VMS Software

VSI OpenVMS

System Services Reference Manual: A–GETUAI

Document Number: DO-DSSRFA-01A

Publication Date: May 2024

Operating System and Version: VSI OpenVMS IA-64 Version 8.4-1H1 or higher
VSI OpenVMS Alpha Version 8.4-2L1 or higher
VSI OpenVMS x86-64 Version 9.2-1 or higher

VMS Software, Inc. (VSI)
Boston, Massachusetts, USA

System Services Reference Manual: A–GETUAI

Legal Notice

Preface

1. About VSI

VMS Software, Inc. (VSI) is an independent software company licensed by Hewlett Packard Enterprise to develop and support the OpenVMS operating system.

2. Intended Audience

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

3. System Services Support for OpenVMS Alpha 64-bit Addressing

As of Version 7.0, the OpenVMS Alpha operating system provides support for 64-bit virtual memory addresses. This support 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 *VSI OpenVMS Programming Concepts Manual*.

The following section briefly describes how 64-bit addressing support affects OpenVMS system services. For complete information about OpenVMS Alpha 64-bit addressing features, refer to the *VSI OpenVMS Programming Concepts Manual*.

64-Bit System Services Terminology

32-Bit System Service

A 32-bit system service 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 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 defined to accept all address arguments as 64-bit addresses (not necessarily 32-bit sign-extended values). A 64-bit system service also 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 suffix 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

The 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 receive sign-extension checking. Any argument passed to these services that is not properly sign-extended causes the error status SS$_ARG_GTR_32_BITS to be returned.

4. Related Documents

The *VSI OpenVMS Programming Concepts 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.

Application developers using XA-compliant or other resource managers should refer to the *VSI OpenVMS Programming Concepts Manual*.

The following documents might also be useful:

- *VSI OpenVMS Programming Concepts Manual*
- *VSI OpenVMS Guide to OpenVMS File Applications*
- *VSI OpenVMS Guide to System Security*
- *DECnet-Plus for OpenVMS Introduction and User's Guide*
- *VSI OpenVMS Record Management Services Reference Manual*
- *VSI OpenVMS I/O User's Reference Manual*
- *VSI OpenVMS Alpha Guide to Upgrading Privileged-Code Applications*

5. VSI Encourages Your Comments

You may send comments or suggestions regarding this manual or any VSI document by sending electronic mail to the following Internet address: <docinfo@vmssoftware.com>. Users who have VSI OpenVMS support contracts through VSI can contact <support@vmssoftware.com> for help with this product.

6. OpenVMS Documentation

The full VSI OpenVMS documentation set can be found on the VMS Software Documentation webpage at https://docs.vmssoftware.com.

7. Typographical Conventions

The following conventions are used in this manual:

Convention	Meaning
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.
PF1 *x*	A sequence such as PF1 *x* indicates that you must first press and release the key labeled PF1 and then press and release another key (*x*) or a pointing device button.
...	A horizontal ellipsis in examples indicates 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.
⋮	A vertical ellipsis indicates 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 you must enclose choices in parentheses if you specify more than one.
[ ]	In the *VSI OpenVMS System Services Reference Manual*, brackets generally indicate default arguments. If an argument is optional, it is specified as such in the argument text.
|	In command format descriptions, vertical bars separate choices within brackets or braces. Within brackets, the choices are optional; within braces, at least one choice is required. Do not type the vertical bars on the command line.
{ }	In command format descriptions, braces indicate required choices; you must choose at least one of the items listed. Do not type the braces on the command line.
bold type	Bold type represents the name of an argument, an attribute, or a reason. Bold type also represents the introduction of a new term.
italic type	Italic type indicates important information, complete titles of manuals, or variables. Variables include information that varies in system output (Internal error *number*), in command lines (/PRODUCER=*name*), and in command parameters in text (where *dd* represents the predefined code for the device type).
UPPERCASE TYPE	Uppercase type indicates a command, the name of a routine, the name of a file, or the abbreviation for a system privilege.
`Example`	This typeface indicates code examples, command examples, and interactive screen displays. In text, this type also identifies URLs, UNIX commands and pathnames, PC-based commands and folders, and certain elements of the C programming language.
–	A hyphen at the end of a command format description, command line, or code line indicates that the command or statement continues on the following line.

Convention	Meaning
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 indicate not only whether the service completed successfully, but can also provide other information. While the usual condition value indicating success is SS$_NORMAL, other values are also defined. For example, the condition value SS$_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, but it also provides additional information.

Warning returns and some error returns indicate that the service might 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.

$ABORT_TRANS

Abort Transaction — Ends a transaction by aborting it.

Format

```
SYS$ABORT_TRANS
   [efn] ,[flags] ,iosb [,[astadr] ,[astprm] ,[tid] ,[reason]
   ,[bid]]
```

C Prototype

```
int sys$abort_trans
   (unsigned int efn, unsigned int flags, struct _iosb *iosb,...);
```

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 used.

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. All undefined bits must be 0. If this argument is omitted, no flags are set.

Table 1. $ABORT_TRANS Option Flags

Flag	Description
DDTM$M_NOWAIT	Set this flag to indicate that the service should return to the caller without waiting for final cleanup. Note that $ABORT_TRANSW with the DDTM$M_NOWAIT flag set is not equivalent to $ABORT_TRANS. $ABORT_TRANS returns when the operation has been queued. The former does not return until the operation has been initiated. The latter returns as soon as the operation has been queued. The full range of status values may be returned from a nowait call.
DDTM$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 following information is returned:

- The completion status of the service, returned as a condition value. See the section called “Condition Values Returned”.
- An abort reason code that gives one reason why the transaction aborted, if the completion status of the service is SS$_NORMAL.

  Note that, if there are multiple reasons why the transaction aborted, the abort reason code returned in the I/O status block might not be the same as the abort reason code passed in the *reason*

argument. The DECdtm transaction manager returns one of the reasons in the I/O status block. It may return different reasons to different branches of the transaction.

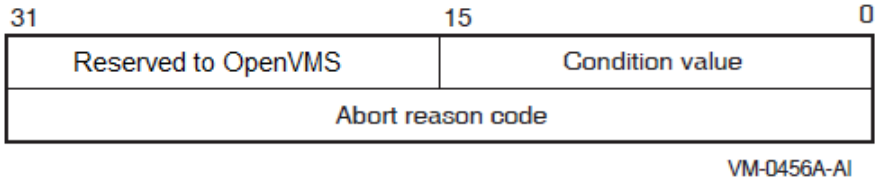
For example, if the call to $ABORT_TRANS gives DDTM$_ABORTED as the reason and the transaction timeout expires at about the same time as the call to $ABORT_TRANS, then either the DDTM$_TIMEOUT or DDTM$_ABORTED reason code can be returned in the I/O status block.

The $DDTMMSGDEF macro defines symbolic names for abort reason codes. Those currently defined are shown in Table 2.

Table 2. Abort Reason Codes

Symbolic Name	Description
DDTM$_ABORTED	The application aborted the transaction without giving a reason.
DDTM$_COMM_FAIL	A communications link failed.
DDTM$_INTEGRITY	A resource manager integrity constraint check failed.
DDTM$_LOG_FAIL	A write operation to the transaction log failed.
DDTM$_ORPHAN_BRANCH	An unauthorized branch caused failure.
DDTM$_PART_SERIAL	A resource manager serialization check failed.
DDTM$_PART_TIMEOUT	The timeout specified by a resource manager expired.
DDTM$_SEG_FAIL	A process or image terminated.
DDTM$_SERIALIZATION	A DECdtm transaction manager serialization check failed.
DDTM$_SYNC_FAIL	The transaction was not globally synchronized; an authorized branch was not added to the transaction.
DDTM$_TIMEOUT	The timeout specified on $START_TRANS expired.
DDTM$_UNKNOWN	The reason is unknown.
DDTM$_VETOED	A resource manager was unable to commit the transaction.

The following diagram shows the structure of the I/O status block:



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, if SS$_NORMAL is returned in R0. The *astadr* argument is the address of the entry mask of this routine. The 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:	trans_id
type:	octaword (unsigned)
access:	read only
mechanism:	by reference

Identifier of the transaction to be aborted.

If this argument is omitted, $ABORT_TRANS aborts the default transaction of the calling process.

reason

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

Code that gives the reason why the application is aborting the transaction. The $DDTMMSGDEF macro defines symbolic names for abort reason codes. The codes currently defined are listed in Table 2. The default value for this argument is DDTM$_ABORTED.

bid

OpenVMS usage:	branch_id
type:	octaword (unsigned)
access:	read only
mechanism:	by reference

The identifier (BID) of the branch that is aborting the transaction.

The default value of this argument is zero, which is the BID of the branch that started the transaction.

Description

The Abort Transaction service ends a transaction by aborting it.

The $ABORT_TRANS system service:

- Initiates abort processing for the specified transaction, if it has not already been initiated.

If abort processing has not already been initiated, the DECdtm transaction manager instructs the resource managers to abort (roll back) the transaction operations so that none of those operations ever take effect. It delivers an abort event to each RM participant in the transaction that is associated with an RMI that requested abort events.

- Removes the specified branch from the specified transaction in this process.

Preconditions for the successful completion of $ABORT_TRANS include:

- If the BID is zero, the calling process must have started the transaction.
- If the BID is nonzero, the calling process must contain the specified branch of the specified transaction.
- If the BID is nonzero, the *tid* argument must not be omitted. If you explicitly pass the BID, you must also explicitly pass the TID.

$ABORT_TRANS may fail for various reasons, including:

- The preconditions were not met.
- There has already been a call to $ABORT_TRANS, $END_TRANS, or $END_BRANCH for the specified branch.

Postconditions on successful completion of $ABORT_TRANS are listed in Table 3.

Table 3. Postconditions When $ABORT_TRANS Completes Successfully

Postcondition	Meaning
The transaction is ended.	If DDTM$M_NOWAIT is clear: • The TID of the transaction is invalid; calls to any DECdtm system services except $GETDTI and $SETDTI that pass the TID will fail, and calls to resource managers that pass the TID will fail. • The transaction no longer has any application or RM participants on the local node. • All communications about the transaction between the local DECdtm transaction manager and other DECdtm transaction managers are finished (including the final "cleanup" acknowledgment).
The outcome of the transaction is abort.	None of the operations of the transaction will ever take effect.
DECdtm quotas are returned.	If DDTM$M_NOWAIT is clear, all quotas allocated for the transaction by calls on the local node to DECdtm services are now returned.
The transaction is not the default transaction of the calling process.	If DDTM$M_NOWAIT is clear, then, if the transaction was the default transaction of the calling process, it is now no longer the default.

$ABORT_TRANS will not complete successfully (that is, the event flag will not be set, the AST routine will not be called, and the I/O status block will not be filled in) until all branches on the local node have

been removed from the transaction. Thus this call to $ABORT_TRANS cannot complete successfully until every authorized and synchronized branch on the local node has initiated a call to $END_TRANS, $END_BRANCH, or $ABORT_TRANS.

$ABORT_TRANS must deliver notification ASTs to resource managers participating in the transaction. Therefore it will not complete successfully while the calling process is either:

- In an access mode that is more privileged than the DECdtm calls made by any resource manager participating in the transaction. RMS journalling calls DECdtm in executive mode. Oracle Rdb and Oracle CODASYL DBMS call DECdtm in user mode.
- At AST level in the same access mode as the least privileged DECdtm calls made by any resource manager participating in the transaction.

For example, if Oracle Rdb is a participant in the transaction, $ABORT_TRANS will not complete successfully while the calling process is in supervisor, executive, or kernel mode, or while the calling process is at AST level.

Note that successful completion of $ABORT_TRANS is not indefinitely postponed by network failure.

Required Access or Privileges

None.

Required Quotas

ASTLM

Related Services

$ABORT_TRANSW, $ACK_EVENT, $ADD_BRANCH, $ADD_BRANCHW, $CREATE_UID, $DECLARE_RM, $DECLARE_RMW, $END_BRANCH, $END_BRANCHW, $END_TRANS, $END_TRANSW, $FORGET_RM, $FORGET_RMW, $GETDTI, $GETDTIW, $GET_DEFAULT_TRANS, $JOIN_RM, $JOIN_RMW, $SETDTI, $SETDTIW, $SET_DEFAULT_TRANS, $SET_DEFAULT_TRANSW, $START_BRANCH, $START_BRANCHW, $START_TRANS, $START_TRANSW, $TRANS_EVENT, $TRANS_EVENTW

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$_BADPARAM

The options flags were invalid or the *tid* argument was omitted and the *bid* argument was not zero.

SS$_BADREASON

The abort reason code was invalid.

SS$_CURTIDCHANGE

The *tid* argument was omitted 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$_ILLEFC

The event flag number was invalid.

SS$_INSFARGS

A required argument was missing.

SS$_INSFMEM

There was insufficient system dynamic memory for the operation.

SS$_NOCURTID

An attempt was made to abort the default transaction (the *tid* argument was omitted), but the calling process did not have a default transaction.

SS$_NOLOG

The local node did not have a transaction log.

SS$_NOSUCHBID

The calling process did not contain the branch identified by the BID passed in the *bid* argument (possibly because there has already been a call to $ABORT_TRANS, $END_TRANS, or $END_BRANCH for that branch).

This error is returned only if the *bid* argument is not zero.

SS$_NOSUCHTID

A transaction with the specified transaction identifier does not exist.

SS$_NOTORIGIN

A *bid* of zero was specified and the calling process did not start the transaction.

SS$_TPDISABLED

The TP_SERVER process was not running on the local node.

SS$_WRONGSTATE

Commit processing for the transaction had already started. This can occur if **bid** is zero or the specified branch was unsynchronized.

$ABORT_TRANSW

Abort Transaction and Wait — Ends a transaction by aborting it. $ABORT_TRANSW always waits for the request to complete before returning to the caller. Other than this, it is identical to $ABORT_TRANS. Do not call $ABORT_TRANSW from AST level, or from an access mode that is more privileged than the DECdtm calls made by any resource manager participant in the transaction. If you do, the $ABORT_TRANSW service will wait indefinitely.

Format

```
SYS$ABORT_TRANSW
   [efn] ,[flags] ,iosb [,[astadr] ,[astprm] ,[tid] ,[reason] ,[bid]]
```

C Prototype

```
int sys$ABORT_TRANSW
   (unsigned int efn, unsigned int flags, struct _iosb *iosb,...);
```

$ACK_EVENT

Acknowledge Event — Acknowledges an event reported to a Resource Manager (RM) participant or Resource Manager instance (RMI).

Format

```
SYS$ACK_EVENT
   [flags] ,report_id ,report_reply [,[reason] ,[beftime] ,[afttime]
   ,[part_name] ,[rm_context], [timout]]
```

C Prototype

```
int sys$ack_event
   (unsigned int flags, unsigned int report_id, int report_reply,...);
```

Arguments

flags

OpenVMS usage:	mask_longword
type:	longword (unsigned)

access:	read only
mechanism:	by value

Reserved to OpenVMS. This argument must be zero.

report_id

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

The identifier of the event report being acknowledged by this call to $ACK_EVENT.

report_reply

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

Acknowledgment code appropriate to the event being acknowledged by this call to $ACK_EVENT. The following tables give the valid acknowledgment codes for the various events. The title of each table gives the event, and in brackets, its event code. The event code is passed in the event report block (see $DECLARE_RM).

Acknowledgment of prepare or one-phase commit events gives a vote on the outcome of the transaction —either to commit or to abort. The tables for these events have a column labeled "Vote". A "yes" vote means that the RM participant wants to commit the transaction, while a "no" vote means that the RM participant cannot commit. The transaction will be committed only if all participants vote "yes".

Table 4. Replies to an Abort Event Report (DDTM$K_ABORT)

report_reply	Description
SS$_FORGET	RM participant guarantees that the effects of its transaction operations will never be detected by any transaction that commits. **Side effects:** On successful completion of the call to $ACK_EVENT, the RM participant is removed from the transaction, and the ASTLM quota consumed by the call to $JOIN_RM or $ACK_EVENT that added it to the transaction is returned. DECdtm also releases any application threads that are waiting for the transaction to end (necessary but not sufficient condition). Any call to $END_TRANS, $END_BRANCH, or $ABORT_TRANS on a node for a transaction whose outcome is abort is not allowed to complete until after all abort event reports delivered to RM participants on that node have been acknowledged.

Table 5. Replies to a Commit Event Report (DDTM$K_COMMIT)

report_reply	Description
SS$_FORGET	Allows the DECdtm transaction manager to forget the RM participant. The RM participant must not give this reply until it has either: • Completed the commit processing for its transaction operations. • Safely stored enough information to ensure that this commit processing will inevitably complete (for example, logged that the transaction has committed in a private log). If the RM participant is associated with a nonvolatile RMI, then at some point after receiving this reply, the DECdtm transaction manager will delete the name of the RM participant from the transaction database. After SS$_FORGET replies have been given for all the RM participants in a transaction, that transaction ceases to be recoverable (some time after all these replies are given, the transaction is deleted from the transaction database). A subsequent call to $GETDTI can lead the resource manager to wrongly assume that the transaction had aborted. If there is a failure after this reply is sent, a recoverable resource manager must be able to rely on its own safely stored information to determine if any of the commit processing associated with the RM participant needs to be restarted. **Side effects:** On successful completion of the call to $ACK_EVENT, the RM participant is removed from the transaction, and the ASTLM quota consumed by the call to $JOIN_RM or $ACK_EVENT that added it to the transaction is returned. DECdtm also releases any application threads that are waiting for the transaction to end (necessary but not sufficient condition). Any call to $END_TRANS or $END_BRANCH on a node for a transaction whose outcome is commit cannot complete successfully until all commit event reports delivered to RM participants on that node have been acknowledged.
SS$_REMEMBER	The RM participant requires that the DECdtm transaction manager stores its name and the outcome of the transaction (commit) in the transaction database. Note that for an RM participant associated with a volatile RMI, SS$_REMEMBER is treated in the same way as SS$_FORGET. **Side effects:** On successful completion of the call to $ACK_EVENT, the RM participant is removed from the transaction, and the ASTLM quota consumed by the call to $JOIN_RM or $ACK_EVENT that added it to the transaction is returned. DECdtm also releases any application threads that are waiting for the transaction to end (necessary but not sufficient condition). Any call to $END_TRANS or $END_BRANCH on a node for a transaction whose outcome is commit cannot complete successfully until all commit event reports delivered to RM participants on that node have been acknowledged.

Table 6. Replies to a One-phase Commit Event Report (DDTM $K_ONE_PHASE_COMMIT)

report_reply	Vote	Meaning
SS$_NORMAL	Yes	The RM participant decided to commit the transaction, and has safely stored enough information to be able to keep this guarantee even if there is a recoverable failure, caused, for example, by a node crash. The DECdtm transaction manager does not log any information about the transaction. **Side effects:** On successful completion of the call to $ACK_EVENT, the RM participant is removed from the transaction, the transaction is ended, and the ASTLM quota consumed by the call to $JOIN_RM or $ACK_EVENT that added the RM participant to the transaction is returned. DECdtm also allows the call to $END_TRANS to complete (necessary and sufficient condition).
SS$_PREPARED	Yes	RM participant decided not to accept the opportunity to decide the outcome of the transaction. It has performed only prepare processing for the transaction and requires full two-phase commit processing. This is equivalent to voting SS$_PREPARED on a prepare event. The RM participant can either commit or abort the operations of the transaction, and guarantees that it will abide by the DECdtm transaction manager's decision on whether the transaction (and therefore these operations) are committed or aborted. A recoverable resource manager must not give this vote until it has safely stored enough information to be able to keep this guarantee even if there is a recoverable failure, caused, for example, by a node crash. The DECdtm transaction manager will decide the outcome of the transaction, then inform the resource manager of the decision with a commit or abort event report delivered to the RM participant (assuming that it is associated with an RMI that requested these event reports). Note that an application or other failure can cause the DECdtm transaction manager to decide to abort the transaction.
SS$_VETO	No	RM participant requires that the transaction be aborted and guarantees that the effects of that transaction on its resources will never be detected by any transaction that commits. The *reason* argument gives the reason why the RM participant is aborting the transaction.

report_reply	Vote	Meaning
		The DECdtm transaction manager does not log any information about the transaction. **Side effects:** On successful completion of the call to $ACK_EVENT, the RM participant is removed from the transaction, the transaction is ended, and the ASTLM quota consumed by the call to $JOIN_RM or $ACK_EVENT that added the RM participant to the transaction is returned. DECdtm also allows the call to $END_TRANS to complete (necessary and sufficient condition).

Table 7. Replies to a Prepare Event Report (DDTM$K_PREPARE)

report_reply	Vote	Meaning
SS$_FORGET	Yes	This is called a read-only vote. It is an optimization that allows an RM participant to vote "yes" and not receive a commit or abort event report. **Side effects:** On successful completion of the call to $ACK_EVENT, the RM participant is removed from the transaction, and the ASTLM quota consumed by the call to $JOIN_RM or $ACK_EVENT that added it to the transaction is returned.
SS$_PREPARED	Yes	The RM participant can either commit or abort the operations of the transaction, and guarantees that it will abide by the DECdtm transaction manager's decision on whether the transaction (and therefore these operations) are committed or aborted. In other words, the RM participant guarantees that the behavior of its resources will never be inconsistent with that decision, insofar as that behavior is detected by any transactions that commit. A recoverable resource manager must not give this vote until it has safely stored enough information to be able to keep this guarantee even if there is a recoverable failure, caused, for example, by a node crash. The DECdtm transaction manager will decide the outcome of the transaction, then inform the resource manager of the decision with a commit or abort event report delivered to the RM participant (assuming that it is associated with an RMI that requested these event reports) or, in the event of a failure, using the resource manager's recovery mechanism.
SS$_VETO	No	RM participant requires that the transaction be aborted. The *reason* argument gives the reason why the RM participant is aborting the transaction.

report_reply	Vote	Meaning
		The RM participant guarantees that the effects of its transaction operations will never be detected by any transaction that commits. **Side effects:** The DECdtm transaction manager will deliver an abort event report for the transaction to the RM participant.

Table 8. Replies to a Default Transaction Started Event Report (DDTM $K_STARTED_DEFAULT)

report_reply	Description
SS$_NORMAL	Adds a new RM participant running in the calling process to the transaction to which a new branch is being added. The new RM participant is associated with the RMI to which the default transaction started event was reported. The *part_name* and *rm_context* arguments specify the name of the new RM participant and its context. **Side effects:** The postconditions on successful completion of the call to $ACK_EVENT are the same as those for $JOIN_RM. DECdtm also allows the call to $START_TRANS or $START_BRANCH that is adding the new branch to complete (necessary but not sufficient condition). That call cannot complete successfully until all default transaction-started event reports delivered to RMIs in that process have been acknowledged.
SS$_FORGET	Acknowledgment of the event report. **Side effects:** DECdtm allows the call to $START_TRANS or $START_BRANCH that is adding the new branch to complete (necessary but not sufficient condition). That call cannot complete successfully until all default transaction-started event reports delivered to RMIs in that process have been acknowledged.

