadr** nor the **prcnam** argument, the wake request is issued for the calling process.

If the longword at address **pidadr** is the value 0, the PID of the target process is returned.

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 of outstanding wakeup requests is maintained.

You can also activate a hibernating process with the Schedule Wakeup ($SCHDWK) service.

Required Access or Privileges

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

Required Quota

None

Related Services

$CANEXH, $CREPRC, $DCLEXH, $DELPRC, $EXIT, $FORCEX, $GETJPI, $GETJPIW, $HIBER, $PROCESS_SCAN, $RESUME, $SETPRI, $SETPRN, $SETPRV, $SETRWM, $SUSPND

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$_INCOMPAT	The remote node is running an incompatible version of the operating system.
SS$_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.
SS$_NOSUCHNODE	The process name refers to a node that is not currently recognized as part of the VSMcluster system.
SS$_REMRSRC	The remote node has insufficient resources to respond to the request. (Bring this error to the attention of your system manager.)
SS$_UNREACHABLE	The remote node is a member of the cluster but is not accepting requests. (This is normal for a brief period early in the system boot process.)

$WFLAND
Wait for Logical AND of Event Flags

Allows a process to specify a set of event flags for which it wants to wait.

Format

SYS$WFLAND efn ,mask

Arguments

efn

OpenVMS 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

OpenVMS 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 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.

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, the operating system 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.

Required Access or Privileges

None

Required Quota

None

Related Services

$ASCEFC, $CLREF, $DACEFC, $DLCEFC, $READEF, $SETEF, $WAITFR, $WFLOR

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

Allows a process to specify a set of event flags for which it wants to wait.

Format

SYS$WFLOR efn ,mask

Arguments

efn

OpenVMS 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

OpenVMS 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 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.

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, the operating system repeats the $WFLOR request on behalf of the process. At this point, if any of the specified event flags has been set, the process resumes execution.

Required Access or Privileges

None

Required Quota

None

Related Services

$ASCEFC, $CLREF, $DACEFC, $DLCEFC, $READEF, $SETEF, $WAITFR, $WFLAND

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 *OpenVMS Obsolete Features Manual.*

Obsolete Service	Current Service
$BRDCST	$BRKTHRU, $BRKTHRUW
$CHANGE_ACL	$GET_SECURITY, $SET_SECURITY
$CNTREG	$DELTVA
$CRELOG	$CRELNM
$DELLOG	$DELLNM
$GETCHN	$GETDVI, $GETDVIW
$GETDEV	$GETDVI, $GETDVIW
$INPUT	$QIO, $QIOW
$OUTPUT	$QIO, $QIOW
$SETSFM	This service is still supported but its use is discouraged
$SETSSF	This service is still supported but its use is discouraged
$SNDACC	$SNDJBC, $SNDJBCW
$SNDSMB	$SNDJBC, $SNDJBCW
$TRNLOG	$TRNLNM

Index

A

Aborting a transaction, SYS1–3
$ABORT_TRANS system service, SYS1–3
$ABORT_TRANSW system service, SYS1–7
Absolute time
  as input to $BINTIM, SYS1–62
  as input to $BINUTC, SYS1–65
  converting to numeric, SYS2–194
Access
  checking, SYS1–84
Access modes
  changing to executive, SYS1–110, SYS1–112
  changing to kernel, SYS1–114, SYS1–116
Access protection
  checking, SYS1–99
Accounting messages
  format of, SYS1–177
ACII character set
  converting strings to binary, SYS1–62
ACLs (access control lists)
  formatting, SYS1–389
Adding holder records to rights database, SYS1–8
Adding identifiers to rights database, SYS1–11
Address space
  creating virtual, SYS1–188
$ADD_HOLDER system service, SYS1–8
$ADD_IDENT system service, SYS1–11
$ADD_PROXY system service, SYS1–14
$ADJSTK system service, SYS1–18
$ADJWSL system service, SYS1–20
Alignment fault data
  getting for system process, SYS2–97
  getting for user image, SYS2–85
Alignment fault reporting
  disabling for user image, SYS2–440
  disabling for user process, SYS2–201
  enabling for user process, SYS2–202
  initializing for system process, SYS2–115
  starting for user image, SYS2–432
Allocating devices, SYS1–22
Allocation classes, SYS1–410
$ALLOC system service, SYS1–22
Arithmetic exceptions
  getting information about, SYS2–87
$ASCEFC system service, SYS1–25
ASCII character set
  converting strings to UTC, SYS1–65
ASCII output
  formatting character string, SYS1–351
ASCII strings
  converting to binary, SYS1–62
  converting to UTC, SYS1–65
$ASCTIM system service, SYS1–29
$ASCTOID system service, SYS1–32
$ASCUTC system service, SYS1–35
Assigning an I/O channel, SYS1–38
$ASSIGN system service, SYS1–38
ASTLM (AST limit) quota
  effect of canceling wakeup on, SYS1–82
ASTs (asynchronous system traps)
  declaring, SYS1–242
  disabling, SYS2–281
  enabling, SYS2–281
  setting for power recovery, SYS2–301
  setting timer for, SYS2–294
Asynchronous system traps
  See ASTs
Audit event messages
  converting, SYS1–402
Auditing events, SYS1–43, SYS1–61
$AUDIT_EVENT system service, SYS1–43
$AUDIT_EVENTW system service, SYS1–61
Automatic unshelving
  controlling, SYS2–321
  determining, SYS1–442