Table 9. Replies to a Nondefault Transaction Started Event Report (DDTM $K_STARTED_NONDEFAULT)

report_reply	Description
SS$_NORMAL	Adds a new RM participant running in the calling process to the transaction to which a new branch is being added. The new RM participant is associated with the RMI to which the nondefault transaction started event was reported. The *part_name* and *rm_context* arguments specify the name of the new RM participant and its context. **Side effects:**

report_reply	Description
	The postconditions on successful completion of the call to $ACK_EVENT are the same as those for $JOIN_RM. DECdtm also allows the call to $START_TRANS or $START_BRANCH that is adding the new branch to complete (necessary but not sufficient condition). That call cannot complete successfully until all default transaction-started event reports delivered to RMIs in that process have been acknowledged.
SS$_FORGET	Acknowledgment of the event report. **Side effects:** DECdtm allows the call to $START_TRANS or $START_BRANCH that is adding the new branch to complete (necessary but not sufficient condition). That call cannot complete successfully until all default transaction-started event reports delivered to RMIs in that process have been acknowledged.

reason

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

A code that gives the reason why the RM participant is aborting the transaction.

This argument is ignored unless the value in the *report_reply* argument is SS$_VETO and the event being acknowledged is a prepare or one-phase commit event.

The $DDTMMSGDEF macro defines symbolic names for abort reason codes described in Table 10. The default value for this argument is DDTM$_VETOED.

Table 10. Abort Reason Codes

Symbolic Name	Description
DDTM$_ABORTED	Application aborted the transaction without giving a reason.
DDTM$_COMM_FAIL	Transaction aborted because a communications link failed.
DDTM$_INTEGRITY	Transaction aborted because a resource manager integrity constraint check failed.
DDTM$_LOG_FAIL	Transaction aborted because an attempt to write to the transaction log failed.
DDTM$_ORPHAN_BRANCH	Transaction aborted because it had an unauthorized branch.
DDTM$_PART_SERIAL	Transaction aborted because a resource manager serialization check failed.
DDTM$_PART_TIMEOUT	Transaction aborted because a resource manager timeout expired.
DDTM$_SEG_FAIL	Transaction aborted because a process or image terminated.
DDTM$_SERIALIZATION	Transaction aborted because a serialization check failed.

Symbolic Name	Description
DDTM$_SYNC_FAIL	Transaction aborted because a branch had been authorized for it but had not been added to it.
DDTM$_TIMEOUT	Transaction aborted because its timeout expired.
DDTM$_UNKNOWN	Transaction aborted; reason unknown.
DDTM$_VETOED	Transaction aborted because a resource manager was unable to commit it.

beftime

OpenVMS usage:	utc_date_time
type:	octaword (unsigned)
access:	read only
mechanism:	by reference

Reserved to OpenVMS.

afttime

OpenVMS usage:	utc_date_time
type:	octaword (unsigned)
access:	read only
mechanism:	by reference

Reserved to OpenVMS.

part_name

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

The name of the new RM participant that is added to the transaction by this call to $ACK_EVENT. This argument is ignored unless the event being acknowledged is of type Transaction Started and the value of the *report_reply* argument is SS$_NORMAL.

If this argument is omitted (the default) or its value is zero, the name of the new RM participant is the same of that of the RMI with which it is associated.

The string passed in this argument must be no longer than 32 characters.

To ensure smooth operation in a mixed-network environment, refer to the chapter entitled Managing DECdtm Services in the *VSI OpenVMS System Manager's Manual*, for information on defining node names.

rm_context

OpenVMS usage:	userarg

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

The context associated with the new RM participant. This argument is ignored unless the value of the *report_reply* argument is SS$_NORMAL, and the event being acknowledged is of type Transaction Started.

The context of the new RM participant is passed in the event reports subsequently delivered to that RM participant.

The context is used to pass information specific to the new RM participant from the main line code into the event handler specified in the call to $DECLARE_RM that created the RMI with which the new RM participant is associated.

If this argument is omitted (the default) or is zero, the context associated with the new RM participant is the same of that of the RMI with which it is associated.

timout

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

Reserved to OpenVMS.

Description

The $ACK_EVENT system service:

- Acknowledges an event report delivered by the DECdtm transaction manager to an RM participant or RMI in the calling process.

  Every event report delivered by the DECdtm transaction manager to an RM participant or RMI must be acknowledged by a call to $ACK_EVENT specifying the identifier of the event report. This acknowledgment need not come from AST context. The caller of $ACK_EVENT must be in the same access mode as, or a more privileged access mode than, that in which the event handler AST was delivered.

  The DECdtm transaction manager may deliver multiple event reports to an RMI, but delivers only one event report at a time to an RM participant. For example, if a prepare event report has been delivered to an RM participant, and the transaction is aborted while the RM participant is doing its prepare processing, then the DECdtm transaction manager does not deliver an abort event report to that RM participant until it has acknowledged the prepare event report by a call to $ACK_EVENT.

  After acknowledging the event report, the RMI or RM participant should no longer access the event report block.

- Adds a new RM participant to a transaction, if the event being acknowledged is of type Transaction Started and the value of the *report_reply* argument is SS$_NORMAL.

  Note that the new RM participant cannot be the coordinator of the transaction.

- Removes an RM participant from a transaction if the event being acknowledged is one of the events listed in the following table and the *report_reply* argument is as listed in this table:

Event	report_reply
Abort	SS$_FORGET
Commit	SS$_FORGET or SS$_REMEMBER
Prepare	SS$_FORGET or SS$_VETO
One-phase commit	SS$_NORMAL or SS$_VETO

Required Privileges

None

Required Quotas

None

Related Services

$ABORT_TRANS, $ABORT_TRANSW, $ADD_BRANCH, $ADD_BRANCHW, $CREATE_UID, $DECLARE_RM, $DECLARE_RMW, $END_BRANCH, $END_BRANCHW, $END_TRANS, $END_TRANSW, $FORGET_RM, $FORGET_RMW, $GETDTI, $GETDTIW, $GET_DEFAULT_TRANS, $JOIN_RM, $JOIN_RMW, $SETDTI, $SETDTIW, $SET_DEFAULT_TRANS, $SET_DEFAULT_TRANSW, $START_BRANCH, $START_BRANCHW, $START_TRANS, $START_TRANSW, $TRANS_EVENT, $TRANS_EVENTW

Condition Values Returned

SS$_NORMAL

The request was successful.

SS$_ACCVIO

An argument was not accessible to the caller.

SS$_BADPARAM

Either the options flags were invalid, or the reply passed in the *report_reply* argument was invalid for the type of event being acknowledged.

SS$_BADREASON

The abort reason code passed in the *reason* argument was invalid.

SS$_EXASTLM

The process AST limit (ASTLM) was exceeded.

SS$_ILLEFC

The event flag number was invalid.

SS$_INSFARGS

A required argument was missing.

SS$_INSFMEM

There was insufficient system dynamic memory for the operation.

SS$_INVBUFLEN

The string passed in the *part_name* argument was too long.

SS$_NOSUCHREPORT

Either an event report with the specified report identifier had not been delivered to any RM participant or RMI in the calling process, or that event report had already been acknowledged.

SS$_WRONGACMODE

The caller was in a less privileged access mode than that of the RMI whose event handler was used to deliver the event report that is being acknowledged by this call to $ACK_EVENT.

$ACM

Authentication and Credential Management — The $ACM service provides a common interface to all functions supported by the Authentication and Credential Management (ACM) authority. The caller must specify the function code and any applicable function modifiers and item codes for the requested operation. The $ACM service completes asynchronously; for synchronous completion, use the $ACMW form of the service.

Format

```
SYS$ACM [efn], func, [context], itmlst, acmsb, [astadr], [astprm]
```

C Prototype

```
int sys$acm
   (unsigned int efn, unsigned int func, struct _acmecb **context, void
    *itmlst, struct _acmesb *acmsb, void (*astadr)(__unknown_params),
    int astprm);
```

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 $ACM request completes. The *efn* argument is a longword containing this number; however, $ACM uses only the low-order byte.

When the request is queued, $ACM clears the specified event flag. Then, when the request completes, the specified event flag is set.

func

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

Function code and modifiers specifying the operation that $ACM is to perform. The *func* argument is a longword containing the function code, logically 'OR'ed together with the modifiers. For some programming languages this may be defined as a record structure.

Function codes have symbolic names of the following format:

ACME$_FC_code

Function modifiers have symbolic names of the following format:

ACME$M_modifier

The language support mechanisms specific to each programming language define the names of each function code and modifier. Only one function code can be specified across a dialogue sequence of related calls to $ACM.

Most function codes require or allow additional information to be passed in the call. This information is passed 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 controls the action to be performed, or requests specific information to be returned.

context

OpenVMS usage:	context
type:	longword pointer (signed)
access:	modify
mechanism:	by 32- or 64-bit reference

Address of a longword to receive the 32-bit address of an ACM communications buffer.

$ACM uses the ACM communications buffer for dialogue functions (ACME$_FC_AUTHENTICATE_PRINCIPAL and ACME$_FC_CHANGE_PASSWORD) to request that the caller provide additional information in a subsequent call.

The *context* argument on a continuation call must contain the same ACM communications buffer address returned by $ACM on the previous call. The *itmlst* provided on a continuation call must contain entries to fulfill each of the input item set entries in the ACM communications buffer returned by $ACM on the previous call.

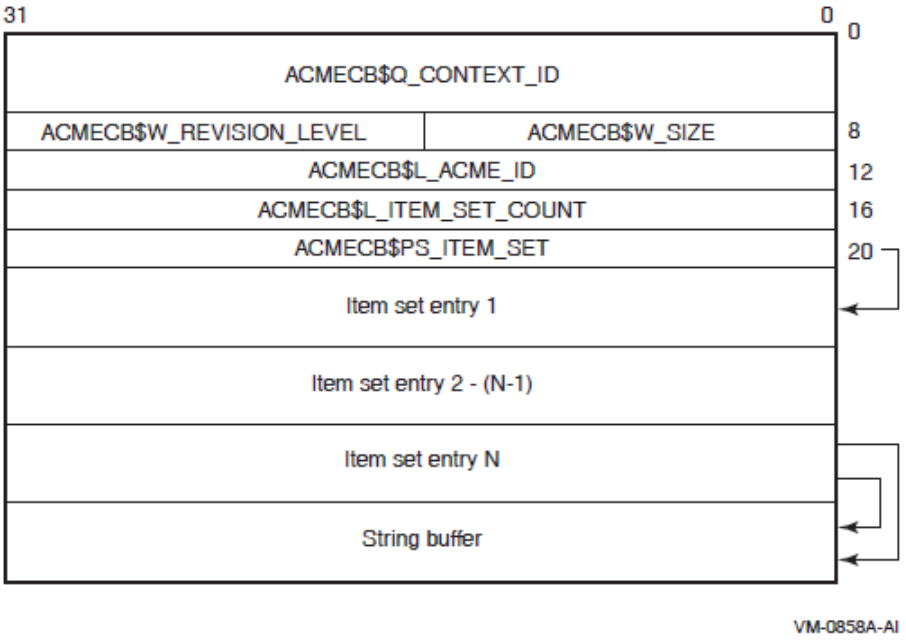
The longword specified must contain a -1 on the first call in a dialogue sequence of related calls. If additional information is required to complete the request, $ACM returns a pointer in the *context* argument to an ACM communications buffer that describes the information. To continue with the particular operation call, $ACM again specifying the *function* argument previously provided.

The ACM communications buffer is user readable, but not user writable. It consists of a structure header, an item set, and a string buffer segment. The item set is an array of item set entries, each describing an item of information that is needed to complete the request or information that can be reported to a human user.

$ACM presents the item set entries in the logical presentation order for a sequential interface, so calling programs that give a sequential view to the user should process the item set entries in that $ACM order. More advanced GUI programs may use simultaneous presentation with distinctive placement for various message types.

The following diagram depicts the overall format of an ACM communications buffer:

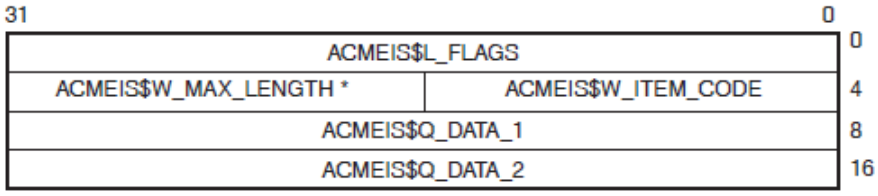
ACM Communications Buffer



The following table defines the ACM communications buffer header fields:

Descriptor Field	Definition
ACMECB$Q_CONTEXT_ID	A quadword containing a value used internally by $ACM to uniquely bind the ACM communications buffer to the associated request across iterative calls.
ACMECB$W_SIZE	A word containing the size of the ACM communications structure.
ACMECB$W_REVISION_LEVEL	A word containing a value that identifies the revision level of the ACM communications structure.
ACMECB$L_ACME_ID	A longword containing the agent ID of the ACME agent that initiated this dialogue sequence.
ACMECB$L_ITEM_SET_COUNT	A longword containing the number of item set entries in the item set.
ACMECB$PS_ITEM_SET	A longword containing the 32-bit address of the item set. The item set is an array of item set entries that describes the information needed by $ACM to complete the particular request.

The following diagram depicts the format of an item set entry:



The following table defines the item set entry fields:

Descriptor Field	Definition
ACMEIS$L_FLAGS	A longword containing a mask of flags that indicates the interpretation of the ACMEIS$Q_DATA_1 and ACMEIS$Q_DATA_2 quadword-sized 32-bit descriptors and the method for processing. Each flag has a symbolic name that the language support mechanisms specific to each programming language define. The following list describes the flags: • ACMEDLOGFLG$V_INPUT When clear: • ACMEIS$W_MSG_TYPE contains the message type associated with this output data. • ACMEIS$Q_DATA_1 and ACMEIS$Q_DATA_2 are 32-bit descriptors that specify text to be displayed to the caller. An address of 0 in either descriptor indicates the calling program should ignore that descriptor. A length of 0 with a nonzero address in either descriptor indicates the calling program should report a separation indication to the user if that is appropriate for the type of human interface involved. A line mode command interface, for instance, will display a blank line. When set: • ACMEIS$W_ITEM_CODE defines the item code to use when providing the requested information. • ACMEIS$Q_DATA_1 is a 32-bit descriptor that specifies prompt text. An address of 0 indicates no prompt text. A length of 0 with a nonzero address is reserved for future definition.

Descriptor Field	Definition
	• Interpretation of ACMEIS$Q_DATA_2 is determined by the state of the ACMEDLOGFLG$V_NOECHO flag. • ACMEDLOGFLG$V_NOECHO When clear: • ACMEIS$Q_DATA_2 is a 32-bit descriptor that specifies the "default" answer that will be assumed by $ACM if the caller responds to this item set entry with an item specifying a length of zero. $ACM provides this "default" answer to the calling program to allow for distinctive display appropriate to the type of interface being supported. For example, traditional line-mode interfaces in VMS will include such a default answer within square brackets when prompting, while a character cell screen interface might pre-fill a field. When set: • The associated input data is not to be echoed. This is typically used to visually protect sensitive data, such as password information. • ACMEIS$Q_DATA_2 is a 32-bit descriptor that specifies secondary prompt text for input verification processing. An address of 0 indicates no verification processing. A length of 0 with a nonzero address is reserved for future definition. The flag ACMEDLOGFLG$V_NOECHO is ignored when ACMEDLOGFLG$V_INPUT is clear.
ACMEIS$W_ITEM_CODE	A word containing the item code that identifies the nature of information associated with the item set. This field defines the item code to use when specifying the requested information in the *itmlst* argument in the subsequent call to $ACM. A sequence of item set entries containing the same item code reflects a linked list of related information. An item set with the ACMEDLOGFLG$V_INPUT flag set or a different item code indicates the end of the related list.
ACMEIS$W_MAX_LENGTH	When ACMEDLOGFLG$V_INPUT is set:A word containing the maximum length (in bytes) of any input data being requested by this item set. For example, the maximum length of a principal name string. A value of 0 indicates that the only item the calling program can return to satisfy the item set entry is an item specifying zero length. When the calling program returns such a zero length item, it thereby indicates confirmation, as might be indicated by a prompt such as "Indicate when you have read that text".

Descriptor Field	Definition
ACMEIS$W_MSG_TYPE	When ACMEDLOGFLG$V_INPUT is clear: A word containing a value that categorizes the output messages described by 32-bit descriptors ACMEIS$Q_DATA_1 and ACMEIS$Q_DATA_2. The language support mechanisms specific to each programming language define the following symbols for the standard message categories: ACMEMC$K_DIALOGUE_ALERT—Text related to the immediately preceding input operation. The calling program should bring this text to the attention of a user prior to handling the renewed input item set entry that typically follows. This message category is used, for example, to indicate that a new password the user has specified does not conform to local security policy. ACM$K_GENERAL—General text. ACM$K_HEADER—Text displayed before a succession of Selections. ACMEMC$K_MAIL_NOTICES—Text related to mail operations, typically an indication that new mail is waiting. ACMEMC$K_PASSWORD_NOTICES—Text indicating that password expiration is imminent. ACM$K_TRAILER—Text displayed after a succession of Selections. ACM$K_SELECTION—Acceptable choices for responding to the subsequent prompt. ACM$K_SYSTEM_IDENTIFICATION—System identification text. ACM$K_SYSTEM_NOTICES—Text displayed to a user before authentication. ACM$K_WELCOME_NOTICES—Text displayed to a user after authentication. ACM$K_LOGON_NOTICES—Logon and logfail statistics. In addition to those standard message categories, individual ACME agents may define their own categories, but the meanings of those categories are useful only to callers of $ACM that are specifically recognized by the ACME agent in question, since those are the only callers of $ACM to which the ACME agent will deliver such ACME-specific categories. Documentation of the ACME-specific codes used by the VMS ACME is shown in the VMS ACME-specific Output Message Categories section of this description. Documentation of ACME-specific codes in general comes in the documentation from the vendor of each ACME agent. For documentation of ACME-specific codes for the VMS ACME, see the section called “VMS ACME-Specific Item Codes”.
ACMEIS$Q_DATA_1	A quadword containing a 32-bit descriptor that describes information (prompt text or other data, as described under the

Descriptor Field	Definition
	prior entry for ACMEDLOGFLG$V_INPUT) applicable to determining an appropriate response.
ACMEIS$Q_DATA_2	A quadword containing a 32-bit descriptor that describes information (default answer, secondary prompt text or other data described under the prior entries for ACMEDLOGFLG$V_INPUT and ACMEDLOGFLG $V_NOECHO) applicable to determining an appropriate response.

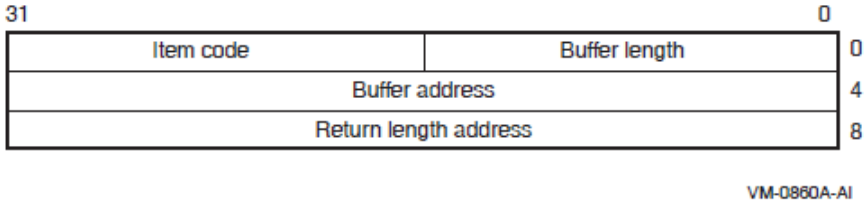
itmlst

OpenVMS usage:	32-bit item_list_3 or 64-bit item_list_64b
type:	longword (unsigned) for 32-bit; quadword (unsigned) for 64-bit
access:	read only
mechanism:	by 32- or 64-bit reference

Item list specifying information to be used in performing the function and requesting information to be returned. The *itmlst* argument is the 32- or 64-bit address of a list of item descriptors, describing an item of information. An item list in 32-bit format is terminated by a longword of 0; an item list in 64-bit format is terminated by a quadword of 0.

The item list can be composed of up to 32 segments, connected by a chain item (using item code ACME$_CHAIN) at the end of all but the last segment pointing to the start of the next segment. All item descriptors in a single segment must be of the same format (32-bit or 64-bit), but a single item list can contain a mixture of segments composed of 32-bit item descriptors and segments composed of 64-bit item descriptors.

The following diagram depicts the 32-bit format of a single item descriptor:



The following table defines the item descriptor fields for 32-bit item list entries:

Descriptor Field	Definition
MBO	A word that must contain a 1. The MBO and MBMO fields are used to distinguish 32-bit and 64-bit item list entries.
Item code	A word containing an item code, which identifies the nature of the information supplied to $ACM or that is received from $ACM. Common item codes have symbolic names (beginning with the characters "ACME$_") defined by the language support mechanisms specific to each programming language. Individual ACME agents may also define additional item codes specific to the functions of those ACME agents.
MBMO	A longword that must contain a -1. The MBMO and MBO fields are used to distinguish 32-bit and 64-bit item list entries.
Buffer length	A quadword specifying the length of the buffer (in bytes); the buffer either supplies information to $ACM or receives information from $ACM. The required

Descriptor Field	Definition
	length of the buffer varies, depending on the item code, and is specified in the description of each item code.
Buffer address	A quadword containing the 64-bit address of the buffer that specifies or receives the information.
Return length address	A quadword containing the 64-bit address of a quadword to receive the length (in bytes) of information returned by $ACM. If specified as 0, no length is returned. For input items the return length address is ignored.

In an item list, no ACME-specific item codes can be included in an item list until the ACME Context has been set with one of the following codes:

ACME$_CONTEXT_ACME_ID
ACME$_CONTEXT_ACME_NAME

You can also implicitly set the ACME Context with one of the following codes:

ACME$_TARGET_DOI_ID
ACME$_TARGET_DOI_NAME

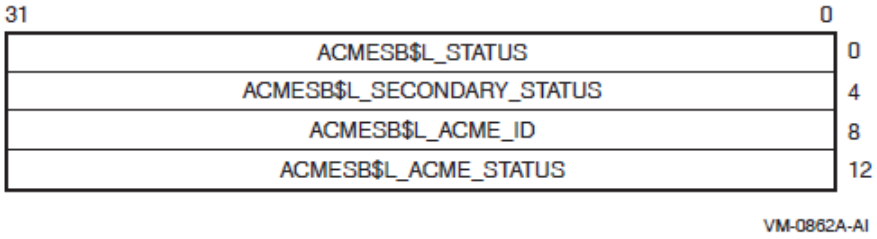
These codes are described in the *VSI OpenVMS Programming Concepts Manual*.

acmsb

OpenVMS usage:	acm_status_block
type:	octaword array of 4 longwords
access:	write only
mechanism:	by 32- or 64-bit reference

ACM status block that is to receive the final completion status. The `acmsb` argument is the 32- or 64-bit address of the ACM status block.

The following diagram depicts the structure of an ACM status block:



The following table defines the ACM status block fields:

Descriptor Field	Definition
ACMESB$L_STATUS	A longword containing the generalized completion status for the requested operation.
ACMESB$L_SECONDARY_STATUS	A longword containing status that may indicate a more detailed description of a failure.
ACMESB$L_ACME_ID	A longword containing the agent ID of the ACME agent that reported additional status. An agent ID of 0 indicates that no additional status was reported.

Descriptor Field	Definition
ACMESB$L_ACME_STATUS	A longword containing additional status information. Aside from a few cases of item list errors described in the following text, the interpretation of this value is specific to the context of the reporting ACME agent.

Upon request initiation, $ACM sets the value of all fields within the ACM status block to 0. When the requested operation completes. The $ACM service writes condition values into the ACMESB$L_STATUS and ACMESB$L_SECONDARY_STATUS fields.

If the status in the ACMESB$L_STATUS field is ACME$_AUTHFAILURE, that is the only result that should be displayed or otherwise made known to a user. The status in ACMESB$L_SECONDARY_STATUS (when the caller has the SECURITY privilege, or is calling from kernel or executive mode) contains the detailed reason for the failure, which may be used for purposes that do not disclose the code to the user, such as the process termination code supplied to $DELPRC (but not the image exit code supplied to $EXIT).

Otherwise, the status in ACMESB$L_SECONDARY_STATUS should be considered as subsidiary to that in ACMESB$L_STATUS and used to form a chained message, unless the two cells contain identical values.

In either case, the caller of $ACM[W] can rely on the success bit (bit 0) of the ACMESB$L_STATUS and the ACMESB$L_SECONDARY_STATUS field having the same setting. Either both low-order bits will be set (success) or both will be clear (failure).

The ACMESB$L_ACME_STATUS field is valid only when the contents of the ACMESB$L_ACME_ID field are nonzero, indicating which ACME agent supplied the (possibly zero) value in ACMESB$L_ACME_STATUS.

There is one special format for such data in ACMESB$L_ACME_STATUS. If $ACM rejects the request because of a bad entry in the item list, then ACMESB$L_STATUS contains one of the following codes:

SS$_BADPARAM	Incorrect contents for the item code
SS$_BADITMCOD	Incorrect item code for the function
SS$_BADBUFLEN	Incorrect length for the item code

If ACMESB$L_STATUS contains one of the listed returns, then ACME$L_ACME_STATUS contains the item code from the incorrect item, which is an aid to debugging.

In all other cases, the value delivered in ACME$L_ACME_STATUS is specific to the ACME agent that failed the request. An ACME agent can return a longword value that indicates additional information about the operation. $ACM writes this value in the ACMESB$L_ACME_STATUS field of the ACM status block.

In these cases, you can expect the success of a valid value (one where ACMESB$L_ACME_ID is not zero) in field ACMESB$L_ACME_STATUS to match the "success" bits (bit 0) in fields ACMESB$L_STATUS and ACMESB$L_SECONDARY_STATUS, although what constitutes a "success" value in ACMESB$L_ACME_STATUS is subject to that interpretation specified for the ACME agent that set the value. In particular, the values in the ACMESB$L_ACME_STATUS field from certain ACME Agents might not be a VMS-style message code.

astadr

OpenVMS usage:	ast_procedure
type:	procedure value
access:	call without stack unwinding
mechanism:	by 32- or 64-bit reference

AST service routine to be executed when $ACM completes. The *astadr* argument is the 32- or 64-bit address of this routine. The AST routine executes at the same access mode as the caller of the $ACM service.

astprm

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

AST parameter to be passed to the AST service routine specified by the *astprm* argument. The *astprm* argument is the longword parameter.

Function Codes

This section describes the various function codes supported by the $ACM service and lists the function modifiers and item codes relevant to each function:

Function Code	Purpose
ACME$_FC_AUTHENTICATE_ PRINCIPAL	Perform authentication and provide credentials.
ACME$_FC_CHANGE_PASSWORD	Select a new password.
ACME$_FC_EVENT	Log an event to an ACME agent.
ACME$_FC_FREE_CONTEXT	Abandon a dialogue mode Authentication or Password Change before it has completed.
ACME$_FC_QUERY	Retrieve information from an ACME agent.
ACME$_FC_RELEASE_CREDENTIALS	Give up some of the credentials from a persona.

See the section called “Description” for information relating to the modes of operation and privilege requirements.

ACME$_FC_AUTHENTICATE_PRINCIPAL

The ACME$_FC_AUTHENTICATE_PRINCIPAL function requests authentication of a principal based on standard or site-specific authentication criteria and optionally requests credentials authorized for that principal.

You can specify the principal name using the ACME$_PRINCIPAL_NAME item code. If you do not specify it on the initial call to $ACM, the ACME agents supporting $ACM will try to use another method for determining the principal name. For example, traditional VMS autologin processing determines a principal name based on your terminal name. Of if your call to $ACM was in dialogue mode, the ACM communication buffer returned may prompt you to supply a principal name.

You can also guide how your request is to be processed by directing it toward a specific domain of interpretation (DOI) with either the ACME$_TARGET_DOI_NAME or ACME$_TARGET_DOI_ID item code. Using that technique ensures that your request will be handled by the ACME agents that support the specified domain of interpretation. If that domain of interpretation is not configured on the system at the time of your call, however, your request will fail. Leaving the domain of interpretation to be defaulted increases your chance of successful authentication, but does not guarantee any particular set of credentials beyond the normal VMS credentials.

When the domain of interpretation under which your request is handled is anything other than VMS, the ACME agents that support that domain of interpretation will indicate an OpenVMS user name to which the principal name should be mapped. In this case, the OpenVMS user name must correspond to a valid entry in the OpenVMS authorization database (SYSUAF.DAT) that has the UAF$V_EXTAUTH flag set. (When the IGNORE_EXTAUTH flag is set in system parameter SECURITY_POLICY, the UAF $V_EXTAUTH flag requirement does not apply.)

The VMS ACME agent uses that OpenVMS user name to provide supplemental authorization processing (for example, account disabled or expired, or model restrictions) and to provide security profile and quota information applicable after a successful authentication.

You can use the ACME$M_ACQUIRE_CREDENTIALS function modifier to specify that, if your authentication call is successful, you want credentials returned. $ACM will return those credentials via persona extensions attached to a persona whose handle is returned to a location specified by item code ACME$_PERSONA_HANDLE_OUT.

If you want that persona to be based on some persona you already have, specify the existing persona handle with the item code ACME$_PERSONA_HANDLE_IN and, in addition to the function modifier ACME$M_ACQUIRE_CREDENTIALS, specify one of the following two function modifiers:

- ACME$M_MERGE_PERSONA—Requests that additional credentials you acquire be added into the existing persona
- ACME$M_COPY_PERSONA—Requests that additional credentials you acquire be added into a copy of the existing persona

In either case, a handle to the resulting persona will be returned as specified by item code ACME $_PERSONA_HANDLE_OUT.

When a new persona is created, the ISS$_PRIMARY_EXTENSION designator indicates which persona extension representing the domain of interpretation was responsible for authenticating the user.

On a subsequent call $ACM will use that designator to guide processing of the ACME $M_DEFAULT_PRINCIPAL function modifier, for instance when there is an ACME $_FC_CHANGE_PASSWORD request.

ACME$_FC_CHANGE_PASSWORD

The ACME$_FC_CHANGE_PASSWORD function performs a password change operation. All aspects of the ACME$FC_CHANGE_PASSWORD function can also be performed as part of the ACME$_FC_AUTHENTICATE_PRINCIPAL function. Some degree of the ACME$_FC_AUTHENTICATE_PRINCIPAL function is also performed as part of ACME $_FC_CHANGE_PASSWORD to ensure the identity of the user changing the password. The primary and secondary passwords can be changed independently.

This function requires the ACME$_NEW_PASSWORD_FLAGS item code.

ACME$_FC_EVENT

The ACME$_FC_EVENT function provides a simple logging feature that can be used to generate certain events related to the policy of a domain of interpretation. To log an event, supply the desired "event type" item code followed by the appropriate "data" item codes pertaining to the "target" domain of interpretation.

To determine what event processing might be available, see the documentation provided by the vendors of the supporting ACME agents.

ACME$_FC_FREE_CONTEXT

The ACME$_FC_FREE_CONTEXT function is used to terminate iterative processing of a request. The address of the ACM communications buffer associated with the request must be specified using the **context** argument.

ACME$_FC_QUERY

The ACME$_FC_QUERY function provides a simple key-based query feature that can be used to obtain certain information related to the policy of a domain of interpretation. To look up an item of information, supply the desired "key" item code followed by the appropriate "data" item code.

To determine what query processing might be available, see the documentation provided by the vendors of the supporting ACME agents.

ACME$_FC_RELEASE_CREDENTIALS

The ACME$_FC_RELEASE_CREDENTIALS function removes credentials for a particular domain of interpretation from the specified persona. When the domain of interpretation is specified as "VMS", all non-native credentials are released and the persona is deleted. The "VMS" credentials cannot be removed from either the currently active or the process' natural persona. Thus, you cannot use the $ACM service to delete these personae. Function Modifiers This section describes the various function modifiers for the function codes supported by the $ACM service.

Table 11 indicates which Function Modifiers are applicable to the various Function Codes:

Table 11. Function Codes and Function Modifiers

Function Modifiers	**Function Codes**					
	Authenticate Principal	**Change Password**	**Event**	**Free Context**	**Query**	**Release Credentials**
ACME$M_ACQUIRE_ CREDENTIALS	**IP**					
ACME$M_COPY_ PERSONA	◆					
ACME$M_DEFAULT_	◆	◆				
Key to Codes **♦—Permitted** **IP—IMPERSONATE Privilege Required for the MAPPED _VMS _USERNAME to differ from the one current when the initial call to $ACM is made** **IR—IMPERSONATE Privilege Required to override default values** **SR—SECURITY Privilege Required**						

Function Modifiers	Function Codes					
	Authenticate Principal	**Change Password**	**Event**	**Free Context**	**Query**	**Release Credentials**
PRINCIPAL						
ACME$M_FOREIGN_ POLICY_HINTS	**SR**					
ACME$M_MERGE_ PERSONA	◆					
ACME$M_NOAUDIT	**SR**					
ACME$M_ NOAUTHORIZATION	**SR**					
ACME$M_ OVERRIDE_ MAPPING	**IR**					
ACME$M_TIMEOUT						
ACME$M_UCS2_4	◆	◆	◆		◆	◆

Key to Codes

♦—Permitted
IP—IMPERSONATE Privilege Required for the MAPPED _VMS _USERNAME to differ from the one current when the initial call to $ACM is made
IR—IMPERSONATE Privilege Required to override default values
SR—SECURITY Privilege Required

ACME$M_ACQUIRE_CREDENTIALS

The ACME$M_ACQUIRE_CREDENTIALS function modifier requests credentials be acquired during a successful authentication.

ACME$M_COPY_PERSONA

The ACME$M_COPY_PERSONA function modifier requests acquired credentials be attached to a copy of the persona specified with item code ACME$_PERSONA_HANDLE_IN.

ACME$M_DEFAULT_PRINCIPAL

The ACME$M_DEFAULT_PRINCIPAL specifies that the principal name and target domain of interpretation should be taken from the input persona, such as for changing the password of the logged-in user or reauthenticating the logged-in user.

ACME$M_FOREIGN_POLICY_HINTS

The ACME$M_FOREIGN_POLICY_HINTS function modifier indicates ACME agents should honor the ACME$M_NOAUDIT and ACME$M_NOAUTHORIZATION function modifiers for non-VMS domains of interpretation.

ACME$M_MERGE_PERSONA

The ACME$M_MERGE_PERSONA function modifier requests acquired credentials be attached to the persona specified with item code ACME$_PERSONA_HANDLE_IN.

ACME$M_NOAUDIT

The ACME$M_NOAUDIT function modifier indicates that auditing actions should not be performed. Unless the ACME$M_FOREIGN_POLICY_HINTS function modifier is also specified, this modifier applies only to the VMS domain of interpretation.

ACME$M_NOAUTHORIZATION

The ACME$M_NOAUTHORIZATION function modifier indicates authorization restrictions, such as the enforcement of modal constraints, should not apply. This provides a mechanism for performing pure authentication operations. Unless the ACME$M_FOREIGN_POLICY_HINTS function modifier is also specified, this modifier applies only to the VMS domain of interpretation.

ACME$M_OVERRIDE_MAPPING

The ACME$M_OVERRIDE_MAPPING function modifier allows for the acquisition of non-VMS credentials during a persona merge or copy operation. This occurs when an externally authorized principal name maps to an OpenVMS user name that differs from the user name associated with the native (VMS) credentials. By default, mixing credentials is prohibited.

ACME$M_TIMEOUT

The ACME$M_TIMEOUT modifier indicates that the caller requests timeout processing. The timeout interval is specified by the ACME$_TIMEOUT_INTERVAL item code.

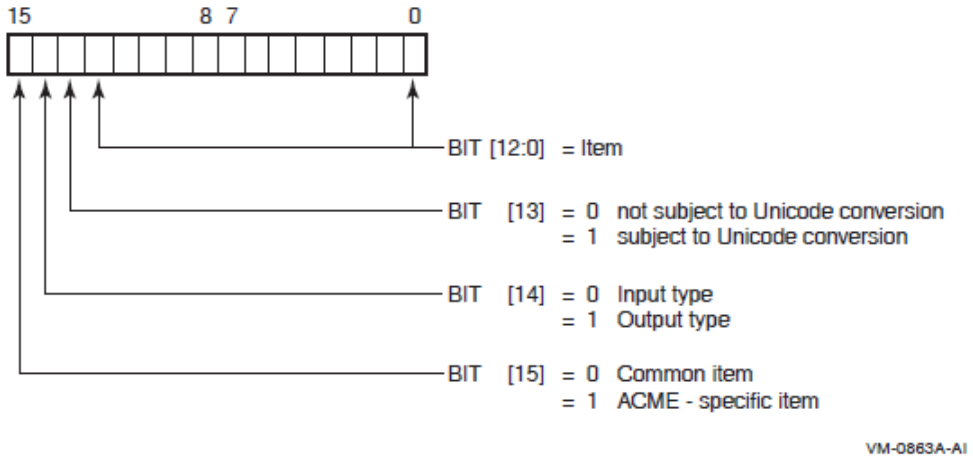
Timeout processing is always enforced for non-privileged callers. Privileged callers (those running in exec mode or kernel mode or possessing SECURITY privilege) must explicitly specify ACME $M_TIMEOUT for timeout processing to be enforced.

ACME$M_UCS2_4

The ACME$M_UCS2_4 function modifier indicates item codes that specify string values use a 4-byte UCS-2 (Unicode) representation rather than 8-bit ASCII.

Item Code Encoding

Item codes are 16-bit unsigned values and are encoded as follows:



The item codes can be categorized in three different ways and are described as follows:

- Common and ACME-specific Item Codes

    - Common item codes

      These item codes are defined for the $ACM system service itself and are available to all ACME agents.

    - ACME-specific item codes

      These item codes are defined separately for each ACME agent.

- Input and Output Item Codes

    - Input item code

      Input item codes specify a buffer that contains information provided to $ACM. The buffer length and buffer address fields in the item descriptor must be nonzero; the return length field is ignored.

    - Output item code

      Output item codes specify a buffer in which $ACM is to return information. The buffer length and buffer address fields of the item descriptor must be nonzero; the return length field can be zero or nonzero.

- Subject and Not Subject to Unicode Conversion

    - Subject to Unicode Conversion

      Text strings can be specified as Latin1 or 4-byte UCS-2 characters, depending on the setting of the ACME$M_UCS2_4 function modifier. An item code that is subject to Unicode conversion indicates it is a text item.

    - Not subject to Unicode Conversion

      Item codes that are not subject to Unicode conversion have a data format implied by the item code, and the nature of the data format must be explicitly understood by the programmer who calls $ACM.

See the section called “Common Item Codes” for a description of the common item codes and their data formats.

Documentation of ACME-specific codes in general comes in the documentation from the vendor of each ACME agent.

For documentation of ACME-specific codes for the VMS ACME, see the section called “VMS ACME-Specific Item Codes”.

Common Item Codes

This section describes the common item codes for the function codes supported by the $ACM service.

The item code space is partitioned into common items and ACME-specific items. ACME-specific items are used to request information that is unique to a particular domain of interpretation. The item codes described in this section fall into the common item code space.

Table 12 indicates which Common Item Codes are applicable to the various Function Codes:

Table 12. Function Codes and Common Item Codes

Item Codes	Function Codes					
	Authenticate Principal	**Change Password**	**Event**	**Free Context**	**Query**	**Release Credentials**
ACME$_ACCESS_ MODE	◆					
ACME$_ACCESS_ PORT **(U)**	**IR**	**IR**				
ACME$_AUTH_ MECHANISM	◆					
ACME$_ AUTHENTICATING_ DOI_ID **(O)**	◆	◆				
ACME$_ AUTHENTICATING_ DOI_NAME **(U,O)**	◆	◆				
ACME$_CHAIN	◆	◆	◆		◆	◆
ACME$_ CHALLENGE_DATA	**IR**					
ACME$_CONTEXT_ ACME_ID **(U)**	◆	◆				
ACME$_CONTEXT_ ACME_NAME	◆	◆				
ACME$_ CREDENTIALS_ NAME **(U)**						◆[2]
ACME$_ CREDENTIALS_TYPE						◆[2]
ACME$_ DIALOGUE_SUPPORT	◆	◆				
ACME$_EVENT_			◆			

Key to Codes

♦—Permitted
IR—IMPERSONATE Privilege Required to override default values
SR—SECURITY Privilege Required
O—Output item code
U—Subject to Unicode Conversion

Item Codes	Function Codes					
	Authenticate Principal	**Change Password**	**Event**	**Free Context**	**Query**	**Release Credentials**
DATA_IN						
ACME$_EVENT_ DATA_OUT **(O)**			◆			
ACME$_EVENT_ TYPE			◆			
ACME$_LOCALE (**U**)	◆	◆				
ACME$_LOGON_ INFORMATION **(O)**	◆					
ACME$_LOGON_ TYPE	◆					
ACME$_MAPPED_ VMS_USERNAME **(U,O)**	◆	◆				
ACME$_MAPPING_ ACME_ID **(O)**	◆	◆				
ACME$_MAPPING_ ACME_NAME **(U,O)**	◆	◆				
ACME$_NEW_ PASSWORD_1 **(U)**	◆	◆				
ACME$_NEW_ PASSWORD_2 **(U)**	◆	◆				
ACME$_NEW_ PASSWORD_FLAGS	◆	◆				
ACME$_NEW_ PASSWORD_ SYSTEM	SR	SR				

Key to Codes

♦—Permitted
IR—IMPERSONATE Privilege Required to override default values
SR—SECURITY Privilege Required
O—Output item code
U—Subject to Unicode Conversion

Item Codes	Function Codes					
	Authenticate Principal	**Change Password**	**Event**	**Free Context**	**Query**	**Release Credentials**
(U)						
ACME$_NULL	◆	◆	◆		◆	◆
ACME$_ PASSWORD_1 **(U)**	◆	◆				
ACME$_ PASSWORD_2 **(U)**	◆	◆				
ACME$_ PASSWORD_SYSTEM **(U)**	◆	◆				
ACME$_PERSONA_ HANDLE_IN	◆					
ACME$_PERSONA_ HANDLE_OUT **(O)**	◆					
ACME$_PHASE_ TRANSITION **(O)**						
ACME$_PRINCIPAL_ NAME_IN **(U)**	◆	◆				
ACME$_PRINCIPAL_ NAME_OUT **(U,O)**	◆	◆				
ACME$_QUERY_ DATA **(O)**					◆[1]	
ACME$_QUERY_ KEY_TYPE					◆[1]	
ACME$_QUERY_					◆[1]	

Key to Codes

♦—Permitted
IR—IMPERSONATE Privilege Required to override default values
SR—SECURITY Privilege Required
O—Output item code
U—Subject to Unicode Conversion

Item Codes	Function Codes					
	Authenticate Principal	**Change Password**	**Event**	**Free Context**	**Query**	**Release Credentials**
KEY_VALUE						
ACME$_QUERY_TYPE					◆[1]	
ACME$_REMOTE_HOST_ADDRESS	**IR**	**IR**				
ACME$_REMOTE_HOST_ADDRESS_TYPE	**IR**	**IR**				
ACME$_REMOTE_HOST_FULLNAME **(U)**	**IR**	**IR**				
ACME$_REMOTE_HOST_NAME **(U)**	**IR**	**IR**				
ACME$_REMOTE_USERNAME **(U)**	**IR**	**IR**				
ACME$_RESPONSE_DATA	◆					
ACME$_SERVER_NAME_IN **(U)**				◆		
ACME$_SERVER_NAME_OUT **(U,O)**				◆		
ACME$_SERVICE_NAME **(U)**	**IR**	**IR**	**IR**	**IR**	**IR**	**IR**
ACME$_TARGET_DOI_ID	◆	◆	◆[2]		◆[2]	
ACME$_TARGET_DOI_NAME	◆	◆	◆[2]		◆[2]	

Key to Codes

♦—Permitted
IR—IMPERSONATE Privilege Required to override default values
SR—SECURITY Privilege Required
O—Output item code
U—Subject to Unicode Conversion

Item Codes	Function Codes					
	Authenticate Principal	**Change Password**	**Event**	**Free Context**	**Query**	**Release Credentials**
(U)						
ACME$_TIMEOUT_ INTERVAL						
Key to Codes **♦—Permitted** **IR—IMPERSONATE Privilege Required to override default values** **SR—SECURITY Privilege Required** **O—Output item code** **U—Subject to Unicode Conversion**						

[1]Required

[2]Either ID or Name Required

ACME$_ACCESS_MODE

The ACME$_ACCESS_MODE item code is an input item code. It specifies the access mode at which a new persona, resulting from credential acquisition processing, is to be created. The buffer must contain a longword value specifying the access mode.

The $PSLDEF macro defines the following symbols for the four access modes:

PSL$C_KERNEL
PSL$C_EXEC
PSL$C_SUPER
PSL$C_USER

The most privileged access mode used is the access mode of the caller. The specified access mode and the access mode of the caller are compared. The less privileged of the two access modes becomes the access mode at which the persona is created.

ACME$_ACCESS_PORT

The ACME$_ACCESS_PORT item code is an input item code. It specifies the name of local device (for example, a terminal) applicable to an authentication request. The buffer must contain a case-insensitive name string.