B

Binary time
  converting to ASCII string, SYS1–29
  converting to numeric time, SYS2–194, SYS2–196
Binary values
  converting to ASCII string, SYS1–351
$BINTIM system service, SYS1–62
$BINUTC system service, SYS1–65
64-bit virtual addressing
  system services support, vii
$BRKTHRU system service, SYS1–68
$BRKTHRUW system service, SYS1–76
Buffer object
  creating, SYS1–122

Buffer objects
  deleting, SYS1–249
BYTLM quota
  using with $GETJPI buffers, SYS2–217

C

Call frames
  removing from stack, SYS2–468
Call stacks
  unwinding, SYS2–99
Canceling
  exit handlers, SYS1–79
  I/O requests, SYS1–77
  timer requests, SYS1–80
  wakeup requests, SYS1–82
$CANCEL system service, SYS1–77
$CANEXH system service, SYS1–79
$CANTIM system service, SYS1–80
$CANWAK system service, SYS1–82
Change mode handlers
  declaring, SYS1–244
Channels
  canceling I/O, SYS1–77
$CHECK_ACCESS system service, SYS1–84
$CHECK_FEN system service
  on Alpha systems only, SYS1–92
$CHECK_PRIVILEGE system service, SYS1–93
$CHECK_PRIVILEGEW system service, SYS1–98
$CHKPRO system service, SYS1–99
Class scheduler processes, SYS2–279
Clearing an event flag, SYS1–109
$CLRCLUEVT system service
  on Alpha systems only, SYS1–107
$CLREF system service, SYS1–109
Cluster events
  clearing request for notification of, SYS1–107
  requesting notification of, SYS2–282
$CMEXEC system service, SYS1–110
$CMEXEC_64 system service, SYS1–112
$CMKRNL system service, SYS1–114
$CMKRNL_64 system service, SYS1–116
Common event flag clusters
  disassociating, SYS1–236
Compatibility mode handlers
  declaring, SYS1–244
Control region
  adding page to, SYS1–345
  deleting page from, SYS1–265
Converting
  ASCII string to binary time, SYS1–62
  ASCII string to UTC format, SYS1–65
  audit event message, SYS1–402
  binary time to ASCII string, SYS1–29
  binary time to numeric time, SYS2–194
  64-bit system time to UTC time, SYS2–449
  UTC format to ASCII, SYS1–35
  UTC time to numeric time, SYS2–196
CPU affinity set
  modifying, SYS2–204
CPU user capability set
  modifying, SYS1–118
$CPU_CAPABILITIES system service, SYS1–118
$CREATE_BUFOBJ_64 system service, SYS1–122
  description, SYS1–124
$CREATE_GFILE system service, SYS1–126
  description, SYS1–129
$CREATE_GPFILE system service, SYS1–131
  description, SYS1–133
$CREATE_GPFN system service, SYS1–135
  description, SYS1–137
$CREATE_RDB system service, SYS1–139
$CREATE_REGION_64 system service, SYS1–141
  description, SYS1–143
$CREATE_USER_PROFILE system service, SYS1–145
Creating
  disk file sections, SYS1–192
  logical names, SYS1–149
  logical name tables, SYS1–155
  mailboxes, SYS1–161
  processes, SYS1–168
  rights databases, SYS1–139
  user profiles, SYS1–145
  virtual address space, SYS1–185
$CRELNM system service, SYS1–149
$CRELNT system service, SYS1–155
$CREMBX system service, SYS1–161
$CREPRC system service, SYS1–168
$CRETVA system service, SYS1–185
  See also $EXPREG system service
$CRETVA_64 system service, SYS1–188
  description, SYS1–190
$CRMPSC system service, SYS1–192
$CRMPSC_FILE_64 system service, SYS1–204
  description, SYS1–207
$CRMPSC_GFILE_64 system service, SYS1–210
  description, SYS1–215
$CRMPSC_GPFILE_64 system service, SYS1–218
  description, SYS1–222
$CRMPSC_GPFN_64 system service, SYS1–225
  description, SYS1–229
$CRMPSC_PFN_64 system service, SYS1–232
  description, SYS1–234

D

$DACEFC system service, SYS1–236
$DALLOC system service, SYS1–238
$DASSGN system service, SYS1–240
$DCLAST system service, SYS1–242
$DCLCMH system service, SYS1–244
$DCLEXH system service, SYS1–247