If not specified, $ACM passes the name string contained in the PCB$T_TERMINAL field of the process control block for the process, or, if that is empty, for the nearest ancestor process (if any) where the PCB$T_TERMINAL field is not empty.

ACME$_AUTH_MECHANISM

The ACME$_AUTH_MECHANISM item code is an input item code. It specifies the authentication mechanism applicable to an authentication request. The buffer must contain a longword value specifying the desired mechanism code. If not specified, the authenticating domain of interpretation applies its default mechanism.

The $ACMEDEF macro defines the following symbols for the standard mechanism types:

ACMEMECH$K_CHALLENGE_RESPONSE
ACMEMECH$K_PASSWORD

Individual ACME agents may define their own authentication mechanisms specific to their domain of interpretation.

ACME$_AUTHENTICATING_DOI_ID

The ACME$_AUTHENTICATING_DOI_ID item code is an output item code. It specifies the buffer to receive the agent ID of the domain of interpretation that successfully authenticated the principal.

ACME$_AUTHENTICATING_DOI_NAME

The ACME$_AUTHENTICATING_DOI_NAME item code is an output item code. It specifies the buffer to receive the name of the domain of interpretation that successfully authenticated the principal.

The maximum data returned for this item code is the number of characters represented by the symbol, ACME$K_MAXCHAR_DOI_NAME, so a caller's buffer should be at least that long, with the number of bytes allocated dependent on whether the ACME$M_UCS2_4 function code modifier was specified on the call to $ACM[W].

ACME$_CHAIN

The ACME$_CHAIN item code is an input item code. It specifies the address of the next item list segment to process immediately after processing the current list segment.

The buffer address field in the item descriptor specifies the address of the next item list segment to be processed. The ACME$_CHAIN item code must be last in the item list segment; $ACM treats this as the logical end of the current item list segment. Any item list entries following the ACME$_CHAIN item code are ignored.

On Alpha and Integrity servers platforms, both 32- and 64-bit item lists can be chained together.

ACME$_CHALLENGE_DATA

The ACME$_CHALLENGE_DATA item code is an input item code. It specifies the challenge data that was used as the basis for generating the response data specified by the ACME$_RESPONSE_DATA item code. The meaning of this data is specific to the domain of interpretation for which it is used.

If this item code is specified, the ACME$_AUTH_MECHANISM and ACME$_RESPONSE_DATA item codes must also be specified. (The VMS domain of interpretation does not support this mechanism type.)

ACME$_CONTEXT_ACME_ID

The ACME$_CONTEXT_ACME_ID item code is an input item code. It establishes the ACME agent context within which ACME-specific item codes are interpreted. This item code has an effect on the parsing of the list of ACME-specific item codes, and takes effect immediately. It is in effect until the next instance of code ACME$_CONTEXT_ACME_ID, code ACME$_CONTEXT_ACME_NAME, code ACME$_TARGET_DOI_ID, or code ACME$_TARGET_DOI_NAME. The buffer must contain a longword value specifying the agent ID of an ACME agent.

ACME$_CONTEXT_ACME_NAME

The ACME$_CONTEXT_ACME_NAME item code is an input item code. It establishes the ACME agent context within which ACME-specific item codes are interpreted. This item code has an effect on the parsing of the list of ACME-specific item codes, and takes effect immediately. It is in effect until the next instance of code ACME$_CONTEXT_ACME_ID, code ACME$_CONTEXT_ACME_NAME, code ACME_TARGET_DOI_ID, or code ACME$_TARGET_DOI_NAME. The buffer must contain the case-insensitive name string of an ACME agent.

ACME$_CREDENTIALS_NAME

The ACME$_CREDENTIALS_NAME item code is an input item code. It specifies the name of the persona extension holding the set of credentials upon which to operate. The buffer must contain the case-insensitive name string of a persona extension that has been registered on the system.

ACME$_CREDENTIALS_TYPE

The ACME$_CREDENTIALS_TYPE item code is an input item code. It specifies the extension ID of the persona extension holding the set of credentials upon which to operate. The buffer must contain a longword value specifying the extension ID of a persona extension that has been registered on the system.

ACME$_DIALOGUE_SUPPORT

The ACME$_DIALOGUE_SUPPORT item code is an input item code. It specifies which dialogue mode features are supported by the caller. The buffer must contain a longword bit vector of applicable flags.

The $ACMEDEF macros defines the following symbols for the valid flags:

ACMEDLOGFLG$V_INPUT—character string input/output capabilities
ACMEDLOGFLG$V_NOECHO—"no echo" input capabilities

These are the same as those for the ACMEIS$L_FLAGS field on an item set entry. See the description of the *context* argument for further information.

Specify the ACME$_DIALOGUE_SUPPORT item code to indicate the interactive capabilities of the user interface. If the caller is unable to support features necessary to complete a given request, the request ultimately fails. The caller receives a condition value ACME$_INSFDIALSUPPORT for insufficient dialogue support.

ACME$_EVENT_DATA_IN

The ACME$_EVENT_DATA_IN item code is an input item code. It specifies the buffer containing information applicable to an event operation.

The meaning of this data is specific to the domain of interpretation for which it is used.

ACME$_EVENT_DATA_OUT

The ACME$_EVENT_DATA_OUT item code is an output item code. It specifies the buffer to receive information returned from an event operation.

The meaning of this data is specific to the domain of interpretation for which it is used.

ACME$_EVENT_TYPE

The ACME$_EVENT_TYPE item code is an input item code. It specifies the type of event being reported. The buffer must contain a longword value. Interpretation of the value is specific to the domain of interpretation to which the event is being reported.

ACME$_LOCALE

The ACME$_LOCALE item code is an input item code. It specifies the collection of data and rules applicable to a language and culture. The buffer must contain a name string that reflects a locale supported by the system.

The buffer must contain a string in the following case-insensitive syntax:

language-country

language is a 2-letter language code (ISO 639)

country is a 2-letter country code (ISO 3166)

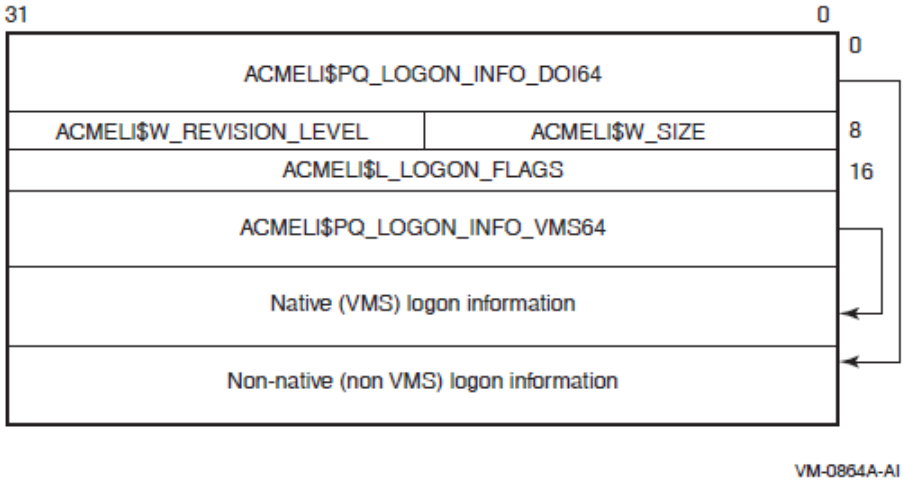
The default is EN-US, and cannot be overridden by the specified locale. Locale information may be interpreted by ACME agents to determine country and language requirements.

ACME$_LOGON_INFORMATION

The ACME$_LOGON_INFORMATION item code is an output item code. It specifies the buffer to receive an ACM logon information structure, which contains statistics pertaining to the authenticated principal name within the contexts of the authenticating and native (VMS) domains of interpretation.

The size of the buffer must be sufficient to handle data from whatever VMS versions are used, as described in the ACME$_LOGON_INFORMATION structure found in the *VSI OpenVMS Programming Concepts Manual*.

The following diagram depicts the overall format of an ACM logon information structure:

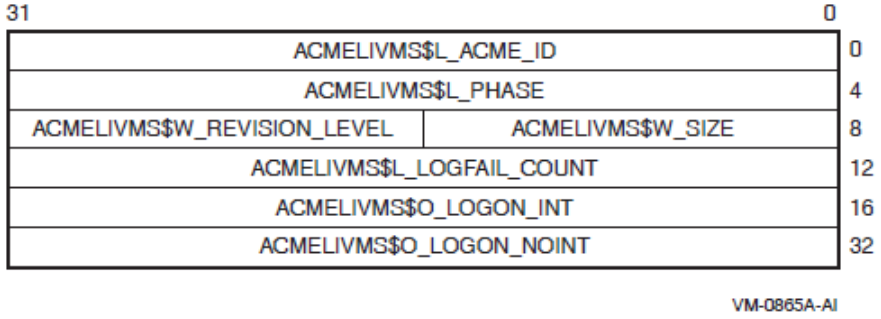


The following table defines the ACM logon information structure header fields:

Descriptor Field	Definition
ACMELI$PQ_LOGON_INFO_DOI64	In this situation, a quadword containing the 64-bit address of the structure segment containing logon information relating to the authenticating domain of interpretation. When the ACM logon information structure resides in 32-bit address space, ACMELI$PQ_LOGON_INFO_DOI64 contains the sign-extended 32-bit address of the structure segment. The field can be referenced as a 32-bit signed pointer using ACMELI $PS_LOGON_INFO_DOI32.
ACMELI$W_SIZE	A word containing the size of the ACM logon information structure.
ACMELI$W_REVISION_LEVEL	A word containing a value that identifies the revision level of the ACM logon information structure.
ACMELI$L_LOGON_FLAGS	Specifies the structure ACMELGIFLG$TYPE, used by LOGINOUT to populate the longword returned by the

Descriptor Field	Definition
	item code JPI$_LOGIN_FLAGS when calling the SYS $GETJPI[W] system service. This provides the client with information regarding what took place during authentication. The ACM Dispatcher manages this item, sending back to the client the merge of all the output it receives from ACMEs by calls to the ACME$CB_SET_LOGIN_FLAG. For the information that is received, see the *VSI OpenVMS Programming Concepts Manual*.
ACMELI$PQ_LOGON_ INFO_VMS64	In this situation, a quadword containing the 64-bit address of the structure segment containing logon information about the native (VMS) domain of interpretation. When the ACM logon information structure resides in 32-bit address space, ACMELI$PQ_LOGON_INFO_VMS64 contains the sign-extended 32-bit address of the structure segment. The field can be referenced as a 32-bit signed pointer using ACMELI $PS_LOGON_INFO_VMS32.

The following diagram depicts the format of the ACM logon structure segment containing information about the VMS domain of interpretation:

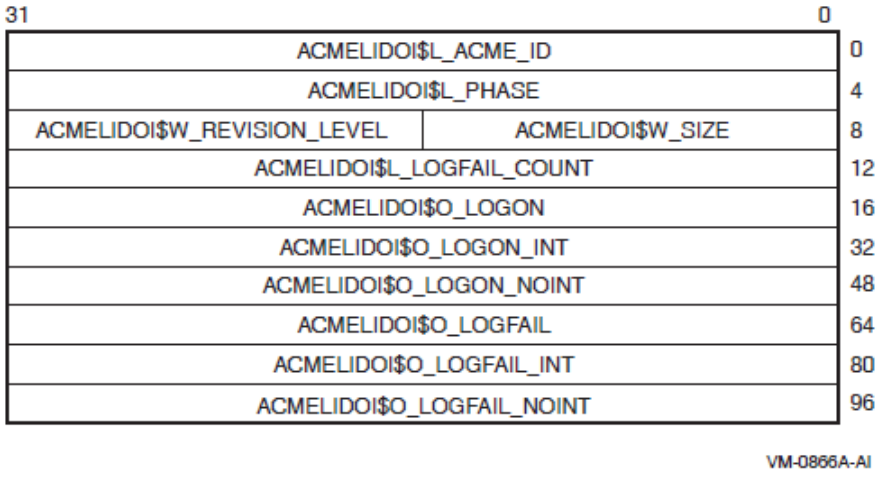


The following table defines the fields for the ACM logon structure segment containing logon information about the native (VMS) domain of interpretation:

Descriptor Field	Definition
ACMELIVMS$L_ACME_ID	A longword containing the agent ID of the ACME agent that reported logon information for the native (VMS) domain of interpretation. If this field is zero, the rest of the structure segment is invalid.
ACMELIVMS$L_PHASE	Indicates the ACME Execution Phase during which this value was provided. ACME Execution Phase numbers are subject to change, so this field is mainly for use by programmers to debug an ACME agent.
ACMELIVMS$W_SIZE	A word containing the size of the ACM logon information structure segment.
ACMELIVMS$W_REVISION_ LEVEL	A word containing a value that identifies the revision level of the ACM logon information structure segment.
ACMELIVMS$L_LOGFAIL_COUNT	A longword containing the number of failed logon attempts with respect to the VMS domain of interpretation.
ACMELIVMS$O_LOGON_INT	An octaword containing the date and time in UTC format of the last interactive logon with respect to the VMS domain

Descriptor Field	Definition
	of interpretation. If the contents of the octaword are zero, no previous non-interactive logon with respect to the VMS domain of interpretation was recorded.
ACMELIVMS$O_LOGON_NONINT	An octaword containing the date and time in UTC format of the last noninteractive logon with respect to the VMS domain of interpretation. If the contents of the octaword are zero, no previous non-interactive logon with respect to the VMS domain of interpretation was recorded.

The following diagram depicts the format of the ACM logon structure segment containing information about the authenticating domain of interpretation:



The following table defines the fields for the ACM logon structure segment containing logon information about the authenticating domain of interpretation:

Descriptor Field	Definition
ACMELIDOI$L_ACME_ID	A longword containing the agent ID of the ACME agent that reported logon information about the non-native authenticating domain of interpretation. If this field is zero, the rest of the structure segment is invalid. If the contents of the longword are zero, the principal was authenticated for the VMS domain of interpretation.
ACMELIDOI$L_PHASE	Indicates the ACME Execution Phase during which this value was provided. ACME Execution Phase numbers are subject to change, so this field is mainly for use by programmers to debug an ACME agent.
ACMELIDOI$W_SIZE	A word containing the size of the ACM logon information structure segment.
ACMELIDOI$W_REVISION_LEVEL	A word containing a value that identifies the revision level of the ACM logon information structure segment.
ACMELIDOI$L_LOGFAIL_COUNT	A longword containing the number of failed logon attempts with respect to the non-native authenticating domain of interpretation.
ACMELIDOI$O_LOGON	An octaword containing the date and time in UTC format of the last logon with respect to the non-native authenticating domain of interpretation. If the contents of the octaword

Descriptor Field	Definition
	are zero, no previous logon with respect to the domain of interpretation was recorded.
ACMELIDOI$O_LOGON_INT	An octaword containing the date and time in UTC format of the last interactive logon with respect to the non-native authenticating domain of interpretation. If the contents of the octaword are zero, no previous interactive logon with respect to the domain of interpretation was recorded.
ACMELIDOI$O_LOGON_NONINT	An octaword containing the date and time in UTC format of the last noninteractive logon with respect to the non-native authenticating domain of interpretation. If the contents of the octaword are zero, no previous non-interactive logon with respect to the domain of interpretation was recorded.
ACMELIDOI$O_LOGFAIL	An octaword containing the date and time in UTC format of the last logon failure with respect to the non-native authenticating domain of interpretation. If the contents of the octaword are zero, no previous logon failure with respect to the domain of interpretation was recorded.
ACMELIDOI$O_LOGFAIL_INT	An octaword containing the date and time in UTC format of the last interactive logon failure with respect to the non-native authenticating domain of interpretation. If the contents of the octaword are zero, no previous interactive logon failure with respect to the domain of interpretation was recorded.
ACMELIDOI$O_LOGFAIL_NONINT	An octaword containing the date and time in UTC format of the last noninteractive logon failure with respect to the non-native authenticating domain of interpretation. If the contents of the octaword are zero, no previous non-interactive logon failure with respect to the domain of interpretation was recorded.

ACME$_LOGON_TYPE

The ACME$_LOGON_TYPE item code is an input item code. It specifies the type of logon being performed. The buffer must contain a longword value specifying a valid type. If not specified, the value defaults to the logon type of the calling process.

The $ACMEDEF macro defines the following symbols for the valid logon types:

ACME$K_DIALUP
ACME$K_LOCAL
ACME$K_REMOTE
ACME$K_BATCH
ACME$K_NETWORK

The values ACME$K_BATCH and zero (0) for batch and detached processes, respectively, are reserved to LOGINOUT.EXE. If either of these values is defaulted or specified by non-LOGINOUT clients, the service returns ACME$_INVREQUEST.

ACME$_MAPPED_VMS_USERNAME

The ACME$_MAPPED_VMS_USERNAME item code is an output item code. It specifies the buffer to receive the name of the local OpenVMS user name to which the principal name was mapped.

The maximum data returned for this item code is the number of characters represented by the symbol, ACMEVMS$S_MAX_VMS_USERNAME, so a caller's buffer should be at least that long, with the number of bytes allocated dependent on whether the ACME$M_UCS2_4 function code modifier was specified on the call to $ACM[W].

ACME$_MAPPING_ACME_ID

The ACME$_MAPPING_ACME_ID item code is an output item code. It specifies the buffer to receive the agent ID of the ACME agent that successfully mapped the principal name to an OpenVMS user name. The buffer descriptor must specify a longword.

ACME$_MAPPING_ACME_NAME

The ACME$_MAPPING_ACME_NAME item code is an output item code. It specifies the buffer to receive the name of the ACME agent that successfully mapped the principal name to an OpenVMS user name.

Data returned for this item code is the number of characters represented by the symbol, ACME $K_MAXCHAR_DOI_NAME, so a caller's buffer should be at least that long, with the number of bytes allocated dependent on whether the ACME$M_UCS2_4 function code modifier was specified on the call to $ACM[W].

ACME$_NEW_PASSWORD_1

The ACME$_NEW_PASSWORD_1 item code is an input item code. It specifies the new primary password for a password change operation. The buffer must contain a password string. The case of this string will be preserved in delivery to ACME agents. Each ACME agent has its own policy regarding whether password strings are treated in a case sensitive or a case-insensitive manner.

This item code might be requested in a dialogue step.

ACME$_NEW_PASSWORD_2

The ACME$_NEW_PASSWORD_2 item code is an input item code. It specifies the new secondary password for a password change operation. The buffer must contain a password string. The case of this string will be preserved in delivery to ACME agents. Each ACME agent has its own policy regarding whether password strings are treated in a case sensitive or a case-insensitive manner.

This item code might be requested in a dialogue step.

ACME$_NEW_PASSWORD_FLAGS

The ACME$_NEW_PASSWORD_FLAGS item code is an input item code. It requests which passwords should be explicitly updated. The buffer must contain a longword bit vector of applicable flags.

The $ACMEDEF macros defines the following symbols for the valid flags:

ACMEPWDFLG$V_SYSTEM
ACMEPWDFLG$V_PASSWORD_1
ACMEPWDFLG$V_PASSWORD_2

ACME$_NEW_PASSWORD_SYSTEM

The ACME$_NEW_PASSWORD_SYSTEM item code is an input item code. It specifies the new system password for a password change operation. The buffer must contain a case-insensitive password string.

This item code might be requested in a dialogue step.

ACME$_NULL

The ACME$_NULL item code indicates that the current item list entry should be ignored.

ACME$_PASSWORD_1

The ACME$_PASSWORD_1 item code is an input item code. It specifies the primary password applicable to the requested operation. The buffer must contain a password string. The case of this string will be preserved in delivery to ACME agents. Each ACME agent has its own policy regarding whether password strings are treated in a case sensitive or a case-insensitive manner.

This item code might be requested in a dialogue step.

ACME$_PASSWORD_2

The ACME$_PASSWORD_2 item code is an input item code. It specifies the secondary password applicable to the requested operation. The buffer must contain a password string. The case of this string will be preserved in delivery to ACME agents. Each ACME agent has its own policy regarding whether password strings are treated in a case sensitive or a case-insensitive manner.

This item code might be requested in a dialogue step.

ACME$_PASSWORD_SYSTEM

The ACME$_PASSWORD_SYSTEM item code is an input item code. It specifies the system password applicable to the requested operation. The buffer must contain a case-insensitive password string.

This item code might be requested in a dialogue step.

ACME$_PERSONA_HANDLE_IN

The ACME$_PERSONA_HANDLE_IN item code is an input item code. It specifies the persona to use as the basis for credential acquisition processing. The buffer must contain a longword value specifying a persona ID of an existing persona.

ACME$_PERSONA_HANDLE_OUT

The ACME$_PERSONA_HANDLE_OUT item code is an output item code. It specifies a buffer to receive the persona ID of the persona created or acted upon by credential acquisition processing. The buffer descriptor must specify a longword.

If no ACME$_PERSONA_HANDLE_OUT item is specified but function modifier ACME $M_ACQUIRE_CREDENTIALS is specified, a persona that is created can be located with the $PERSONA_FIND system service.

ACME$_PHASE_TRANSITION

The ACME$_PHASE_TRANSITION is used by LOGINOUT to convey synchronization information to the VMS ACME for support of backward compatible interfaces for LGI-callouts and DECwindows login.

Use of this item code is reserved to OpenVMS.

ACME$_PRINCIPAL_NAME_IN

The ACME$_PRINCIPAL_NAME_IN item code is an input item code. It specifies the name of the entity that is subject to authentication within the domain of interpretation to which it belongs. The buffer must contain a name string.

This item code might be requested in a dialogue step.

ACME$_PRINCIPAL_NAME_OUT

The ACME$_PRINCIPAL_NAME_OUT item code is an output item code. It specifies the buffer to receive the name of the entity that was authenticated by the authenticating domain of interpretation. This item code is useful when the principal name is not explicitly provided, such as during autologon style processing during which an ACME agent provides the principal name.

The maximum data returned for this item code is the number of characters represented by the symbol, ACME$K_MAXCHAR_PRINCIPAL_NAME, so a caller's buffer should be at least that long, with the number of bytes allocated dependent on whether the ACME$M_UCS2_4 function code modifier was specified on the call to $ACM[W].

ACME$_QUERY_DATA

The ACME$_QUERY_DATA item code is an output item code. It specifies the buffer to receive the data returned from the query operation relating to the corresponding ACME$_QUERY_TYPE item code.

The ACME$_QUERY_DATA item code requires that an ACME$_QUERY_TYPE item code immediately precede it in the item list.

ACME$_QUERY_KEY_TYPE

The ACME$_QUERY_KEY_TYPE item code is an input item code. It specifies the key type for establishing the context of a query operation. The key format is specific to the ACME agent to which the call is directed.

An ACME$_QUERY_KEY_TYPE item requires an ACME$_QUERY_KEY_VALUE item immediately following it in the item list.