Deallocating devices, SYS1–238
Deassigning an I/O channel, SYS1–240
DECdns names
  converting, SYS1–303, SYS1–304, SYS1–305, SYS1–307
  converting full name, SYS1–303
DECdns objects
  creating, SYS1–298
  deleting, SYS1–299
  enumerating, SYS1–301
Declaring an AST (asynchronous system trap), SYS1–242
Default directories
  setting, SYS2–285
Default file protection
  setting, SYS2–287
Default form, SYS2–382
$DELETE_BUFOBJ system service, SYS1–249
  description, SYS1–249
$DELETE_INTRUSION system service, SYS1–250
$DELETE_PROXY system service, SYS1–252
$DELETE_REGION_64 system service, SYS1–255
  description, SYS1–256
Deleting
  DECdns objects, SYS1–299
  event flag clusters, SYS1–292
  global sections, SYS1–279
  intrusion records, SYS1–250
  logical names, SYS1–258
  mailboxes, SYS1–261
  processes, SYS1–263
  proxies, SYS1–252
  virtual address space, SYS1–265
$DELLNM system service, SYS1–258
$DELMBX system service, SYS1–261
$DELPRC system service, SYS1–263
Delta time
  as input to $BINTIM, SYS1–62
  converting to numeric, SYS2–194
$DELTVA system service, SYS1–265
$DELTVA_64 system service, SYS1–267
  description, SYS1–268
$DEQ system service, SYS1–270
Dequeuing lock requests, SYS1–270
Detached processes
  creating, SYS1–180
Devices
  allocating, SYS1–22
  deallocating, SYS1–238
  dual-pathed, SYS1–410
  getting information asynchronously, SYS1–406
  getting information synchronously, SYS1–426
  lock name, SYS1–414
  scanning of across the cluster, SYS1–275
  served, SYS1–418
$DEVICE_SCAN system service, SYS1–275
$DGBLSC system service, SYS1–279
Disk file sections
  creating, SYS1–192
  mapping, SYS1–192
Disks
  initializing from within a program, SYS2–118
Dismounting a volume, SYS1–282
$DISMOU system service, SYS1–282
$DISPLAY_PROXY system service, SYS1–286
$DLCEFC system service, SYS1–292
$DNS system service
  on VAX systems only, SYS1–294
$DNSW system service
  on VAX systems only, SYS1–321

E

$END_TRANS system service, SYS1–322
$END_TRANSW system service, SYS1–327
$ENQ system service, SYS1–328
$ENQW system service, SYS1–340
Equivalence names
  specifying, SYS1–149
$ERAPAT system service, SYS1–341
Error logger
  sending message to, SYS2–358
Event flag clusters
  associating with a process, SYS1–25
  deleting, SYS1–292
  disassociating, SYS1–236
  getting current status, SYS2–246
Event flags, SYS1–294
  clearing, SYS1–109
  getting current status, SYS2–246
  setting, SYS2–289
  waiting for entire set of, SYS2–490
  waiting for one of set, SYS2–492
  waiting for setting of, SYS2–487
Events
  auditing, SYS1–43, SYS1–61
Exception vectors
  setting, SYS2–290
Executive mode
  changing to, SYS1–110, SYS1–112
Exit handlers
  canceling, SYS1–79
  control block, SYS1–247
    deleting, SYS1–79
  declaring, SYS1–247
Exits
  forcing, SYS1–386
$EXIT system service, SYS1–344
  issuing for specified process, SYS1–386
Expanding program/control region, SYS1–345
$EXPREG system service, SYS1–345

$EXPREG_64 system service, SYS1–348
  description, SYS1–349

F

$FAOL system service, SYS1–351
$FAOL_64 system service, SYS1–371
$FAO system service, SYS1–351
$FAO system service directives
  format of, SYS1–353
  table of, SYS1–355
Files
  getting information asynchronously, SYS2–3
  getting information synchronously, SYS2–46
$FILESCAN system service, SYS1–372
File specifications
  parsing components of, SYS1–372
  searching string for, SYS1–372
$FIND_HELD system service, SYS1–378
$FIND_HOLDER system service, SYS1–381
$FINISH_RDB system service, SYS1–384
Floating point
  checking, SYS1–92
$FORCEX system service, SYS1–386
  See also $DELPRC and $EXIT
Forcing an exit, SYS1–386
Formatting
  ACL entry, SYS1–389
  security audit messages, SYS1–402
$FORMAT_ACL system service, SYS1–389
$FORMAT_AUDIT system service, SYS1–402
Forms
  getting information asynchronously, SYS2–3
  getting information synchronously, SYS2–46
Full names
  converting to string, SYS1–303

G

$GETDVI system service, SYS1–406
$GETDVIW system service, SYS1–426
$GETJPI system service, SYS1–427
$GETJPIW system service, SYS1–448
$GETLKI system service, SYS1–449
$GETLKIW system service, SYS1–461
$GETMSG system service, SYS1–462
$GETQUI system service, SYS2–3
$GETQUIW system service, SYS2–46
$GETSYI system service, SYS2–51
$GETSYIW system service, SYS2–70
$GETTIM system service, SYS2–71
$GETUAI system service, SYS2–72
$GETUTC system service, SYS2–84
$GET_ALIGN_FAULT_DATA system service
  on Alpha systems only, SYS2–85
$GET_ARITH_EXCEPTION system service
  on Alpha systems only, SYS2–87
$GET_REGION_INFO system service, SYS2–47
  description, SYS2–50
$GET_SECURITY system service, SYS2–89
$GET_SYS_ALIGN_FAULT_DATA system service
  on Alpha systems only, SYS2–97
Global disk file section
  creating, SYS1–126
  creating and mapping, SYS1–210
  mapping, SYS2–157
Global page file
  Create, SYS1–131
Global page file section
  creating and mapping, SYS1–218
  mapping, SYS2–157
Global page frame section
  creating and mapping, SYS1–225
  mapping, SYS2–163
Global section file
  updating on disk (asynchronously), SYS2–475
  updating on disk (synchronously), SYS2–481
Global sections
  creating, SYS1–192
  deleting, SYS1–279
  mapping, SYS1–192, SYS2–151
$GOTO_UNWIND system service
  on Alpha systems only, SYS2–99
$GRANTID system service, SYS2–101