ACME$_QUERY_KEY_VALUE

The ACME$_QUERY_KEY_VALUE item code is an input item code. It specifies the key data for establishing the context of a query operation.

An ACME$_QUERY_KEY_VALUE item requires that an ACME$_QUERY_KEY_TYPE item immediately precede it in the item list.

ACME$_QUERY_TYPE

The ACME$_QUERY_TYPE item code is an input item code. It specifies the data to be returned in the buffer described by the corresponding ACME$_QUERY_DATA item code.

The ACME$_QUERY_TYPE item code requires that an ACME$_QUERY_DATA item code immediately follow it in the item list.

ACME$_REMOTE_HOST_ADDRESS

The ACME$_REMOTE_HOST_ADDRESS item code is an input item code. It specifies the network address of the system from which the request originated. The buffer must contain a network address using the representation consistent with ACME$_REMOTE_HOST_ADDRESS_TYPE item code is specified.

ACME$_REMOTE_HOST_ADDRESS_TYPE

The ACME$_REMOTE_HOST_ADDRESS_TYPE item code is an input item code that specifies the representation of the ACME$_REMOTE_HOST_ADDRESS item code. The buffer must contain a longword value specifying the address type.

The $ACMEDEF macro defines the following symbols for the standard address types:

Symbol	Meaning
ACMEHAT$K_DECNET_IV	DECnet Phase IV
ACMEHAT$K_DECNET_OSI	DECnet OSI
ACMEHAT$K_IP_V4	Internet Protocol V4
ACMEHAT$K_IP_V6	Internet Protocol V6

ACME$_REMOTE_HOST_FULLNAME

The ACME$_REMOTE_HOST_FULLNAME item code is an input item code. It specifies the fully expanded name of the remote system from which the request originated. The buffer must contain a name string.

ACME$_REMOTE_HOST_NAME

The ACME$_REMOTE_HOST_NAME item code is an input item code. It specifies the name of the remote system from which the request originated. The buffer must contain a name string.

ACME$_REMOTE_USERNAME

The ACME$_REMOTE_USERNAME item code is an input item code. It specifies the name of the remote user on whose behalf the request is being initiated. The buffer must contain a name string.

ACME$_RESPONSE_DATA

The ACME$_RESPONSE_DATA item code is an input item code. It specifies the response data that was calculated using the challenge data.

Interpretation of this data is specific to a domain of interpretation. This item code may be requested in a dialogue step.

ACME$_SERVER_NAME_IN

Specifies the Event Server to which an Event should be directed. The meaning of this item is specific to the target domain of interpretation.

ACME$_SERVER_NAME_OUT

Reports the Event Server to which an Event was directed. The meaning of this item is specific to the target domain of interpretation.

ACME$_SERVICE_NAME

Indicates the client program making the call to $ACM. The buffer must contain the case-insensitive service name string. The default value is the current image name if the client program is an installed image.

Names beginning with *x-* are reserved for local use.

ACME$_TARGET_DOI_ID

Establishes the domain of interpretation within which nonquery operations are performed and the context within which ACME-specific items codes are interpreted.

This item code also has an effect on the parsing of the list of ACME-specific item codes and takes effect immediately. It is in effect until the next instance of code ACME$_CONTEXT_ACME_ID, code ACME$_CONTEXT_ACME_NAME, code ACME$_TARGET_DOI_ID, or code ACME $_TARGET_DOI_NAME. It also specifies which ACME is to be responsible for the authentication.

The buffer must contain a longword value specifying the agent ID of a domain of interpretation.

ACME$_TARGET_DOI_NAME

Establishes the domain of interpretation within which nonquery operations are performed and the context within which ACME-specific item codes are interpreted.

This item code also has an effect on the parsing of the list of ACME-specific item codes, and takes effect immediately. It is in effect until the next instance of code ACME$_CONTEXT_ACME_ID, code ACME$_CONTEXT_ACME_NAME, code ACME$_TARGET_DOI_ID, or code ACME $_TARGET_DOI_NAME. It also specifies which ACME is to be responsible for the authentication.

The buffer must contain the case-insensitive name string of a domain of interpretation.

ACME$_TIMEOUT_INTERVAL

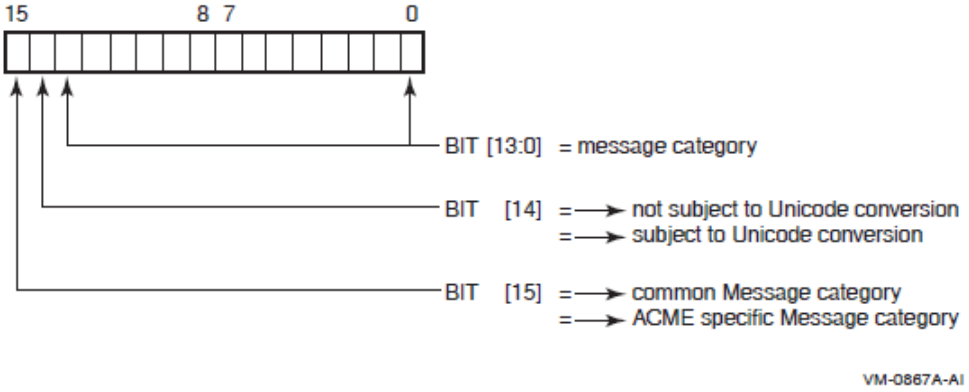
Specifies the number of seconds that must elapse before the current request times out. (See the ACME $M_TIMEOUT function modifier.)

Timeout interval values are specified in seconds and must be between 1 and 300 seconds. If an invalid value is specified, the service returns SS$_IVTIME.

The default timeout interval is 30 seconds. This value may be adjusted by defining the exec mode logical name ACME$TIMEOUT_DEFAULT in the LNM$SYSTEM_TABLE logical name table. This timeout is enforced for non-dialogue requests and for the first request in a sequence of dialogue calls. The default value for subsequent dialogue requests can be adjusted by defining the exec mode logical name ACME $DIALOGUE_TIMEOUT_DEFAULT in the LNM$SYSTEM_TABLE logical name table.

Unprivileged clients can specify only timeout interval values less than or equal to the default value. Values greater than the default are ignored. Output Message Categories This section describes the various output message categories supported by the $ACM service.

Message Types are 16-bit unsigned values, encoded as follows:



Function-Independent Common Output Message Categories

The following table lists the function-independent common output messages and their meanings:

Message Category	Meaning
ACMEMC$K_GENERAL	Specifies a general text message

Message Category	Meaning
ACMEMC$K_HEADER	Specifies a header text message
ACMEMC$K_TRAILER	Specifies a trailer text message
ACMEMC$K_SELECTION	Specifies an acceptable choices message
ACMEMC$K_DIALOGUE_ALERT	Specifies an advisory alert message

Authentication Common Output Message Categories

The following table lists the authentication common output message categories and their meanings:

Message Category	Meaning
ACMEMC$K_SYSTEM_ IDENTIFICATION	Specifies system identification text messages
ACMEMC$K_SYSTEM_NOTICES	Specifies system notices
ACMEMC$K_WELCOME_NOTICES	Specifies welcome notices
ACMEMC$K_LOGON_NOTICES	Specifies logon notices
ACMEMC$K_PASSWORD_ NOTICES	Specifies password notices
ACMEMC$K_MAIL_NOTICES	Specifies MAIL notices

Description

The Authentication and Credential Management ($ACM) service presents a unified interface for performing authentication-related operations in a manner independent of applicable policy.

On a given OpenVMS system, multiple authentication policies may be applicable. The system may be configured to augment the native (local OpenVMS) policy with alternatives pertaining to external environments, such as LAN Manager. Each policy, together with the operating environment to which it pertains, constitutes a domain of interpretation. Within a given domain, any entity, such as a user, that is subject to the applicable authentication policy, is referred to as a principal.

The $ACM service can be used to authenticate a principal, initiate a password change request on behalf of a principal, query information about a particular domain, or report event data within a particular domain.

The $ACM service completes asynchronously; that is, it returns to the caller after queuing the request, without waiting for the operation to complete.

To synchronize completion of an operation, use the Authentication and Credential Management and Wait ($ACMW) service. The $ACMW service is identical to $ACM in every way except that $ACMW returns to the caller after the operation has completed.

Modes of Operation

The typical authentication policy employs the traditional reusable password; however, various alternative mechanisms exist for forming stronger policies. Some of these mechanisms, such as challenge-response, require interaction. The $ACM service is designed to accommodate these mechanisms.

The authentication and change_password functions are capable of operating in a dialogue (iterative) mode to support different types of interactive authentication mechanisms. The query, event, and free_context functions only support the nondialogue (noniterative) mode of operation.

Nondialogue (Noniterative) Mode

The default nature of the $ACM service is to operate in a noniterative mode. All information needed to complete the request must be provided in a single call; otherwise, the request ultimately fails. This requires the caller to know beforehand what information is required to complete the request.

The following list summarizes the control flow for a typical nondialogue mode authentication request. For simplicity, the scenario assumes a single domain of interpretation with a traditional user name and password policy. Also, error processing is ignored.

1. The caller of $ACM prompts the user for the principal name and password, builds an item list specifying the principal name and password, and then calls $ACM specifying the authenticate principal function, the item list with the principal name and password, and a zero address for the *context* argument.

2. $ACM processes the request and ultimately returns control to the caller with the final status for the operation.

Dialogue (Iterative) Mode

The caller can use the interactive capabilities of the $ACM service for authentication and password change operations by specifying the ACME$_DIALOGUE_SUPPORT item code and a valid *context* argument. In this mode, ACME agents can request additional information from the caller to complete the request. In effect, the $ACM service is called in an iterative fashion until all information required to complete the request has been provided. The sequence of calls are linked together by passing the *context* argument returned in one call back in the next call.

In this scenario, when an ACME agent requires additional information, it builds an item set that describes the nature of the information. The item set is passed back to the caller in the communications buffer (see the description for the *context* argument regarding the format of the communications buffer) and the service returns with the ACME$_OPINCOMPL status. The caller processes each item set entry, gathers the requested information, and then passes it back to the ACME agent using the *itmlst* argument in the next call. The sequence continues until the call returns with a status code other than ACME$_OPINCOMPL.

The following list summarizes the control flow for a typical dialogue-mode authentication sequence. For simplicity, the scenario assumes a single domain of interpretation with a traditional user name and password policy. Also, error processing is ignored.

1. Make an initial call to $ACM specifying the authenticate principal function code, an item list that merely contains the ACME$_DIALOGUE_SUPPORT item code, and a *context* argument that has been initialized to -1.

2. $ACM builds a communications buffer containing an item set in the buffer requesting the principal name (user name), sets the *context* argument to reference the buffer, and returns control to the caller with a status code of ACME$_OPINCOMPL.

3. The caller processes the item set, prompts for the principal name, builds an item list specifying the principal name, and then calls $ACM again specifying the authenticate principal function as before, the item list with the principal name, and a *context* argument that contains the buffer address returned in the previous call.

4. $ACM validates the *context* argument, processes the username then builds another communications buffer to contain an item set list requesting the password, sets the *context*

argument to reference the buffer, and returns control to the caller again with a status code of ACME $_OPINCOMPL.

5. The caller processes the item set, prompts for the password, builds an item list specifying the password, and then calls $ACM again specifying the authenticate principal functions as before, the item list with the password, and a `context` argument that contains the buffer address returned in the previous call.

6. $ACM validates the `context` argument again, clears it, and then completes the processing of the request, now that it has all the necessary information, and ultimately returns control to the caller with the final status for the operation.

Unprivileged callers (those running in user mode and not possessing SECURITY privilege) are limited by the number of iterative requests they can make in a dialogue sequence of calls. The default is set at 26 dialogue requests. The default can be overridden by defining the exec mode logical name ACME $DIALOGUE_ITERATIVE_LIMIT in the LNM$SYSTEM_TABLE logical name table. Valid values are 1 through 100.

Determining an ACME Name Based on an ACME ID

The identity of the ACME that supplied the ACME$L_ACME_STATUS contents is indicated in the ACMEID$V_ACME_NUM subfield of the ACMESB$L_ACME_ID field. This value is consistent for the duration of one boot of the system, but may have a different value on the next boot. The name of a particular ACME agent can be determined from the ACME ID by calling $ACM with function code ACME$_FC_QUERY and the following item list entries:

- Special ACM Dispatch query—ID value zero:

  ITMCOD = ACME$_TARGET_DOI_ID
  BUFSIZ = 4
  BUFADR = Address of longword containing 0

- Query ACME name based on ACME ID:

  ITMCOD = ACME$_QUERY_KEY_TYPE
  BUFSIZ = 4
  BUFADR = Address of longword containing ACME$K_QUERY_ACME_ID

- Specify ACME ID value:

  ITMCOD = ACME$_QUERY_KEY_VALUE
  BUFSIZ = 4
  BUFADR = Address of longword containing the ACME_ID

- Specify ACME name for the return value:

  ITMCOD = ACME$_QUERY_TYPE
  BUFSIZ = 4
  BUFADR = Address of longword containing ACME$K_QUERY_ACME_NAME

- Specify the output buffer:

  ITMCOD = ACME$_QUERY_DATA
  BUFSIZ = ACME$K_MAXCHAR_DOI_NAME or (ACME$K_MAXCHAR_DOI_NAME*4) depending on whether function modifier ACME$M_UCS2_4 has been specified

BUFADR = Address of buffer large enough to hold ACME$K_MAXCHAR_DOI_NAME bytes or (ACME$K_MAXCHAR_DOI_NAME*4) depending on whether function modifier ACME $M_UCS2_4 has been specified

Privileges and Restrictions

The $ACM service constitutes a trusted interface. It restricts operations that override the security policy applicable to a given domain of interpretation to those callers who are suitably privileged. The status returned in the ACMESB$L_STATUS field of the ACM Status Block for a failed authentication operation is typically nonspecific, so as not to reveal sensitive information to untrusted callers.

If the caller has the SECURITY privilege, the ACMESB$L_SECONDARY_STATUS field of the ACM Status Block may contain a detailed status that more accurately reflects the actual nature of the failure.

To specify the following function modifiers, the caller must have the SECURITY privilege:

ACME$M_NOAUDIT
ACME$M_NOAUTHORIZATION
ACME$M_FOREIGN_POLICY_HINTS

To specify the following function modifier, the caller must have the IMPERSONATE privilege:

ACME$M_OVERRIDE_MAPPING

To specify the following item code, the caller must have the SECURITY privilege:

ACME$_NEW_PASSWORD_SYSTEM

To specify the following item codes, the caller must have the IMPERSONATE privilege:

ACME$_ACCESS_PORT
ACME$_CHALLENGE_DATA
ACME$_REMOTE_HOST_ADDRESS
ACME$_REMOTE_HOST_ADDRESS_TYPE
ACME$_REMOTE_HOST_FULLNAME
ACME$_REMOTE_HOST_NAME
ACME$_REMOTE_USERNAME
ACME$_SERVICE_NAME

Condition Values Returned

SS$_NORMAL

The service completed successfully.

SS$_ACCVIO

The item list or an input buffer cannot be read in the access mode of the caller; or an output buffer, a return length buffer, or the I/O status block cannot be written in the access mode of the caller.

SS$_ARG_GTR_32_BITS

A 64-bit address was passed in a context requiring a 32-bit address.

SS$_BADBUFADR

The buffer address associated with an entry in the item list is inappropriate in the context of the call. The address may be invalid (for example, 0).

SS$_BADBUFLEN

The buffer length associated with an entry in the item list is inappropriate in the context of the call. The length may be invalid (for example, 0) or outside the range of acceptable values.

SS$_BADCHAIN

A chained item list is inaccessible, or the chain is circular.

SS$_BADCONTEXT

The *context* argument does not specify a valid context buffer.

SS$_BADITMCOD

A specified item code is invalid or out-of-range.

SS$_BADPARAM

The item list contains an invalid item code.

SS$_BADRETLEN

The return length address associated with an entry in the item list is inappropriate in the context of the call. The address may be invalid (for example, 0).

SS$_EXASTLM

The *astadr* argument was specified and the process has exceeded its ASTLM quota.

SS$_EXQUOTA

A process quota was exceeded.

SS$_ILLEFC

The *efn* argument specifies an illegal event flag number.

SS$_ILLMODIFIER

The *func* argument specifies function modifiers that are inappropriate in the context of the call.

SS$_INSFMEM

Insufficient space exists for completing the request.

SS$_IVTIME

An invalid value was specified for the ACME$_TIMEOUT_INTERVAL item code.

SS$_NOEXTAUTH

External authentication is not available.

SS$_NOPRIV

The caller does not have the necessary privileges to complete the requested operation.

SS$_TOOMUCHDATA

The request size exceeds $ACM messaging constraints.

SS$_UNASEFC

The *efn* argument specifies an unassociated event flag cluster.

SS$_UNSUPPORTED

The *func* argument specifies an unsupported function.

Condition Values Returned in the ACM Status Block

ACME$_NORMAL

The service completed successfully.

ACME$_ACCOUNTLOCK

The account associated with specified principal name is disabled.

ACME$_AUTHFAILURE

Authorization failed.

ACME$_BUFFEROVF

An output item returned by the service is larger than the user buffer provided to receive the item; the item is truncated.

ACME$_DOIUNAVAILABLE

The specified domain of interpretation is not processing requests.

ACME$_INCONSTATE

The ACME server detected an internal consistency error.

ACME$_INSFDIALSUPPORT

Caller dialogue capabilities specified with the ACME$DIALOGUE_SUPPORT item code are inadequate to meet the needs of one or more ACME agents.

ACME$_INTRUDER

A record matching the request was found in the intrusion database.

ACME$_INVALIDCTX

The *context* argument is not consistent with the *itmlst* argument.

ACME$_INVALIDPWD

The specified password is invalid.

ACME$_INVITMSEQ

The service encountered a query type or query key item code without a corresponding query data or query key value item code.

ACME$_INVMAPPING

The OpenVMS user name to which the principal name was mapped is invalid.

ACME$_INVNEWPWD

The new password provided during a change password request does not pass qualification checks.

ACME$_INVPERSONA

The persona handle specified by the `itmlst` argument is invalid.

ACME$_INVREQUEST

A parameter is invalid in the context of the request. This error code is returned when the caller either defaults or specifies ACME$_BATCH or the value zero (0) for ACME$_LOGON_TYPE.

ACME$_MAPCONFLICT

An attempt was made to merge credentials for a principal name, which maps to an OpenVMS user name that differs from the one associated with existing credentials.

ACME$_NOACMECTX

The service encountered an ACME-specific item code when no ACME context had been established.

ACME$_NOCREDENTIALS

The ACME agent did not issue any credentials.

ACME$_NOEXTAUTH

The specified principal name cannot be authenticated externally.

ACME$_NOPRIV

The caller does not have the necessary privileges to complete the requested operation.

ACME$_NOSUCHDOI

The specified domain of interpretation does not exist.

ACME$_NOSUCHUSER

The specified principal name does not exist.

ACME$_NOTARGETCRED

The persona does not contain credentials for the specified domain of interpretation.

ACME$_NOTAUTHORIZED

Authorization failed due to account restrictions.

ACME$_OPINCOMPL

Interaction is required to complete the request. The context buffer contains information describing how to proceed.

ACME$_PWDEXPIRED

The password provided during an authentication request has expired and a new password is required to complete the request.

ACME$_TIMEOUT

The server did not respond within the designated time-out interval.

ACME$_UNSUPPORTED

The requested operation or an item code is not supported with the selected domain of interpretation.

Status Codes and Function Codes Table

Table 13 lists status codes and their function codes:

Table 13. Status Codes and Function Codes

Status Codes	Function Codes					
	Authenticate Principal	**Change Password**	**Event**	**Free Context**	**Query**	**Release Credentials**
ACME$_ ACCOUNTLOCK	◆	◆				
ACME$_ AUTHFAILURE	◆	◆				
ACME$_ BUFFEROVR	◆	◆				
ACME$_ DOIUNAVAILABLE	◆	◆	◆	◆		
ACME$_ INCONSTATE	◆	◆	◆	◆	◆	◆
ACME$_ INSFDIALSUPPORT	◆	◆				
ACME$_ INTRUDER	◆					
ACME$_ INVALIDCTX	◆	◆	◆			
ACME$_ INVALIDPWD	◆	◆				
ACME$_ INVITMSEQ					◆	
ACME$_ INVMAPPING	◆	◆				
ACME$_ INVNEWPWD	◆	◆				
ACME$_ INVPERSONA	◆					◆
ACME$_ INVREQUEST	◆	◆	◆	◆		
Key to Codes **♦—Permitted**						

Status Codes	Function Codes					
	Authenticate Principal	**Change Password**	**Event**	**Free Context**	**Query**	**Release Credentials**
ACME$_ MAPCONFLICT	◆	◆				
ACME$_ NOACMECTX	◆	◆	◆	◆	◆	◆
ACME$_ NOCREDENTIALS	◆					
ACME$_ NOEXTAUTH	◆	◆				
ACME$_ NOPRIV	◆	◆	◆	◆		◆
ACME$_ NORMAL	◆	◆	◆	◆	◆	◆
ACME$_ NOSUCHDOI	◆	◆	◆	◆		
ACME$_ NOSUCHUSER	◆	◆				
ACME$_ NOTARGETCRED						◆
ACME$_ NOTAUTHORIZED	◆					
ACME$_ OPINCOMPL	◆	◆				
ACME$_ PWDEXPIRED	◆					
ACME$_ TIMEOUT	◆	◆	◆		◆	
ACME$_ UNSUPPORTED	◆	◆	◆		◆	
Key to Codes **♦—Permitted**						

VMS ACME Use of Function Codes

The VMS ACME use of the Event function is reserved to OpenVMS.

The VMS ACME does not support the Query function.

VMS ACME-Specific Item Codes

This section describes the $ACM item codes that are ACME-specific for the VMS ACME.

Table 14 indicates which OpenVMS ACME-specific Item Codes are applicable to the various Function Codes:

Table 14. Function Codes and OpenVMS Specific Item Codes

Item Codes	Function Codes					
	Authenticate Principal	**Change Password**	**Event**	**Free Context**	**Query**	**Release Credentials**
ACMEVMS$_ AUTOLOGIN_ ALLOWED_FLAG						
ACMEVMS$_ CLASS_DAYS **(O)**						
ACMEVMS$_ CLASS_FLAGS **(O)**						
ACMEVMS$_ CLASS_NUMBER **(O)**						
ACMEVMS$_ CLASS_ PRIMEDAY_LIMIT **(O)**						
ACMEVMS$_ CLASS_ SECONDARY_LIMIT **(O)**						
ACMEVMS$_ CLASS_NAME **(U,O)**						
ACMEVMS$_ CONFIRM_ PASSWORD_1 **(U)**						
ACMEVMS$_ CONFIRM_ PASSWORD_2						

Key to Codes

♦—Permitted
IR—IMPERSONATE Privilege Required to override default values
SR—SECURITY Privilege Required
O—Output item code
U—Subject to Unicode Conversion
BC—Backward Compatibility—Reserved to OpenVMS support of historical interface

Item Codes	Function Codes					
	Authenticate Principal	**Change Password**	**Event**	**Free Context**	**Query**	**Release Credentials**
(U)						
ACMEVMS$_ CONFIRM_ PASSWORD_SYS **(U)**						
ACMEVMS$_ CREPRC_BASPRI **(O)**						
ACMEVMS$_ CREPRC_IMAGE **(O)**						
ACMEVMS$_ CREPRC_PRCNAM **(O)**						
ACMEVMS$_ CREPRC_PRVADR **(O)**						
ACMEVMS$_ CREPRC_QUOTA **(O)**						
ACMEVMS$_ CREPRC_UIC **(O)**						
ACMEVMS$_ GENPWD_COUNT						
ACMEVMS$_ GENPWD_ MANDATORY_FLAG						
ACMEVMS$_ GENPWD_ MAXLENGTH						

Key to Codes

♦—Permitted
IR—IMPERSONATE Privilege Required to override default values
SR—SECURITY Privilege Required
O—Output item code
U—Subject to Unicode Conversion
BC—Backward Compatibility—Reserved to OpenVMS support of historical interface

Item Codes	Function Codes					
	Authenticate Principal	**Change Password**	**Event**	**Free Context**	**Query**	**Release Credentials**
ACMEVMS$_ GENPWD_ MINLENGTH						
ACMEVMS$_OLD_ CONNECTION_FLAG	**BC**					
ACMEVMS$_OLD_ DECWINDOWS_FLAG	**BC**					
ACMEVMS$_OLD_ HASHED_ PASSWORD_1	**BC**					
ACMEVMS$_OLD_ HASHED_ PASSWORD_2	**BC**					
ACMEVMS$_OLD_ LGI_PHASE	**BC**					
ACMEVMS$_OLD_ LGI_STATUS	**BC**					
ACMEVMS$_OLD_ PROCESS_NAME	**BC**					
ACMEVMS$_UAI_* **(O)**						
ACMEVMS$_ LOGINOUT_ CLI_FLAG	**BC**					
ACMEVMS$_ LOGINOUT_ CREPRC_FLAGS	**BC**					
ACMEVMS$_ NET_PROXY	**BC**					
ACMEVMS$_ PREAUTHENTICATION_ FLAG	**IR**					
ACMEVMS$_ REQUESTOR_PID	**IR**	**IR**				
ACMEVMS$_	**IR**	**IR**				

Key to Codes

♦—Permitted
IR—IMPERSONATE Privilege Required to override default values
SR—SECURITY Privilege Required
O—Output item code
U—Subject to Unicode Conversion
BC—Backward Compatibility—Reserved to OpenVMS support of historical interface

Item Codes	Function Codes					
	Authenticate Principal	**Change Password**	**Event**	**Free Context**	**Query**	**Release Credentials**
REQUESTOR_UIC						
ACMEVMS$_ REQUESTOR_ USERNAME **(U)**	**IR**	**IR**				
ACMEVMS$_USES_ SYSTEM_PASSWORD	**SR**					
Key to Codes **♦—Permitted** **IR—IMPERSONATE Privilege Required to override default values** **SR—SECURITY Privilege Required** **O—Output item code** **U—Subject to Unicode Conversion** **BC—Backward Compatibility—Reserved to OpenVMS support of historical interface**						

SYS$CREPRC-Ready Item Codes

For users that need to create a process based on quotas and privileges from System User Authorization (SYSUAF) data, the following item codes return data in a form ready to be used in a call to SYS$CREPRC:

Item Code	Direction	Size	Data Provided
ACMEVMS$_CREPRC_BASPRI	Output	Longword	Base priority
ACMEVMS$_CREPRC_IMAGE	Output	String[1]	LOGINOUT
ACMEVMS$_CREPRC_PRCNAM	Output	String[1]	Null
ACMEVMS$_CREPRC_PRVADR	Output	Quadword	Privilege mask
ACMEVMS$_CREPRC_QUOTA	Output	Sequence-of-bytes	Quotas
ACMEVMS$_CREPRC_UIC	Output	Longword	UIC

[1]The caller must create a descriptor for this item returned as a string to pass it to SYS$CREPRC.

To receive results of these item codes without authentication requires you to use the ACMEVMS$_PREAUTHENTICATION_FLAG, which in turn requires the IMPERSONATE privilege. No additional privilege for these item codes is required.

ACMEVMS$_CREPRC_BASPRI

This output item code requests UAI data in a format suitable for passing to SYS$CREPRC.

This output item code request UAI data in a format suitable for passing to SYS$CREPRC.

ACMEVMS$_CREPRC_IMAGE

This output item code requests UAI data in a format suitable for passing to SYS$CREPRC. The $ACM[W] client is responsible for creating a descriptor for this string.

ACMEVMS$_CREPRC_PRCNAM

This output item code requests UAI data in a format suitable for passing to SYS$CREPRC. The $ACM[W] client is responsible for creating a descriptor for this string.

ACMEVMS$_CREPRC_PRVADR

This output item code requests UAI data in a format suitable for passing to SYS$CREPRC.

ACMEVMS$_CREPRC_QUOTA

This output item code requests UAI data in a format suitable for passing to SYS$CREPRC, regardless of what quota might be handled by this service in the future.

ACMEVMS$_CREPRC_UIC

This output item code requests UAI data in a format suitable for passing to SYS$CREPRC.

Generated Password Item Codes

Any generated password list is returned in the ACM Communications Buffer, which is accessed by the context parameter. The following item codes are used to affect this password list:

Item Code	Direction	Size	Data Provided
ACMEVMS$_GENPWD_COUNT	Input	Longword	Unsigned
ACMEVMS$_GENPWD_MANDATORY_ FLAG	Input	Longword	Boolean
ACMEVMS$_GENPWD_MAXLENGTH	Input	Longword	Unsigned
ACMEVMS$_GENPWD_MINLENGTH	Input	Longword	Unsigned

ACMEVMS$_GENPWD_COUNT

The value of this item code indicates the number of any passwords that are generated, regardless of whether generation is due to the SYS$CREPRC bit or the presence of the ACMEVMS$_GENPWD_MANDATORY_FLAG input item code.

ACMEVMS$_GENPWD_MANDATORY_FLAG

The caller of SYS$AMCW requests password generation if this item code is present. A value whose low bit is set indicates the caller wants to force the use of the generated passwords, with the VMS ACME rejecting any provided passwords that do not match a password on the list. A value whose low bit is clear indicates that the generated password list is just advisory, with no enforcement by the VMS ACME. However, VMS ACME might actually enforce generated passwords anyway, depending on the setting of the SYS$CREPRC bit within the UAI_FLAGS longword bit mask.

ACMEVMS$_GENPWD_MAXLENGTH

The value of this item code indicates the maximum length of any passwords that are generated, regardless of whether generation is due to the SYS$CREPRC bit or the presence of the ACMEVMS$_GENPWD_MANDATORY_FLAG input item code.

ACMEVMS$_GENPWD_MINLENGTH

The value of this item code indicates the minimum length of any passwords that are generated, regardless of whether generation is due to the SYS$CREPRC bit or the presence of the ACMEVMS$_GENPWD_MANDATORY_FLAG input item code.

Backward Compatibility Item Codes

The ACME-specific item codes that provide backward compatibility are listed in the following table:

Item Code	Direction	Size	Data Provided
ACMEVMS$_LOGINOUT_CLI_FLAGS	Input	Longword	Boolean
ACMEVMS$_LOGINOUT_CREPRC_FLAGS	Input	Longword	Bit mask
ACMEVMS$_OLD_CONNECTION_FLAG	Input	Longword	Boolean
ACMEVMS$_OLD_DECWINDOWS_FLAG	Input	Longword	Boolean
ACMEVMS$_OLD_HASHED_PASSWORD_1	Input	Variable	String
ACMEVMS$_OLD_HASHED_PASSWORD_2	Input	Variable	String
ACMEVMS$_OLD_LGI_PHASE	Input	Longword	Code value
ACMEVMS$_OLD_LGI_STATUS	Input	Longword	Message code
ACMEVMS$_OLD_PROCESS_NAME	Input	Variable	String

ACMEVMS$_LOGINOUT_CLI_FLAGS

This input item code supplies the traditional LOGINOUT qualifiers to the VMS ACME, including particularly the /LOCAL_PASSWORD and /CONNECT qualifiers. This item is never provided on an initial call. It is only provided in response to a dialogue step.

Use of this item code is reserved to LOGINOUT, and is enforced by the VMS ACME to prevent spoofing.

ACMEVMS$_LOGINOUT_CREPRC_FLAGS

This input item code provides the CTL$GL_CREPRC_FLAGS longword corresponding to the FLAGS argument used for process creation. The use of this item code is reserved to LOGINOUT and is enforced by the VMS ACME to prevent spoofing.

ACMEVMS$_OLD_CONNECTION_FLAG

This input item code is used by LOGINOUT to indicate to the VMS ACME that a terminal user logging in has chosen to connect to a disconnected process rather than proceed with a new process.

Use of this item code is reserved to LOGINOUT, and is enforced by the VMS ACME to prevent spoofing.

ACMEVMS$_OLD_DECWINDOWS_FLAG

This input item code indicates the old DECwindows callout interface is being used. Use of this item code is reserved to LOGINOUT, and is enforced by the VMS ACME to prevent spoofing.

ACMEVMS$_OLD_HASHED_PASSWORD_1

This input item code specifies a primary password in an alternate form. You can only use this item code when specifying a value of ACMEVMS$_ARGUS for ACME$_AUTH_MECHANISM.

To use this item code, you need the IMPERSONATE privilege.

ACMEVMS$_OLD_HASHED_PASSWORD_2

This input item code specifies a secondary password in an alternate form. You can only use this item code when specifying a value of ACMEVMS$_ARGUS for ACME$_AUTH_MECHANISM.

To use this item code, you need the IMPERSONATE privilege.

ACMEVMS$_OLD_LGI_PHASE

This input item code specifies the phase of the latest LGI-callout. It is used to provide processing equivalent so that when authentication is performed inside LOGINOUT, the following actions occur:

- Allows LGI$_SKIPRELATED from an LGI-callout routine to be honored by ACMEs.
- Allows the VMS ACME to update UAF$W_LOGFAILS and possibly UAF$V_DISACNT even for a failure declared by an LGI-callout routine.

Use of this item code is reserved to LOGINOUT and is enforced by the VMS ACME to prevent LGI$_SKIPRELATED spoofing. If you want to perform a similar function, you should write an ACME.

ACMEVMS$_OLD_LGI_STATUS

This input item code specifies the status returned from the latest LGI-callout. It is used to provide processing equivalent so that when authentication is performed inside LOGINOUT, the following actions occur.

- Allows LGI$_SKIPRELATED from an LGI-callout routine to be honored by ACMEs.
- Allows the VMS ACME to update UAF$W_LOGFAILS and possibly UAF$V_DISACNT even for a failure declared by an LGI-callout routine.

Use of this item code is reserved to LOGINOUT, enforced by the VMS ACME to prevent LGI$_SKIPRELATED spoofing. If you want to perform a similar function, you should write an ACME.

ACMEVMS$_OLD_PROCESS_NAME

This input item code is used by LOGINOUT to indicate to the VMS ACME the process name after it has attempted to change the process name to match the username.

Use of this item code is reserved to LOGINOUT, and is enforced by the VMS ACME to prevent spoofing.

User Authorization Information (UAI) Item Codes

he VMS ACME supports the UAI codes that return SYSUAF values. SYSUAF contents are required for authorization, initialization, and auditing. The UAI codes are transmitted to the VMS ACME as ACME-specific codes. For the definition of these item codes, see the SYS$GETUAI system service in the *VSI OpenVMS System Services Reference Manual: GETUTC-Z*.

When in dialogue mode and when you ask for the value in the fields, the VMS ACME returns the value from that of the previous login, that is, the login before the current login.

The following ACME UAI item codes are supported:

ACMEVMS$_UAI_ACCOUNTS
ACMEVMS$_UAI_ACCOUNT_LIM
ACMEVMS$_UAI_ASTLM
ACMEVMS$_UAI_AUDIT_FLAGS (*)
ACMEVMS$_UAI_BATCH_ACCESS_P
ACMEVMS$_UAI_BATCH_ACCESS_S
ACMEVMS$_UAI_BIOLM
ACMEVMS$_UAI_BYTLM
ACMEVMS$_UAI_CLITABLES
ACMEVMS$_UAI_CPUTIM
ACMEVMS$_UAI_DEF_CLASS
ACMEVMS$_UAI_DEFCLI
ACMEVMS$_UAI_DEFDEV
ACMEVMS$_UAI_DEFDIR
ACMEVMS$_UAI_DEF_PRIV
ACMEVMS$_UAI_DFWSCNT
ACMEVMS$_UAI_DIOLM
ACMEVMS$_UAI_DIALUP_ACCESS_P
ACMEVMS$_UAI_DIALUP_ACCESS_S
ACMEVMS$_UAI_ENCRYPT
ACMEVMS$_UAI_ENCRYPT2
ACMEVMS$_UAI_ENQLM
ACMEVMS$_UAI_EXPIRATION
ACMEVMS$_UAI_FILLM
ACMEVMS$_UAI_FLAGS
ACMEVMS$_UAI_GRP
ACMEVMS$_UAI_JTQUOTA
ACMEVMS$_UAI_LASTLOGIN_I
ACMEVMS$_UAI_LASTLOGIN_N
ACMEVMS$_UAI_LGICMD
ACMEVMS$_UAI_LOCAL_ACCESS_P
ACMEVMS$_UAI_LOCAL_ACCESS_S
ACMEVMS$_UAI_LOGFAILS
ACMEVMS$_UAI_MAXACCTJOBS
ACMEVMS$_UAI_MAX_CLASS
ACMEVMS$_UAI_MAXDETACH
ACMEVMS$_UAI_MAXJOBS
ACMEVMS$_UAI_MEM
ACMEVMS$_UAI_MIN_CLASS
ACMEVMS$_UAI_NETWORK_ACCESS_P
ACMEVMS$_UAI_NETWORK_ACCESS_S
ACMEVMS$_UAI_OWNER
ACMEVMS$_UAI_PARENT_ID
ACMEVMS$_UAI_PASSWORD (*)
ACMEVMS$_UAI_PASSWORD2 (*)
ACMEVMS$_UAI_PBYTLM
ACMEVMS$_UAI_PGFLQUOTA
ACMEVMS$_UAI_PRCCNT
ACMEVMS$_UAI_PRI
ACMEVMS$_UAI_PRIMEDAYS
ACMEVMS$_UAI_PRIV
ACMEVMS$_UAI_PROXYIES
ACMEVMS$_UAI_PROXY_LIM
ACMEVMS$_UAI_PWD
ACMEVMS$_UAI_PWD2
ACMEVMS$_UAI_PWD_DATE
ACMEVMS$_UAI_PWD2_DATE
ACMEVMS$_UAI_PWD_LENGTH
ACMEVMS$_UAI_PWD_LIFETIME
ACMEVMS$_UAI_QUEPRI
ACMEVMS$_UAI_REMOTE_ACCESS_P
ACMEVMS$_UAI_REMOTE_ACCESS_S
ACMEVMS$_UAI_RTYPE
ACMEVMS$_UAI_SALT
ACMEVMS$_UAI_SHRFILLM
ACMEVMS$_UAI_SUB_ID
ACMEVMS$_UAI_TQCNT
ACMEVMS$_UAI_UIC
ACMEVMS$_UAI_USER_DATA
ACMEVMS$_UAI_USRDATOFF
ACMEVMS$_UAI_USERNAME
ACMEVMS$_UAI_USERNAME_TAG
ACMEVMS$_UAI_JSVERSION
ACMEVMS$_UAI_WSQUOTA

* These items are defined for the following numeric calculations purposes because the base for the ACME-specific UAI item codes is ACMEVMS$K_UAI_BASE. ACMEVMS$K_UAI_BASE can be added to a UAI$_* code to produce the corresponding ACMEVMS$_UAI_* code.

Class Scheduling Item Codes

The following table lists class scheduling item codes:

Item Code	Direction	Size	Data Provided
ACMEVMS$_CLASS_DAYS	Output	Byte	Bit-mask
ACMEVMS$_CLASS_FLAGS	Output	Longword	Bit-mask
ACMEVMS$_CLASS_NAME	Output	Variable	String
ACMEVMS$_CLASS_NUMBER	Output	Word	Integer
ACMEVMS$_CLASS_PRIMEDAY_LIMIT	Output	24 bytes	Integer Array
ACMEVMS$_CLASS_SECONDAY_LIMIT	Output	24 bytes	Integer Array

ACMEVMS$_CLASS_DAYS

This item returns a 7-bit array, one for each day of the week starting with Monday as the low-order bit.

If a given bit is set, it means the corresponding day of the week is to be treated as a Secondary Day for purposes of class scheduling. If a given bit is clear, the corresponding day of the week is to be treated as a Primary Day for purposes of class scheduling. These designations are overridden if the $GETSYI item code SYI$_DAY_OVERRIDE is set.

This data is intended primarily for LOGINOUT in setting up any class scheduling required for a new process, although other callers of $ACM are free to request it for their own purposes.

Data returned for this item code is 1 byte long, so a caller's buffer should be at least that long.

ACMEVMS$_CLASS_FLAGS

This item code returns a 32-bit mask of flags used for class scheduling.

This data is intended primarily for LOGINOUT in setting up any class scheduling required for a new process, although other callers of $ACM are free to request it for their own purposes.

Data returned for this item code is 4 bytes long, so a caller's buffer should be at least that long.

ACMEVMS$_CLASS_NAME

This item code returns a string indicating the Class Name for class scheduling the VMS Username just authenticated.

This data is intended primarily for LOGINOUT in setting up any class scheduling required for a new process, although other callers of $ACM are free to request it for their own purposes.

Data returned for this item code is up to 16 characters long, so a caller's buffer should be at least that long, with the number of bytes allocated dependent on whether the ACME$M_UCS2_4 function code modifier was specified on the call to $ACM[W].

ACMEVMS$_CLASS_NUMBER

This item code returns the Class Number for class scheduling the VMS Username just authenticated. A Class Number of zero means no Class applies to this VMS Username.

This data is intended primarily for LOGINOUT in setting up any class scheduling required for a new process, although other callers of $ACM are free to request it for their own purposes.

Data returned for this item code is 2 bytes long, so a caller's buffer should be at least that long.

ACMEVMS$_CLASS_PRIMEDAY_LIMIT

This item code returns an array of 24 bytes, one for each hour of a Primary Day, each containing a number from 1 to 100 indicating the percentage of the overall system CPU time reserved for members of that class.

This data is intended primarily for LOGINOUT in setting up any class scheduling required for a new process, although other callers of $ACM are free to request it for their own purposes.

Data returned for this item code is 24 bytes long, so a caller's buffer should be at least that long.

ACMEVMS$_CLASS_SECONDAY_LIMIT

This item code returns an array of 24 bytes, one for each hour of a Secondary Day, each containing a number from 1 to 100 indicating the percentage of the overall system CPU time reserved for members of that class.

This data is intended primarily for LOGINOUT in setting up any class scheduling required for a new process, although other callers of $ACM are free to request it for their own purposes.

Data returned for this item code is 24 bytes long, so a caller's buffer should be at least that long.

Miscellaneous Item Codes

The following ACME-specific item codes cannot be classified into any of the previous categories:

Item Code	Direction	Size	Data Provided
ACMEVMS$_AUTOLOGIN_ALLOWED_ FLAG	Input	Longword	Boolean
ACMEVMS$_CONFIRM_PASSWORD_1	Input	Variable	String
ACMEVMS$_CONFIRM_PASSWORD_2	Input	Variable	String
ACMEVMS$_CONFIRM_PASSWORD_SYS	Input	Variable	String
ACMEVMS$_NET_PROXY	Input	Variable	String
ACMEVMS$_PREAUTHENTICATION_FLAG	Input	Longword	Boolean
ACMEVMS$_REQUESTOR_PID	Input	Longword	Hexadecimal
ACMEVMS$_REQUESTOR_UIC	Input	Longword	Hexadecimal
ACMEVMS$_REQUESTOR_USERNAME	Input	Variable	String
ACMEVMS$_USES_SYSTEM_PASSWORD	Input	Longword	Boolean

ACMEVMS$_AUTOLOGIN_ALLOWED_FLAG

This input item code specifies that a particular access port is of a type eligible for VMS Autologin. If the port is not specified in the Autologin file read by the VMS ACME, then this item code has no effect.

ACMEVMS$_CONFIRM_PASSWORD_1

The VMS ACME uses this input item code as a separate verification prompt when a new primary password is being specified. Use of a separate dialogue step rather than the verification method built into the Item Set definition allows some initial checking to be done for acceptability of the proposed password before the user is asked to type the password in again.

Some networked ACME agents are tied to network protocols that do not allow independent checking of the acceptability of a proposed password, so even when an item set with this item code is returned, the proposed password could be rejected later.

This item code might be requested in a dialogue step.

ACMEVMS$_CONFIRM_PASSWORD_2

The VMS ACME uses this input item code as a separate verification prompt when a new secondary password is being specified. Use of a separate dialogue step rather than the verification method built into the Item Set definition allows some initial checking to be done for acceptability of the proposed password before the user is asked to type the password again.

Some networked ACME agents are tied to network protocols that do not allow independent checking of the acceptability of a proposed password, so even when an item set with this item code is returned, the proposed password could be rejected later. Most networked ACME agents do not support secondary passwords, so after an item set with this item code has been returned, rejection later is unlikely, though possible.

This item code might be requested in a dialogue step.

ACMEVMS$_CONFIRM_PASSWORD_SYS

The VMS ACME uses this input item code as a separate verification prompt when a new system password is being specified. Use of a separate dialogue step rather than the verification method built into the Item Set definition allows full initial checking to be done for acceptability of the proposed system password before the user is asked to type the entire password in again.

This item code might be requested in a dialogue step.

ACMEVMS$_NET_PROXY

This input item code specifies the proxy user name for which a network login is to be processed, without authentication information, just as for a batch login or preauthenticated network login.

This item code requires the IMPERSONATE privilege.

ACMEVMS$_PREAUTHENTICATION_FLAG

This input item code specifies a login that is to be processed without authentication information, such as for a batch login. When first received by the VMS ACME, this item code causes the setting of the WQE_PREAUTHENTICATED flag in the Work Queue Entry Context, which is honored by all ACMEs.