H

$HASH_PASSWORD system service, SYS2–105
$HIBER system service, SYS2–108
  See also $WAKE
Holder records
  adding to rights database, SYS1–8
  modifying in rights database, SYS2–169
  removing from rights database, SYS2–249
Holders of an identifier
  finding, SYS1–381
Host
  checking availability of, SYS1–410

I

I/O channels
  assigning, SYS1–38
  deassigning, SYS1–240
I/O devices
  getting information asynchronously, SYS1–406
  getting information synchronously, SYS1–426
I/O requests
  canceling, SYS1–77
  queuing asynchronously, SYS2–239
  queuing synchronously, SYS2–245
Identifier names
  translating to identifier, SYS1–32

Identifiers
adding record to rights list, SYS2–101
finding, SYS1–378
modifying in rights database, SYS2–172
removing from rights database, SYS2–251
revoking from process, SYS2–260
translating value to identifier name, SYS2–110
$IDTOASC system service, SYS2–110
IEEE floating-point control register
setting, SYS2–113
$IEEE_SET_FP_CONTROL system service
on Alpha systems only, SYS2–113
Image exit, SYS1–344
Image rundown
forcing, SYS1–386
Initializing a volume
from within a program, SYS2–118
$INIT_SYS_ALIGN_FAULT_REPORT system service
on Alpha systems only, SYS2–115
$INIT_VOL system service, SYS2–118
Intrusion records
deleting, SYS1–250
Intrusions
returning information about, SYS2–351
scanning for, SYS2–268
$IOSETUP system service, SYS2–136
$IO_CLEANUP system service, SYS2–131
description, SYS2–131
$IO_PERFORM system service, SYS2–132
description, SYS2–133
$IO_SETUP system service
description, SYS2–137

J

Job controllers
asynchronous, SYS2–359
synchronous, SYS2–417
Jobs
getting information asynchronously, SYS1–427, SYS2–3
getting information synchronously, SYS1–448, SYS2–46

K

Kernel mode
changing to, SYS1–114, SYS1–116

L

$LCKPAG system service, SYS2–139
$LCKPAG_64 system service, SYS2–142
description, SYS2–143
$LKWSET system service, SYS2–145
$LKWSET_64 system service, SYS2–148
description, SYS2–149
Lock database
in a VMScluster, SYS1–458
Lock requests
dequeuing, SYS1–270
queuing asynchronously, SYS1–328
queuing synchronously, SYS1–340
Locks
getting information asynchronously, SYS1–449
getting information synchronously, SYS1–461
Logical names
creating, SYS1–149
deleting, SYS1–258
getting information about, SYS2–451
translating, SYS2–451
Logical name tables
creating, SYS1–155
deleting, SYS1–258

M

Magnetic tapes
initializing from within a program, SYS2–118
Mailboxes
assigning channel to, SYS1–161
creating, SYS1–161
deleting permanent, SYS1–164, SYS1–261
deleting temporary, SYS1–164
Mapping disk file sections, SYS1–192
Memory
locking page into, SYS2–139
unlocking page from, SYS2–458
Messages
converting security message from binary to ASCII, SYS1–402
filtering sensitive information, SYS1–402
formatting and outputting, SYS2–231
obtaining text of, SYS1–462
sending to error logger, SYS2–358
sending to one or more terminals, SYS1–68, SYS1–76
sending to operator, SYS2–418
writing to terminal, SYS1–68, SYS1–76
Message symbols, SYS2–236
$MGBLSC system service, SYS2–151
$MGBLSC_64 system service, SYS2–157
description, SYS2–161
$MGBLSC_GPFN_64 system service, SYS2–163
description, SYS2–166
$MOD_HOLDER system service, SYS2–169
$MOD_IDENT system service, SYS2–172
$MOUNT system service, SYS2–176
$MTACCESS system service, SYS2–191

N

Notification ASTs
  testing functionality of, SYS2–456
$NUMTIM system service, SYS2–194
$NUMUTC system service, SYS2–196

O

Obsolete system services, A–1
Opaque names
  converting to string, SYS1–303
Operators
  sending messages to, SYS2–418