To use this item code, you need the IMPERSONATE privilege.

ACMEVMS$_REQUESTOR_PID

This input item code specifies the Request or Processor ID for use by the VMS ACME in auditing and breakin detection. Combined with the codes ACMEVMS$_REQUESTOR_UIC and

ACMEVMS$_REQUESTOR_USERNAME, it is used when the process calling $ACM is not actually the process to which the authentication should be attributed. When first received by the VMS ACME, the value of this item is stored in the REQUESTOR_PID longword in the Request Context for later use. This item code is available to support LGI-callout operations and other callers to LGI$AUTHENTICATE_USER.

To use this item code, you need the IMPERSONATE privilege to guard against spoofing.

ACMEVMS$_REQUESTOR_UIC

This input item code specifies the Request or UIC for use by the VMS ACME in auditing and breakin detection. When first received by the VMS ACME, the value of this item is stored in the REQUESTOR_UIC longword in the Request Context for later use. This item code is available to support LGI-callout operations and other callers of LGI$AUTHENTICATE_USER.

This item allows the caller of $ACM to provide an accurate value because a call to SYS$GETJPI, based on the ACMEVMS$_REQUESTOR_PID ACME-specific item code value, might produce inaccurate results due to a subsequent assumption of a different persona in the request or process.

To use this item code, you need the IMPERSONATE privilege to guard against spoofing.

ACMEVMS$_REQUESTOR_USERNAME

This input item code specifies the Requestor Username for use by the VMS ACME in auditing and breakin detection. When first received by the VMS ACME, the value of this item is stored in the OWNER_USERNAME varying string descriptor in the Request Context for later use. This item code supports LGI-callout operations and other callers of LGI$AUTHENTICATE_USER.

This item allows the caller of $ACM to provide an accurate value because a call to SYS$GETJPI, based on the ACMEVMS$_REQUESTOR_PID item code value, might produce inaccurate results due to a subsequent assumption of a different persona in the requestor process.

To use this item code, you need the IMPERSONATE privilege to guard against spoofing.

ACMEVMS$_USES_SYSTEM_PASSWORD

This input item code specifies that a particular access port is enabled for use of the System Password. Other conditions, such as not having a System Password defined, may mean that no Item Set requesting a System Password is actually returned to the client. When first received by the VMS ACME, the value of this item is stored in the USES_SYSTEM_PASSWORD_FLAG boolean in the Request Context for later use.

To use this item code, you need the SECURITY privilege to guard against password guessing.

VMS ACME-Specific---Output Message Categories

The following table lists the output message categories specific to the VMS ACME and their meanings:

Message Category	Meaning
ACMEVMS$K_OLD_AUTH_FLAGS	Password requirement flags
ACMEVMS$K_OLD_DECW_PWD_EXP_1	Binary expiration warning
ACMEVMS$K_OLD_DECW_PWD_EXP_2	Binary expiration warning
ACMEVMS$K_OLD_DECW_PWD_QUALITY	Binary password quality status

Message Category	Meaning
ACMEVMS$K_OLD_SYSUAF_070	Authorization record
ACMEVMS$K_OLD_TERMINAL_CONNECT	Advance notice of authentication

These categories appear in output item set entries and are specially interpreted by LOGINOUT.

These categories are restricted to the LOGINOUT client since they exist only for supporting the old style DECwindows and LGI-callouts.

ACMEVMS$K_OLD_AUTH_FLAGS

This output message category provides a series of flags that are used to construct the Authentication Flags indicating what passwords are required for the mapped user name.

These flags are used directly for DECwindows V1.2-4 and earlier in LGI$AUTHENTICATE_USER, and for calculations that are passed to LGI-callouts in cell ICR_PWDCOUNT by INTERACT/ INTERACITVE_VALIDATION. It provides data abstraction from the specific SYSUAF fields.

The following table lists the flags, their bit number, and meaning:

Name	Bit Number	Meaning
OPENACCT	0	No password
PASSWORD_1	1	Primary password exists
PASSWORD_2	2	Secondary password exists
GENPWD	3	Passwords are not generated

This data is only supplied to LOGINOUT to support previous LGI-callout and DECwindows callout models. Other clients should follow the ACME model for user interaction.

This output item category provides a buffer that is to be filled with ACMEVMS$K_OLD_AUTH_FLAGS output message category data.

ACMEVMS$K_OLD_DECW_PWD_EXP_1

This output message category is a binary quadword indicating future password expiration.

It is provided only for compatibility with older versions of DECwindows. Its use is restricted to LOGINOUT, which is enforced by the VMS ACME.

ACMEVMS$K_OLD_DECW_PWD_EXP_2

This output message category is a binary quadword indicating future password expiration.

This data is provided only for compatibility with older versions of DECwindows. Its use is restricted to LOGINOUT, which is enforced by the VMS ACME.

ACMEVMS$K_OLD_DECW_PWD_QUALITY Output Message Category

This output message category is a binary longword indicating the failure category on password change.

This data is provided only for compatibility with older versions of DECwindows. Its use is restricted to LOGINOUT, which is enforced by the VMS ACME.

ACMEVMS$K_SYSUAF_070

This output message category provides a buffer that is to be filled with UAF070$ data. LOGINOUT now uses the $ACMW service, relying exclusively on UAI$_information rather than the direct manipulation of SYSUAF records. This output message category provides the same information as various UAI$_*xxxx* codes, but in the format of the OpenVMS V7.0 SYSUAF record. This item code allows LOGINOUT to continue supporting the LGI-callout interface. Your new software should use UAI$_*xxxx* item codes and avoid the ACMEVMS$K_SYSUAF_070 output message category.

This data is only supplied to LOGINOUT to support previous LGI-callout and DECwindows callout models. Other clients should follow the ACME model for user interaction.

ACMEVMS$K_OLD_TERMINAL_CONNECT

This output message category is passed from the VMS ACME to LOGINOUT to allow for changing the process name prior to auditing and to give an opportunity for the users to reconnect to any disconnected job they may have.

This data is provided only for compatibility with the historic behavior of LOGINOUT. Its use is restricted to LOGINOUT, which is enforced by the VMS ACME.

VMS ACME-Specific Authentication Mechanisms

ACMEVMS$K_AUTH_MECH_ARGUS

This authentication mechanism is used by the TNT Server component used for system management from Intel machines. Its use is restricted to that client, enforced by the VMS ACME.

$ACMW

Authentication and Credential Management — The $ACM service provides a common interface to all functions supported by the Authentication and Credentials Management (ACM) authority. The caller must specify the function code and any applicable function modifiers and item codes for the requested operation. The $ACM service completes asynchronously; for synchronous completion, use the $ACMW form of the service.

Format

```
SYS$ACMW [efn], func, [context], itmlst, acmsb, [astadr], [astprm]
```

C Prototype

```
int sys$acmw
   (unsigned int efn, unsigned int func, struct _acmecb *context, void
 *itmlst,
    struct _acmesb *acmsb, void (*astadr)(__unknown_params), int astprm);
```

Description

Beginning in OpenVMS Version 7.3-2, a number of functional changes were made to SYS$ACM[W]. In the following descriptions of these changes, **nonprivileged processes** refer to processes running in user mode that do not have SECURITY privilege.

These changes are the following:

- Timeout processing

  Timeout processing is now enforced for nonprivileged processes. Other processes can request time processing by specifying the ACME$M_TIMEOUT function modifier.

- Dialogue mode iteration limit

  Nonprivileged processes are now limited in the number of iterative requests they can make in a dialogue sequence of calls.

$ACQUIRE_GALAXY_LOCK (Alpha Only)

Acquire GALAXY Lock — Acquires ownership of an OpenVMS Galaxy lock. Note that this system service is supported only in an OpenVMS Alpha Galaxy environment. For more information about programming with OpenVMS Galaxy system services, see the *VSI OpenVMS Alpha Partitioning and Galaxy Guide*.

Format

```
SYS$ACQUIRE_GALAXY_LOCK handle ,timeout ,flags
```

C Prototype

```
int sys$acquire_galaxy_lock
   (unsigned __int64 lock_handle, unsigned int timeout, unsigned int
 flags);
```

Arguments

handle

OpenVMS usage:	galaxy lock handle
type:	quadword (unsigned)
access:	read
mechanism:	input by value

The 64-bit lock handle that identifies the lock to be acquired. This value is returned by SYS$CREATE_GALAXY_LOCK.

timeout

OpenVMS usage:	wait timeout
type:	longword (unsigned)
access:	read
mechanism:	input by value

The 32-bit wait or spin timeout specified in 10 microsecond units. If not specified, defaults to 10 microseconds.

flags

OpenVMS usage:	bit mask

type:	longword (unsigned)
access:	read
mechanism:	input by value

Control flags defined by the GLOCKDEF macro: GLOCK$C_NOBREAK, GLOCK$C_NOSPIN, and GLOCK$C_NOWAIT.

Description

This service is used to acquire ownership of an OpenVMS Galaxy lock. If the lock is free, the caller becomes the owner and control returns immediately. If the lock is owned, based on the input flags and the timeout value, either the caller will wait or an error will be returned.

The default behavior when an attempt is made to acquire a lock that is owned, is to spin for 10 microseconds and then to wait. If a wait timeout value was specified in the call, it is used. Otherwise the timeout value set in the lock by $CREATE_GALAXY_LOCK will be used. This behavior can be changed with the input flags.

If just GLOCK$C_NOSPIN is specified and the lock is owned, no spin will be done.

If just GLOCK$C_NOWAIT is specified and the lock is owned, the caller will only spin on the lock. If a timeout value is specified in the call, it is used as the spin time. Otherwise, the caller will spin for 10 microseconds. If the lock does not become available during the spin, the lock is not acquired and SS$_NOWAIT is returned.

If both GLOCK$C_NOSPIN and GLOCK$C_NOWAIT are specified and the lock is owned, control returns immediately. The lock is not acquired and SS$_NOWAIT is returned.

Due to system events such an OpenVMS Galaxy instance shutting down, a lock may become owned by a non-existent entity. If this occurs, the default behavior of $ACQUIRE_GALAXY_LOCK is to break the old lock ownership. The caller becomes the new owner and the service returns SS$_BROKEN. If GLOCK$C_NOBREAK is specified, $ACQUIRE_GALAXY_LOCK will not break the lock ownership and returns SS$_NOBREAK.

Required Access or Privileges

Write access to OpenVMS Galaxy lock table contains lock to acquire.

Required Quota

None

Related Services

$CREATE_GALAXY_LOCK, $CREATE_GALAXY_LOCK_TABLE, $DELETE_GALAXY_LOCK, $DELETE_GALAXY_LOCK_TABLE, $GET_GALAXY_LOCK_INFO, $GET_GALAXY_LOCK_SIZE, $RELEASE_GALAXY_LOCK

Condition Values Returned

SS$_NORMAL

Normal completion.

SS$_BADPARAM

Bad parameter value.

SS$_BROKEN

Lock acquired after lock ownership was broken.

SS$_IVLOCKID

Invalid lock id.

SS$_IVLOCKOP

Invalid lock operation.

SS$_IVLOCKTBL

Invalid lock table.

SS$_LOCK_TIMEOUT

Failed to acquire lock; request has timed out.

SS$_NOBREAK

Failed to acquire lock; lock ownership is broken.

SS$_NOWAIT

Failed to acquire lock; NOWAIT was specified.

$ADD_BRANCH

Add Branch — Authorizes a new branch to be added to a transaction.

Format

```
SYS$ADD_BRANCH
   [efn] ,[flags] ,iosb ,[astadr] ,[astprm] ,tid ,tm_name ,bid
```

C Prototype

```
int sys$add_branch
   (unsigned int efn, unsigned int flags, struct _iosb *iosb,
    void (*astadr)(__unknown_params), int astprm, unsigned int tid [4],
    void *tmname, unsigned int bid [4]);
```

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 used.

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 the option flag, which is described in Table 15. All undefined bits must be 0. If this argument is omitted, no flags are used.

Table 15. $ADD_BRANCH Option Flag

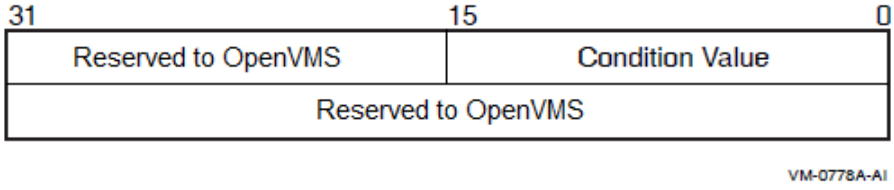
Flag Name	Description
DDTM$M_SYNC	Specifies successful synchronous completion by returning SS$_SYNCH. When SS$_SYNCH is returned, the AST routine is not called, the event flag is not set, and the I/O status block is not filled in.

iosb

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

The I/O status block in which the completion status of the service is returned as a condition value. See the section called “Condition Values Returned”.

The following diagram shows the structure of the I/O status block:



astadr

OpenVMS usage:	ast_procedure
type:	procedure entry mask
access:	call without stack unwinding

mechanism:	by reference

The AST routine executed when the service completes, if SS$_NORMAL is returned in R0. The `astadr` argument is the address of the entry mask of this routine. The routine is executed in the same access mode as that of the caller of the $ADD_BRANCH service.

astprm

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

The AST parameter that is passed to the AST routine specified by the `astadr` argument.

tid

OpenVMS usage:	trans_id
type:	octaword (unsigned)
access:	read only
mechanism:	by reference

The identifier (TID) of the transaction for which a new branch is to be authorized.

tm_name

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

The name of the node on which the new branch is running. Note that this cannot be a cluster alias.

To ensure smooth operation in a mixed-network environment, refer to the chapter entitled Managing DECdtm Services in the *VSI OpenVMS System Manager's Manual*, for information on defining node names.

bid

OpenVMS usage:	branch_id
type:	octaword (unsigned)
access:	write only
mechanism:	by reference

An octaword in which the identifier (BID) of the new branch is returned. No other call to $ADD_BRANCH on any node ever returns the same BID value.

Description

The $ADD_BRANCH system service:

- Authorizes a new branch to be added to the specified transaction.
- Checks, if the *tm_name* argument specifies a remote node, that there is a communications link between the DECdtm transaction manager on that node and the DECdtm transaction manager on the local node.

The precondition for the successful completion of $ADD_BRANCH is that the calling process must contain at least one branch of the specified transaction.

$ADD_BRANCH may fail for several reasons, including:

- The precondition was not satisfied.
- An abort event has occurred for the transaction.
- A call to $END_TRANS to end the transaction is in progress and it is now too late to authorize a new branch for the transaction.
- The node specified by the *tm_name* argument was a remote node and a failure was detected by the IPC mechanism.

Postconditions on successful completion of $ADD_BRANCH are described in Table 16:

Table 16. Postconditions When$ADD_BRANCH Completes Successfully

Postcondition	Meaning
A new branch is authorized for the transaction and its identifier is returned.	The identifier (BID) of the new branch is returned in the octaword to which the *bid* argument points. $ADD_BRANCH uses the $CREATE_UID system service to generate the BID. No other call to $ADD_BRANCH or $CREATE_UID on any node ever returns the same BID value.
The transaction cannot commit until the new branch has been added to the transaction by a matching call to $START_BRANCH.	See the description of $START_BRANCH for the definition of a "matching" call to $START_BRANCH.

There is also a wait form of the service, $ADD_BRANCHW.

Required Privileges

None

Required Quotas

BYTLM, ASTLM

Related Services

$ABORT_TRANS, $ABORT_TRANSW, $ACK_EVENT, $ADD_BRANCHW, $CREATE_UID, $DECLARE_RM, $DECLARE_RMW, $END_BRANCH, $END_BRANCHW, $END_TRANS, $END_TRANSW, $FORGET_RM, $FORGET_RMW, $GETDTI, $GETDTIW, $GET_DEFAULT_TRANS, $JOIN_RM, $JOIN_RMW, $SETDTI, $SETDTIW, $SET_DEFAULT_TRANS, $SET_DEFAULT_TRANSW, $START_BRANCH,

$START_BRANCHW, $START_TRANS, $START_TRANSW, $TRANS_EVENT, $TRANS_EVENTW

Condition Values Returned

SS$_NORMAL

If returned in R0, the request was successfully queued. If 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 to the caller.

SS$_BADPARAM

The options flags were invalid.

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

A required argument was missing.

SS$_INSFMEM

There was insufficient system dynamic memory for the operation.

SS$_INVBUFLEN

The string passed in the *tm_name* argument was longer than 256 characters.

SS$_NOSUCHTID

The calling process did not contain any branches in the transaction.

SS$_WRONGSTATE

The transaction was in the wrong state for the attempted operation because either an abort event has occurred for the transaction, or a call to $END_TRANS to end the transaction is in progress and it is now too late to authorize new branches for the transaction.

Any IPC status

An error has occurred while attempting to communicate with the node specified by the *tm_name* argument. The set of IPC statuses includes the set of DECnet errors.

$ADD_BRANCHW

Add Branch and Wait — Authorizes a new branch to be added to a transaction. $ADD_BRANCHW always waits for the request to complete before returning to the caller. Other than this, it is identical to $ADD_BRANCH.

Format

```
SYS$ADD_BRANCHW
   [efn] ,[flags] ,iosb ,[astadr] ,[astprm] ,tid ,tm_name ,bid
```

C Prototype

```
int sys$add_branchw
   (unsigned int efn, unsigned int flags, struct _iosb *iosb,
    void (*astadr)(__unknown_params), int astprm, unsigned int tid [4],
    void *tmname, unsigned int bid [4]);
```

$ADD_HOLDER

Add Holder Record to Rights Database — Adds a specified holder record to a target identifier.

Format

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

C Prototype

```
int sys$add_holder
   (unsigned int id, struct _generic_64 *holder, unsigned int attrib);
```

Arguments

id

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

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

holder

OpenVMS usage:	rights_holder

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

Holder identifier that is granted access to the target identifier when $ADD_HOLDER completes execution. The *holder* argument is the address of a quadword data structure that consists of a longword containing the holder's UIC identifier followed by a longword containing a value of 0.

attrib

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

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

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

Bit Position	Meaning When Set
KGB$V_DYNAMIC	Allows holders of the identifier to remove it from or add it to the process rights database by using the DCL command SET RIGHTS_LIST.
KGB$V_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$V_RESOURCE	Allows holders of an identifier to charge disk space to the identifier. It is used only for file objects.
KGB$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 Add Holder Record to Rights Database service registers the specified user as a holder of the specified identifier with the rights database.

Required Access or Privileges

Write access to the rights database is required.

Required Quota

None

Related Services

$ADD_IDENT, $ASCTOID, $CREATE_RDB, $FIND_HELD, $FIND_HOLDER, $FINISH_RDB, $GRANTID, $IDTOASC, $MOD_HOLDER, $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$_DUPIDENT

The specified holder already exists in the rights database for this identifier.

SS$_INSFMEM

The process dynamic memory is insufficient for opening the rights database.

SS$_IVIDENT

The specified identifier or holder is of an invalid format, the specified holder is 0, or the specified identifier and holder are equal.

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 *VSI OpenVMS Record Management Services Reference Manual*.

$ADD_IDENT

Add Identifier to Rights Database — Adds the specified identifier to the rights database.

Format

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

Prototype

```
int sys$add_ident
   (void *name, unsigned int id, unsigned int attrib, unsigned int *resid);
```

Arguments

name

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

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

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

id

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

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

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

attrib

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

mechanism:	by value

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

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

Bit Position	Meaning When Set
KGB$V_DYNAMIC	Allows holders of the identifier to remove it from or add it to the process rights database by using the DCL command SET RIGHTS_LIST.
KGB$V_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$V_RESOURCE	Allows holders of an identifier to charge disk space to the identifier. It is used only for file objects.
KGB$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.

resid

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

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

Description

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

Required Access or Privileges

Write access to the rights database is required.

Required Quota

None

Related Services

$ADD_HOLDER, $ASCTOID, $CREATE_RDB, $FIND_HELD, $FIND_HOLDER, $FINISH_RDB, $GRANTID, $IDTOASC, $MOD_HOLDER, $MOD_IDENT, $REM_HOLDER, $REM_IDENT, $REVOKID

Condition Values Returned

SS$_NORMAL

The service completed successfully.

SS$_ACCVIO

The *name* argument cannot be read by the caller, or the *resid* argument cannot be written by the caller.

SS$_BADPARAM

The specified attributes contain invalid attribute flags.

SS$_DUPIDENT

The specified identifier already exists in the rights database.

SS$_DUPLNAM

The specified identifier name already exists in the rights database.

SS$_INSFMEM

The process dynamic memory is insufficient for opening the rights database.

SS$_IVIDENT

The format of the specified identifier is invalid.

SS$_NORIGHTSDB

The rights database does not exist.

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 *VSI OpenVMS Record Management Services Reference Manual*.

$ADD_PROXY

Add or Modify Proxy — Adds a new proxy to, or modifies an existing proxy in, the proxy database.