P

Page frame section
  creating, SYS1–135
Page protection
  setting, SYS2–311
Pages
  locking into memory, SYS2–139
  locking into working set, SYS2–145
  removing from working set, SYS2–227
  setting protection, SYS2–308
  unlocking from memory, SYS2–458
  unlocking from working set, SYS2–463
$PARSE_ACL system service, SYS2–198
Passwords
  returning hash value, SYS2–105
$PERFORMW system service, SYS2–135
$PERM_DIS_ALIGN_FAULT_REPORT system service
  on Alpha systems only, SYS2–201
$PERM_REPORT_ALIGN_FAULT system service
  on Alpha systems only, SYS2–202
PID numbers
  using with $GETJPI to return information about a process, SYS1–427
Power recovery
  setting AST for, SYS2–301
Priority setting, SYS2–303
Private disk file section
  create and map, SYS1–204
Private page frame
  create and map, SYS1–232
Privileges
  checking, SYS1–93
  setting for process, SYS2–314
Processes
  affecting scheduling of, SYS2–276
  creating, SYS1–168
  deleting, SYS1–263
  getting information asynchronously, SYS1–427
  getting information synchronously, SYS1–448
  hibernating, SYS2–108
Processes (cont'd)
  locating a subset of, SYS2–214
  rescheduling, SYS2–253
  resuming after suspension, SYS2–258
  scanning, SYS2–214
  scheduling wakeup for, SYS2–273
  setting default protection for, SYS2–287
  setting name of, SYS2–307
  setting priority of, SYS2–303
  setting privileges, SYS2–314
  setting stack limits, SYS2–323
  setting swap mode for, SYS2–325
  suspending, SYS2–444
  waiting for entire set of event flags, SYS2–490
  waiting for event flag to be set, SYS2–487
  waiting for one of set of event flags, SYS2–492
  waking, SYS2–488
  writing messages to, SYS2–231
Process identification numbers
  See PID numbers
Process names
  setting, SYS2–307
  specifying processes by, SYS2–220
  specifying processes with node name, SYS2–219
Process scan, SYS2–214
Process scheduling
  affecting, SYS2–276
Process user capability set
  modifying, SYS2–209
$PROCESS_AFFINITY system service, SYS2–204
$PROCESS_CAPABILITIES system service, SYS2–209
$PROCESS_SCAN system service, SYS2–214
Program regions
  adding page to, SYS1–345
  deleting page from, SYS1–265
Protection
  of queues, SYS2–409
  setting for page, SYS2–308
Proxies
  adding, SYS1–14
  deleting, SYS1–252
  displaying, SYS1–286
  modifying, SYS1–14, SYS1–252
  verifying, SYS2–482
$PURGE_WS system service, SYS2–229
  description, SYS2–229
$PURGWS system service, SYS2–227
  See also $ADJWSL
$PUTMSG system service, SYS2–231

Q

$QIO system service, SYS2–239
$QIOW system service, SYS2–245
Queues
  creating and managing asynchronously, SYS2–359
  creating and managing synchronously, SYS2–417
  getting information asynchronously, SYS2–3
  getting information synchronously, SYS2–46
  protection, SYS2–409
  types of, SYS2–406

R

$READEF system service, SYS2–246
Region
  creating a virtual, SYS1–141
Regions
  deleting, SYS1–255
$RELEASE_VP system service
  on VAX systems only, SYS2–248
$REM_HOLDER system service, SYS2–249
$REM_IDENT system service, SYS2–251
$RESCHED system service, SYS2–253
Resource wait mode
  setting, SYS2–319
$RESTORE_VP_EXCEPTION system service
  on VAX systems only, SYS2–254
$RESTORE_VP_STATE system service
  on VAX systems only, SYS2–256
$RESUME system service, SYS2–258
$REVOKID system service, SYS2–260
Rights database context
  terminating, SYS1–384
Rights databases
  creating, SYS1–139
$RMSRUNDWN system service, SYS2–264

S

$SAVE_VP_EXCEPTION system service
  on VAX systems only, SYS2–266
Scanning
  for devices, SYS1–275
  intrusion database, SYS2–268
  processes, SYS2–214
$SCAN_INTRUSION system service, SYS2–268
$SCHDWK system service, SYS2–273
$SCHED system service, SYS2–276
Section files
  updating asynchronously, SYS2–470
  updating synchronously, SYS2–480
Sections
  creating, SYS1–192
  deleting global, SYS1–279
Sections (cont'd)
  mapping, SYS1–192
  writing modifications to disk, SYS2–470, SYS2–480
Security
  auditing events, SYS1–43, SYS1–61
  checking privileges, SYS1–93, SYS1–98
  converting message from binary to ASCII, SYS1–402
  filtering sensitive message information, SYS1–402
  getting erase patterns, SYS1–341
  hashing passwords, SYS2–105
  modifying characteristics of an object, SYS2–344
  retrieving information about objects, SYS2–89
Security characteristics
  modifying for an object, SYS2–344
  retrieving for an object, SYS2–89
Sending a message to one or more terminals, SYS1–68, SYS1–76
$SETAST system service, SYS2–281
$SETCLUEVT system service
  on Alpha systems only, SYS2–282
$SETDDIR system service, SYS2–285
$SETDFPROT system service, SYS2–287
$SETEF system service, SYS2–289
$SETEXV system service, SYS2–290
$SETIME system service, SYS2–292
$SETIMR system service, SYS2–294
$SETPRA system service, SYS2–301
$SETPRI system service, SYS2–303
$SETPRN system service, SYS2–307
$SETPRT system service, SYS2–308
$SETPRT_64 system service, SYS2–311
  description, SYS2–312
$SETPRV system service, SYS2–314
$SETRWM system service, SYS2–319
$SETSHLV system service, SYS2–321
$SETSTK system service, SYS2–323
$SETSWM system service, SYS2–325
Setting the resource wait mode, SYS2–319
$SETUAI system service, SYS2–327
$SET_IMPLICIT_AFFINITY, SYS2–297
$SET_RESOURCE_DOMAIN system service, SYS2–339
$SET_SECURITY system service, SYS2–344
Shelving
  See also Automatic unshelving
$SHOW_INTRUSION system service, SYS2–351
$SIGNAL_ARRAY system service, SYS2–356
Simple names
  converting to opaque, SYS1–305
$SNDERR system service, SYS2–358
$SNDJBC system service, SYS2–359

$SNDJBCW system service, SYS2–417
$SNDOPR system service, SYS2–418
Stack limit
  changing size of, SYS2–323
Stack pointer
  adjusting, SYS1–18
$START_ALIGN_FAULT_REPORT system service
  on Alpah systems only, SYS2–432
$START_TRANS system service, SYS2–435
$START_TRANSW system service, SYS2–439
$STOP_ALIGN_FAULT_REPORT system service
  on Alpha systems only, SYS2–440
$STOP_SYS_ALIGN_FAULT_REPORT system service
  on Alpha systems only, SYS2–441
Strings
  formatting output, SYS1–351
  searching for file specification in, SYS1–372
Subprocesses
  creating, SYS1–180
$SUBSYSTEM system service, SYS2–442
$SUSPND system service, SYS2–444
$SYNCH system service, SYS2–447
SYS$NUMUTC system service, SYS2–196
SYS$SYSTEM:LOGINOUT.EXE file
  using as image to create new processes, SYS1–168, SYS1–180
System alignment fault reporting
  disabling for user image, SYS2–441
Systems
  getting information asynchronously, SYS2–51
  getting information synchronously, SYS2–70
System services, SYS2–131, SYS2–132, SYS2–135, SYS2–136
  Abort Transaction, SYS1–3
  Abort Transaction and Wait, SYS1–7
  Add Holder Record to Rights Database, SYS1–8
  Add Identifier to Rights Database, SYS1–11
  Add Proxy, SYS1–14
  Adjust Outer Mode Stack Pointer, SYS1–18
  Adjust Working Set Limit, SYS1–20
  Affect Process Scheduling, SYS2–276
  Allocate Device, SYS1–22
  Assign I/O Channel, SYS1–38
  Associate Common Event Flag Cluster, SYS1–25
  Audit Event, SYS1–43
  Audit Event and Wait, SYS1–61
  Breakthrough, SYS1–68
  Breakthrough and Wait, SYS1–76
  Cancel Exit Handler, SYS1–79
  Cancel I/O on Channel, SYS1–77
  Cancel Timer, SYS1–80
  Cancel Wakeup, SYS1–82
  Change to Executive Mode, SYS1–110
    with quadword argument list, SYS1–112
  Change to Kernel Mode, SYS1–114, SYS1–116
System services (cont'd)
  Check Access, SYS1–84
  Check Access Protection, SYS1–99
  Check Floating Point (Alpha only), SYS1–92
  checking completion status of, SYS2–447
  Check Privilege, SYS1–93
  Check Privilege and Wait, SYS1–98
  Clear Cluster Event (Alpha only), SYS1–107
  Clear Event Flag, SYS1–109
  Convert ASCII String to Binary Time, SYS1–62
  Convert ASCII String to UTC Binary Time, SYS1–65
  Convert Binary Time to ASCII String, SYS1–29
  Convert Binary Time to Numeric Time, SYS2–194
  Convert UTC Time to Numeric Components, SYS2–196
  Convert UTC to ASCII, SYS1–35
  Create and Map a Global Disk File Section, SYS1–210
  Create and Map Global Page File Section, SYS1–218
  Create and Map Global Page Frame Section, SYS1–225
  Create and Map Private Disk File Section, SYS1–204
  Create and Map Private Page Frame Section, SYS1–232
  Create and Map Section, SYS1–192
  Create Global Page Frame Section, SYS1–135
  Create Logical Name, SYS1–149
  Create Logical Name Table, SYS1–155
  Create Mailbox and Assign Channel, SYS1–161
  Create Permanent Global Disk File Section, SYS1–126
  Create Permanent Global Page File, SYS1–131
  Create Process, SYS1–168
  Create Rights Database, SYS1–139
  Create User Profile, SYS1–145
  Create Virtual Address Space, SYS1–185, SYS1–188
  Create Virtual Region, SYS1–141
  Deallocate Device, SYS1–238
  Deassign I/O Channel, SYS1–240
  Declare AST, SYS1–242
  Declare Change Mode or Compatibility Mode Handler, SYS1–244
  Declare Exit Handler, SYS1–247
  Delete a Virtual Region, SYS1–255
  Delete Buffer Object, SYS1–249
  Delete Common Event Flag Cluster, SYS1–292
  Delete Global Section, SYS1–279
  Delete Intrusion Records, SYS1–250
  Delete Logical Name, SYS1–258
  Delete Mailbox, SYS1–261
  Delete or Modify Proxy, SYS1–252

System services (cont'd)
Delete Process, SYS1–263
Delete Virtual Address Space, SYS1–265, SYS1–267
Dequeue Lock Request, SYS1–270
Disable Alignment Fault Reporting (Alpha only), SYS2–201
Disassociate Common Event Flag Cluster, SYS1–236
Dismount Volume, SYS1–282
Display Proxy Information, SYS1–286
Distributed Name Service (DNS) Clerk (VAX only), SYS1–294, SYS1–321
End Transaction, SYS1–322
End Transaction and Wait, SYS1–327
Enqueue Lock Request, SYS1–328
Enqueue Lock Request and Wait, SYS1–340
Exit, SYS1–344
Expand Program/Control Region, SYS1–345
Expand Virtual Address Space, SYS1–348
Find Holder of Identifier, SYS1–381
Find Identifiers Held by User, SYS1–378
Force Exit, SYS1–386
Format Access Control List Entry, SYS1–389
Format Security Audit Event Message, SYS1–402
Formatted ASCII Output Services, SYS1–351
Formatted ASCI Output with List Parameter for 64-Bit Memory, SYS1–371
Get Alignment Fault Data (Alpha only), SYS2–85
Get Arithmetic Exception Information (Alpha only), SYS2–87
Get Device/Volume Information, SYS1–406
Get Device/Volume Information and Wait, SYS1–426
Get Information About a Specified Virtual Region, SYS2–47
Get Job/Process Information, SYS1–427
Get Job/Process Information and Wait, SYS1–448
Get Lock Information, SYS1–449
Get Lock Information and Wait, SYS1–461
Get Message, SYS1–462
Get Queue Information, SYS2–3
Get Queue Information and Wait, SYS2–46
Get Security Characteristics, SYS2–89
Get Security Erase Pattern, SYS1–341
Get System Alignment Fault Data (Alpha only), SYS2–97
Get Systemwide Information, SYS2–51
Get Systemwide Information and Wait, SYS2–70
Get Time, SYS2–71
Get User Authorization Information, SYS2–72
Get UTC Time, SYS2–84
Grant Identifier to Process, SYS2–101
Hash Password, SYS2–105

System services (cont'd)
Hibernate, SYS2–108
Initialize System Alignment Fault Reporting (Alpha only), SYS2–115
Initialize Volume, SYS2–118
Lock Pages in Memory, SYS2–139, SYS2–142
Lock Pages in Working Set, SYS2–145, SYS2–148
Magnetic Tape Accessibility, SYS2–191
Map Global Disk or Page File Section, SYS2–157
Map Global Page Frame Section, SYS2–163
Map Global Section, SYS2–151
Modify CPU User Capabilities, SYS1–118
Modify Holder Record in Rights Database, SYS2–169
Modify Identifier in Rights Database, SYS2–172
Modify Process Affinity, SYS2–204
Modify Process Implicit Affinity, SYS2–297
Modify Process User Capabilities, SYS2–209
Mount Volume, SYS2–176
obsolete, A–1
Parse Access Control List Entry, SYS2–198
Process Scan, SYS2–214
Purge Working Set, SYS2–227, SYS2–229
Put Message, SYS2–231
Queue I/O Request, SYS2–239
Queue I/O Request and Wait, SYS2–245
Read Event Flags, SYS2–246
Release Vector Processor (VAX only), SYS2–248
Remove Holder Record from Rights Database, SYS2–249
Remove Identifier from Rights Database, SYS2–251
Report Alignment Fault (Alpha only), SYS2–202
Reschedule Process, SYS2–253
Restore Vector Processor Exception State (VAX only), SYS2–254
Restore Vector State (VAX only), SYS2–256
Resume Process, SYS2–258
Revoke Identifier from Process, SYS2–260
RMS Rundown, SYS2–264
Save Vector Processor Exception State (VAX only), SYS2–266
Scan for Devices, SYS1–275
Scan Intrusion Database, SYS2–268
Scan String for File Specification, SYS1–372
Schedule Wakeup, SYS2–273
Send Message to Error Logger, SYS2–358
Send Message to Operator, SYS2–418
Send to Job Controller, SYS2–359
Send to Job Controller and Wait, SYS2–417
Set AST Enable, SYS2–281
Set Automatic Unshelving, SYS2–321
Set Cluster Event (Alpha only), SYS2–282
Set Default Directory, SYS2–285

System services (cont'd)
Set Default File Protection, SYS2–287
Set Event Flag, SYS2–289
Set Exception Vector, SYS2–290
Set IEEE Floating-Point Control Register (Alpha only), SYS2–113
Set Power Recovery AST, SYS2–301
Set Priority, SYS2–303
Set Privileges, SYS2–314
Set Process Name, SYS2–307
Set Process Swap Mode, SYS2–325
Set Protection on Pages, SYS2–308, SYS2–311
Set Resource Domain, SYS2–339
Set Resource Wait Mode, SYS2–319
Set Security, SYS2–344
Set Stack Limits, SYS2–323
Set System Time, SYS2–292
Set Timer, SYS2–294
Set User Authorization Information, SYS2–327
Show Intrusion Information, SYS2–351
Signal Array, SYS2–356
Start Alignment Fault Reporting (Alpha only), SYS2–432
Start Transaction, SYS2–435
Start Transaction and Wait, SYS2–439
Stop Alignment Fault Reporting (Alpha only), SYS2–440
Stop System Alignment Fault Reporting (Alpha only), SYS2–441
Subsystem, SYS2–442
Suspend Process, SYS2–444
Synchronize, SYS2–447
Terminate Rights Database Context, SYS1–384
Test Cluster Event (Alpha only), SYS2–456
Time Converter, SYS2–449
Translate Identifier Name to Identifier, SYS1–32
Translate Identifier to Identifier Name, SYS2–110
Translate Logical Name, SYS2–451
Unlock Pages from Memory, SYS2–458, SYS2–460
Unlock Pages from Working Set, SYS2–463
Unlock Pages in Working Set, SYS2–465
Unwind Call Stack, SYS2–468
Unwind Call Stack (Alpha only), SYS2–99
Update Global Section File on Disk, SYS2–475
Update Global Section File on Disk and Wait, SYS2–481
Update Section File on Disk, SYS2–470
Update Section File on Disk and Wait, SYS2–480
Verify Proxy, SYS2–482
Wait for Logical AND of Event Flags, SYS2–490
Wait for Logical OR of Event Flags, SYS2–492
Wait for Single Event Flag, SYS2–487
Wake Process from Hibernation, SYS2–488
System Services
Create Buffer Object, SYS1–122
System time
See also Time
converting 64-bit time to UTC time, SYS2–449
setting, SYS2–292

T

Tapes
initializing from within a program, SYS2–118
Termination messages
format of, SYS1–177
$TIMCON system service, SYS2–449
Time
converting 64-bit system format to UTC, SYS2–449
converting binary to ASCII string, SYS1–29
converting binary to numeric, SYS2–194
converting UTC to 64-bit system format, SYS2–449
converting UTC to ASCII, SYS1–35
converting UTC to numeric components, SYS2–196
getting current system, SYS2–71
setting system, SYS2–292
Timer requests
canceling, SYS1–80
Timers
setting, SYS2–294
TQELM (timer queue entry limit)
See TQELM process limit
TQELM process limit
effect of canceling timer request, SYS1–80
Transactions
aborting asynchronously, SYS1–3
aborting synchronously, SYS1–7
default, SYS2–437
ending asynchronously, SYS1–322
ending synchronously, SYS1–327
starting asynchronously, SYS2–435
starting synchronously, SYS2–439
Translating identifier name to identifier, SYS1–32
$TRNLNM system service, SYS2–451
$TSTCLUEVT system service
on Alpha systems only, SYS2–456

U

UAFs (user authorization files)
getting information about, SYS2–72
modifying, SYS2–327
$ULKPAG system service, SYS2–458
$ULKPAG_64 system service, SYS2–460
description, SYS2–461

$ULWSET system service, SYS2–463
$ULWSET_64 system service, SYS2–465
  description, SYS2–466
Unlocking pages from memory, SYS2–460
Unlocking pages in the working set, SYS2–465
$UNWIND system service, SYS2–468
$UPDSEC system service, SYS2–470
$UPDSECW system service, SYS2–480
$UPDSEC_64 system service, SYS2–475
  description, SYS2–478
$UPDSEC_64W system service, SYS2–481
User profiles
  creating, SYS1–145
UTC Coordinated Universal Time
  converting format to ASCII, SYS1–35
UTC format
  converting to ASCII, SYS1–35
  converting to numeric components, SYS2–196
  getting, SYS2–84

V

Vector processors
  releasing, SYS2–248
  restoring the exception state of, SYS2–254
  saving the exception state of, SYS2–266
Vector state
  restoring, SYS2–256
$VERIFY_PROXY system service, SYS2–482
Virtual address space
  adding page to, SYS1–185, SYS1–345
  creating, SYS1–185
  deleting page from, SYS1–265
Virtual Address Space
  expanding, SYS1–348
Virtual address space, deleting, SYS1–267
Virtual I/O
  canceling requests for, SYS1–77
Virtual region
  getting information about, SYS2–47
Volumes
  dismounting, SYS1–282
  getting information asynchronously, SYS1–406
  getting information synchronously, SYS1–426
  initializing from within a program, SYS2–118
  mounting, SYS2–176

W

$WAITFR system service, SYS2–487
$WAKE system service, SYS2–488
  See also $HIBER
Wakeup requests
  canceling, SYS1–82
$WFLAND system service, SYS2–490
$WFLOR system service, SYS2–492
Wildcard operations, SYS1–427
Wildcard searches
  obtaining information about processes, SYS2–214
Working set
  purging, SYS2–229
Working sets
  adjusting limit, SYS1–20
  locking page into, SYS2–145
  purging, SYS2–227
  unlocking page from, SYS2–463

NOTES