Format

```
SYS$ADD_PROXY rem_node ,rem_user ,local_user ,[flags]
```

Prototype

```
int sys$add_proxy
   (void *rem_node, void *rem_user, void  *local_user,
    unsigned int 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 added to or modified in the proxy database. 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. node names are converted to their DECnet for OpenVMS full name unless the PRX$M_BYPASS_EXPAND flag is set with the *flags* argument.

If you specify a single asterisk (*) for the *rem_node* argument, the user name specified by the *rem_user* argument on all nodes is served by the proxy.

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 added to or modified in the proxy database. 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.

If you specify a single asterisk (*) for the *rem_user* argument, all users from the node specified by the *rem_node* argument are served by the same user names specified by the *local_user* argument.

local_user

OpenVMS usage:	char_string

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

Local user name to add to the proxy record specified by the *rem_node* and *rem_user* arguments in the proxy database as either the default user or local user. The *local_user* argument is the address of a character-string descriptor pointing to the local user name.

A local user name consists of 1 to 32 alphanumeric characters, including dollar signs ($) and underscores (_). Any lowercase characters specified are automatically converted to uppercase.

The user name specified by the *local_user* argument must be a user name known to the local system.

If the PRX$M_DEFAULT flag is specified in the *flags* argument, the user name specified by the *local_user* argument will be added to the proxy record in the proxy database as the default user. If a default user already exists for the specified proxy record, the default user is placed into the proxy's local user list and is replaced by the user name specified by the *local_user* argument.

Proxy records can contain no more than 16 local users and 1 default user. To add multiple users to a single proxy, you must call this service once for each local user.

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 names.

Symbolic Name	Description
PRX$M_BYPASS_EXPAND	The service should not convert the node name specified in the *rem_node* argument to its corresponding DECnet for OpenVMS full name. If this flag is set, it is the caller's responsibility to ensure that the fully expanded node name is passed into the service.
PRX$M_DEFAULT	The user name specified by the *local_user* argument is the default user for the proxy. If this flag is not specified, the user name specified by the *local_user* argument is added to the proxy record's local user list.
PRX$M_IGNORE_RETURN	The service should not wait for a return status from the security server. No return status from the server's function will be returned to the caller.

Description

The Add Proxy service adds a new proxy to, or modifies an existing proxy in, the proxy database.

Required Access or Privileges

The caller must have either SYSPRV privilege or a UIC group less than or equal to the MAXSYSGRP system parameter.

Required Quota

None

Related Services

$DELETE_PROXY, $DISPLAY_PROXY, $VERIFY_PROXY

Condition Values Returned

SS$_NORMAL

The service completed successfully.

SS$_ACCVIO

The *rem_node*, *rem_user*, *local_user*, or *flags* argument cannot be read by the service.

SS$_BADPARAM

An invalid flag was specified in the *flags* argument.

SS$_BADBUFLEN

The length of the *rem_node*, *rem_user*, or *local_user* argument was out of range.

SS$_NOSYSPRV

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$_DUPLICATEUSER

The user name specified by the *local_user* argument already exists in the proxy record's local user list.

SECSRV$_PROXYEXISTS

The specified proxy already exists.

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.

SECSRV$_TOOMANYUSERS

The specified proxy already has 16 local users and cannot accommodate any more.

$ADJSTK

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

Format

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

C Prototype

```
int sys$adjstk (unsigned int acmode, short int adjust, void  *(*(newadr)));
```

Arguments

acmode

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

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

adjust

OpenVMS usage:	word_signed
type:	word (signed)
access:	read only

mechanism:	by value

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

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

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

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

newadr

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

Value that *adjust* is to adjust. The *newadr* argument is the address of this longword value.

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

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

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

Description

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

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

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

If the adjust argument specifies:	And the value specified by newadr is:	The stack pointer is:
A value	An address	Loaded with the specified address, adjusted by the specified value

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

Required Access or Privileges

None.

Required Quota

None.

Related Services

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

Condition Values Returned

SS$_NORMAL

The service completed successfully.

SS$_ACCVIO

The value specified by *newadr* or a portion of the new stack segment cannot be written by the caller.

SS$_NOPRIV

The specified access mode is equal to or more privileged than the calling access mode.

$ADJWSL

Adjust Working Set Limit — Adjusts a process's current working set limit by the specified number of pagelets (on Alpha or Integrity server systems) and returns the new value to the caller. The working set limit specifies the maximum number of process pagelets that can be resident in physical memory. On Alpha and Integrity server systems, this service accepts 64-bit addresses.

Format

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

C Prototype

```
int sys$adjwsl (int pagcnt, unsigned int *wsetlm);
```

Arguments

pagcnt

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

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

Note that, on Alpha and Integrity server systems, the specified value is rounded up to an even multiple of the CPU-specific page size.

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

wsetlm

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

Value of the working set limit, in pagelets , returned by $ADJWSL. The *wsetlm* argument is the 32- or 64-bit address of this longword value. The *wsetlm* argument receives the newly adjusted value if *pagcnt* is specified, and it receives the prior, unadjusted value if *pagcnt* is not specified.

Description

The Adjust Working Set Limit service adjusts a process's current working set limit by the specified number of pagelets (rounded up or down to a whole page count) and returns the new value to the caller. The working set limit specifies the maximum number of process pagelets that can be resident in physical memory.

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

Required Access or Privileges

None

Required Quota

The initial value of a process's working set limit is controlled by the working set default (WSDEFAULT) quota. The maximum value to which it can be increased is controlled by the working set extent

(WSEXTENT) quota; the minimum value to which it can be decreased is limited by the system parameter MINWSCNT.

Related Services

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

Condition Values Returned

SS$_NORMAL

The service completed successfully.

SS$_ACCVIO

The longword specified by *wsetlm* cannot be written by the caller.

$ALLOC

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

Format

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

C Prototype

```
int sys$alloc
   (void *devnam, unsigned short int *phylen,  void *phybuf,
    unsigned int acmode, unsigned int flags);
```

Arguments

devnam

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

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

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

phylen

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

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

phybuf

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

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

acmode

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

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

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

flags

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

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

This feature is available for the following mass storage devices:

RA60	RA80	RA81	RC25
RCF25	RK06	RK07	RL01

RL02	RM03	RM05	RM80
RP04	RP05	RP06	RP07
RX01	RX02	TA78	TA81
TS11	TU16	TU58	TU77
TU78	TU80	TU81	

Description

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

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

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

Required Access or Privileges

Read, write, or control access to the device is required.

Required Quota

None

Related Services

$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$_BUFFEROVF

The service completed successfully. The physical name returned overflowed the buffer provided, and was truncated.

SS$_DEVALRALLOC

The service completed successfully. The device was already allocated to the calling process.

SS$_ACCVIO

The device name string, string descriptor, or physical name buffer descriptor cannot be read by the caller, or the physical name buffer cannot be written by the caller.

SS$_DEVALLOC

The device is already allocated to another process, or an attempt to allocate an unmounted shareable device failed because other processes had channels assigned to the device.

SS$_DEVMOUNT

The specified device is currently mounted and cannot be allocated, or the device is a mailbox.

SS$_DEVOFFLINE

The specified device is marked off line.

SS$_IVDEVNAM

The device name string contains invalid characters, or no device name string was specified.

SS$_IVLOGNAM

The device name string has a length of 0 or has more than 63 characters.

SS$_IVSTSFLG

The bits set in the longword of status flags are invalid.

SS$_NODEVAVL

The specified device in a generic search exists but is allocated to another user.

SS$_NONLOCAL

The device is on a remote node.

SS$_NOPRIV

The requesting process attempted to allocate a spooled device and does not have the required privilege, or the device protection or access control list (or both) denies access.

SS$_NOSUCHDEV

The specified device does not exist in the host system. This error is usually the result of a typographical error.

SS$_TEMPLATEDEV

The process attempted to allocate a template device; a template device cannot be allocated.

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

$ASCEFC

Associate Common Event Flag Cluster — Associates a named common event flag cluster with a process to execute the current image and to be assigned a process-local cluster number for use with other event

flag services. If the named cluster does not exist but the process has suitable privilege, the service creates the cluster.

Format

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

C Prototype

```
int sys$ascefc (unsigned int efn, void *name, char prot,  char perm);
```

Arguments

efn

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

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

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

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

name

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

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

The character string descriptor can be 1 to 15 bytes in length, and each byte can be any 8-bit value.

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

You can specify any name from 1 to 43 characters. All processes mapping to the same global section must specify the same name. Note that the name is case sensitive.

Use of characters valid in logical names is strongly encouraged. Valid values include alphanumeric characters, the dollar sign ($), and the underscore (_). If the name string begins with an underscore (_),

the underscore is stripped and the resultant string is considered to be the actual name. Use of the colon (:) is not permitted.

Names are first subject to a logical name translation, after the application of the prefix GBL$ to the name. If the result translates, it is used as the name of the section. If the resulting name does not translate, the name specified by the caller is used as the name of the section.

Additional information on logical name translations and on section name processing is available in the *VSI OpenVMS Programming Concepts Manual*.

prot

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

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

The default value 0 specifies that any process with the same UIC group number as the creating process can access the event flag cluster. The value 1 specifies that only processes having the UIC of the creating process can access the event flag cluster.

When the `prot` argument is 1, all access to the Group category is denied.

The process must have associate access to access an existing common event flag cluster.

perm

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

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

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

Description

The Associate Common Event Flag Cluster service associates a named common event flag cluster with a process for the execution of the current image and to assign it a process-local cluster number for use with other event flag services. A process needs associate access to call the $ASCEFC service.

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

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

When a new cluster is created, a security profile is created with the process UIC as the owner of the common event flag cluster; the remaining characteristics are taken from the COMMON_EVENT_CLUSTER.DEFAULT template profile.

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

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

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

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

Required Access or Privileges

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

Required Quota

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

Related Services

$CLREF, $DACEFC, $DLCEFC, $READEF, $SETEF, $WAITFR, $WFLAND, $WFLOR

Condition Values Returned

SS$_NORMAL

The service completed successfully.

SS$_ACCVIO

The cluster name string or string descriptor cannot be read by the caller.

SS$_EXPORTQUOTA

The process has exceeded the number of event flag clusters with which processes on this port of the multiport (shared) memory can associate.

SS$_EXQUOTA

The process has exceeded its timer queue entry quota; this quota controls the creation of temporary common event flag clusters.

SS$_INSFMEM

The system dynamic memory is insufficient for completing the service.

SS$_ILLEFC

You specified an illegal event flag number. The cluster number must be in the range of event flags 64 through 127.

SS$_INTERLOCK

The bit map lock for allocating common event flag clusters from the specified shared memory is locked by another process.

SS$_IVLOGNAM

The cluster name string has a length of 0 or has more than 15 characters.

SS$_NOPRIV

The process does not have the privilege to create a permanent cluster; the process does not have the privilege to create a common event flag cluster in memory shared by multiple processors; or the protection applied to an existing cluster by its creator prohibits association.

SS$_NOSHMBLOCK

The common event flag cluster has no shared memory control block available.

$ASCTIM

Convert Binary Time to ASCII String — Converts an absolute or delta time from 64-bit system time format to an ASCII string. On Alpha and Integrity server systems, this service accepts 64-bit addresses.

Format

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

C Prototype

```
int sys$asctim
   (unsigned short int *timlen, void *timbuf, struct _generic_64 *timadr,
    char cvtflg);
```

Arguments

timlen

OpenVMS usage:	word_unsigned
type:	word (unsigned)
access:	write only

mechanism:	by 32- or 64-bit reference

Length (in bytes) of the ASCII string returned by $ASCTIM. The *timlen* argument is the 32- or 64-bit address of a word containing this length.

timbuf

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

Buffer into which $ASCTIM writes the ASCII string. The *timbuf* argument is the 32-bit address (on VAX systems) or the 32- or 64-bit address of a character string descriptor pointing to the buffer.

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

timadr

OpenVMS usage:	date_time
type:	quadword
access:	read only
mechanism:	by 32- or 64-bit reference

Time value that $ASCTIM is to convert. The *timadr* argument is the 32- or 64-bit address of this 64-bit time value. A positive time value represents an absolute time. A negative time value represents a delta time. If you specify a delta time, it must be less than 10,000 days.

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

cvtflg

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

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

Description

The Convert Binary Time to ASCII String service converts an absolute or delta time from 64-bit system time format to an ASCII string. The service executes at the access mode of the caller and does not check whether address arguments are accessible before it executes. Therefore, an access violation causes an

exception condition if the input time value cannot be read or the output buffer or buffer length cannot be written.

This service returns the SS$_INSFARG (insufficient arguments) condition value if one or both of the required arguments are not supplied.

The ASCII strings returned have the following formats:

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

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

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

Month abbreviations must be uppercase.

The hundredths-of-second field now represents a true fraction. For example, the string .1 represents ten-hundredths of a second (one-tenth of a second), and the string .01 represents one-hundredth of a second.

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

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

Time Value	**Buffer Length Specified**	**CVTFLG Argument**	**Information Returned**
Absolute	23	0	Date and time

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

Required Access or Privileges

None

Required Quota

None

Related Services

$BINTIM, $CANTIM, $CANWAK, $GETTIM, $GETTIM_PREC, $NUMTIM, $SCHDWK, $SETIME, $SETIMR

Condition Values Returned

SS$_NORMAL

The service completed successfully.

SS$_BUFFEROVF

The buffer length specified in the *timbuf* argument is too small.

SS$_INSFARG

Required argument is missing.

SS$_IVTIME

The specified delta time is equal to or greater than 10,000 days.

$ASCTOID

Translate Identifier Name to Identifier — Translates the specified identifier name into its binary identifier value. On Alpha and Integrity server systems, this service accepts 64-bit addresses.

Format

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

C Prototype

```
int sys$asctoid (void *name, unsigned int *id, unsigned int *attrib);
```

Arguments

name

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

Identifier name translated when $ASCTOID completes execution. The *name* argument is the 32- or 64-bit address of a character-string descriptor pointing to the identifier name.

id

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

Identifier value resulting when $ASCTOID completes execution. The *id* argument is the 32- or 64-bit address of a longword in which the identifier value is written.

attrib

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

Attributes associated with the identifier returned in *id* when $ASCTOID completes execution. The *attrib* argument is the 32- or 64-bit address of a longword containing a bit mask specifying the attributes.

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

Bit Position	Meaning When Set
KGB$V_DYNAMIC	Allows holders of the identifier to remove it from or add it to the process rights database by using the DCL command SET RIGHTS_LIST.
KGB$V_HOLDER_HIDDEN	Prevents someone from getting a list of users who hold an identifier, unless they own the identifier themselves. Special privilege is required to translate hidden names.
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

Bit Position	Meaning When Set
	from translating the identifier. Special privilege is required to translate hidden names.
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 access control entry (ACE) to the application images in the subsystem.

Description

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

Required Access or Privileges

None, unless the *id* is KGB$V_NAME_HIDDEN, in which case you must hold the *id* or have access to the rights database.

Required Quota

None

Related Services

$ADD_HOLDER, $ADD_IDENT, $CREATE_RDB, $FIND_HELD, $FIND_HOLDER, $FINISH_RDB, $GRANTID, $IDTOASC, $MOD_HOLDER, $MOD_IDENT, $REM_HOLDER, $REM_IDENT, $REVOKID

Condition Values Returned

SS$_NORMAL

The service completed successfully.

SS$_ACCVIO

The *name* argument cannot be read by the caller, or the *id* or *attrib* arguments cannot be written by the caller.

SS$_INSFMEM

The process dynamic memory is insufficient for opening the rights database.

SS$_IVIDENT

The format of the specified identifier is invalid.

SS$_NOSUCHID

The specified identifier name does not exist in the rights database, or the identifier is hidden and you do not have access to the rights database.

SS$_NORIGHTSDB

The rights database does not exist.

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 *VSI OpenVMS Record Management Services Reference Manual*.

$ASCUTC

Convert UTC to ASCII — Converts an absolute time from 128-bit UTC format to an ASCII string. On Alpha and Integrity server systems, this service accepts 64-bit addresses.

Format

```
SYS$ASCUTC [timlen] ,timbuf ,[utcadr] ,[cvtflg]
```

C Prototype

```
int sys$ascutc
   (unsigned short int *timlen, void *timbuf,  unsigned int *utcadr [4],
    char cvtflg);
```

Arguments

timlen

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

Length (in bytes) of the ASCII string returned by $ASCUTC. The *timlen* argument is the 32- or 64-bit address of a word containing this length.

timbuf

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

Buffer into which $ASCUTC writes the ASCII string. The *timbuf* argument is the 32- or 64-bit address of a character string descriptor pointing to the buffer. The buffer length specified in the *timbuf* argument, together with the *cvtflg* argument, controls what information is returned.

utcadr

OpenVMS usage:	coordinated universal time
type:	utc_date_time
access:	read only
mechanism:	by 32- or 64-bit reference

Time value that $ASCUTC is to convert. The *timadr* argument is the 32- or 64-bit address of this 128-bit time value. Relative times are not permitted. If the *timadr* argument is not specified, it defaults to 0 and $ASCUTC returns the current date and time.

cvtflg

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

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

Description

The Convert UTC to ASCII service converts an absolute time from 128-bit UTC format to an ASCII string. The service executes at the access mode of the caller and does not check whether address arguments are accessible before it executes. Therefore, an access violation causes an exception condition if the input time value cannot be read or the output buffer or buffer length cannot be written.

The $ASCUTC service uses the time zone differential factor encoded in the 128-bit UTC to convert the UTC to an ASCII string.

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

The ASCII strings returned have the following format:

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

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

Field	Length (Bytes)	Contents	Range of Values
dd	2	Day of month	1–31
–	1	Hyphen	Required syntax
mmm	3	Month	JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC
–	1	Hyphen	Required syntax

Field	Length (Bytes)	Contents	Range of Values
yyyy	4	Year	1858–9999
blank	n	Blank	Required syntax
hh	2	Hour	00–23
:	1	Colon	Required syntax
mm	2	Minutes	00–59
:	1	Colon	Required syntax
ss	2	Seconds	00–59
.	1	Period	Required syntax
cc	2	Hundredths-of-second	00–99

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

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

Required Access or Privileges

None

Required Quota

None

Related Services

$BINUTC, $GETUTC, $NUMUTC, $TIMCON

Condition Values Returned

SS_$NORMAL

The service completed successfully.

SS_$BUFFEROVF

The buffer length specified in the *timbuf* argument is too small.

SS_$INVTIME

The UTC time supplied is too small to be represented as a Smithsonian Time, or the UTC time is not valid.

$ASSIGN

Assign I/O Channel — Provides a process with an I/O channel so input/output operations can be performed on a device, or establishes a logical link with a remote node on a network. On Alpha and Integrity server systems, this service accepts 64-bit addresses.

Format

```
SYS$ASSIGN devnam ,chan ,[acmode] ,[mbxnam] ,[flags]
```

C Prototype

```
int sys$assign
   (void *devnam, unsigned short int *chan, unsigned int acmode,
    void *mbxnam,...);
```

Arguments

devnam

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

Name of the device to which $ASSIGN is to assign a channel. The *devnam* argument is the 32- or 64-bit address of a character string descriptor pointing to the device name string.

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

chan

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

Number of the channel that is assigned. The *chan* argument is the 32- or 64-bit address of a word into which $ASSIGN writes the channel number.

acmode

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

Access mode to be associated with the channel. The *acmode* argument specifies the access mode. The $PSLDEF macro defines the following symbols for the four access modes.

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

The specified access mode and the access mode of the caller are compared. The less privileged (but the higher numeric valued) of the two access modes becomes the access mode associated with the assigned channel. I/O operations on the channel can be performed only from equal and more privileged access modes. For more information, see the section on access modes in the *VSI OpenVMS Programming Concepts Manual*.

mbxnam

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

Logical name of the mailbox to be associated with the device. The *mbxnam* argument is the 32- or 64-bit address of a character string descriptor pointing to the logical name string.

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

You must specify the *mbxnam* argument when performing a nontransparent, task-to-task, network operation.

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

For unshareable, nonspooled devices, an implicit $ALLOCATE is done. This requires read, write, or control access to the device.

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

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

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

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

flags

OpenVMS usage:	mask_longword

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

An optional device-specific argument. The `flags` argument is a longword bit mask. For more information on the applicability of the `flags` argument for a particular device, see the *VSI OpenVMS I/O User's Reference Manual*.

Description

The Assign I/O Channel service provides a process with an I/O channel so input/output operations can be performed on a device. This service also establishes a logical link with a remote node on a network.

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

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

Required Access or Privileges

The calling process must have NETMBX privilege to perform network operations and system dynamic memory is required if the target device is on a remote system.

Note that you should use the SHARE privilege with caution. Applications, application protocols, and device drivers coded to expect only exclusive access can encounter unexpected and errant behavior when access to the device is unexpectedly shared. Unless the SHARE privilege is explicitly supported by the application, the application protocol, and the device driver, its use is generally discouraged. For additional information, see the *VSI OpenVMS Programming Concepts Manual*.

Required Quota

If the target of the assignment is on a remote node, the process needs sufficient buffer quota to allocate a network control block.

Related Services

$ALLOC, $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$_REMOTE

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

SS$_ABORT

A physical line went down during a network connect operation.

SS$_ACCVIO

The device or mailbox name string or string descriptor cannot be read by the caller, or the channel number cannot be written by the caller.

SS$_CONNECFAIL

For network operations, the connection to a network object timed out or failed.

SS$_DEVACTIVE

You specified a mailbox name, but a mailbox is already associated with the device.

SS$_DEVALLOC

The device is allocated to another process.

SS$_DEVNOTMBX

You specified a logical name for the associated mailbox, but the logical name refers to a device that is not a mailbox.

SS$_DEVOFFLINE

For network operations, the physical link is shutting down.

SS$_EXBYTLM

The process has exceeded the byte count quota.

SS$_EXQUOTA

The target of the assignment is on a remote node and the process has insufficient buffer quota to allocate a network control block.

SS$_FILALRACC

For network operations, a logical link already exists on the channel.

SS$_INSFMEM

The target of the assignment is on a remote node and there is insufficient system dynamic memory to complete the request.

SS$_INVLOGIN

For network operations, the access control information was found to be invalid at the remote node.

SS$_IVDEVNAM

No device name was specified, the logical name translation failed, or the device or mailbox name string contains invalid characters. If the device name is a target on a remote node, this status code indicates that the network connect block has an invalid format.

SS$_IVLOGNAM

The device or mailbox name string has a length of 0 or has more than 63 characters.

SS$_LINKEXIT

For network operations, the network partner task was started, but exited before confirming the logical link (that is, $ASSIGN to SYS$NET).

SS$_NOIOCHAN

No I/O channel is available for assignment.

SS$_NOLINKS

For network operations, no logical links are available. The maximum number of logical links as set for the Network Control Program (NCP) executor MAXIMUM LINKS parameter was exceeded.

SS$_NOPRIV

For network operations, the issuing task does not have the required privilege to perform network operations or to confirm the specified logical link.

SS$_NOSUCHDEV

The specified device or mailbox does not exist, or, for DECnet for OpenVMS operations, the network device driver is not loaded (for example, the DECnet for OpenVMS software is not currently running on the local node).

SS$_NOSUCHNODE

The specified network node is nonexistent or unavailable.

SS$_NOSUCHOBJ

For network operations, the network object number is unknown at the remote node; for a TASK= connect, the named DCL command procedure file cannot be found at the remote node.

SS$_NOSUCHUSER

For network operations, the remote node could not recognize the login information supplied with the connection request.

SS$_PROTOCOL

For network operations, a network protocol error occurred, most likely because of a network software error.

SS$_REJECT

The network connect was rejected by the network software or by the partner at the remote node, or the target image exited before the connect confirm could be issued.

SS$_REMRSRC

For network operations, the link could not be established because system resources at the remote node were insufficient.

SS$_SHUT

For network operations, the local or remote node is no longer accepting connections.

SS$_THIRDPARTY

For network operations, the logical link connection was terminated by a third party (for example, the system manager).

SS$_TOOMUCHDATA

For network operations, the task specified too much optional or interrupt data.

SS$_UNREACHABLE

For network operations, the remote node is currently unreachable.

$AUDIT_EVENT

Audit Event — Appends an event message to the system security audit log file or sends an alarm to a security operator terminal.

Format

```
SYS$AUDIT_EVENT [efn] ,[flags] ,itmlst ,[audsts] ,[astadr] ,[astprm]
```

C Prototype

```
int sys$audit_event
   (unsigned int efn, unsigned int flags,  void *itmlst,
    unsigned int *audsts, void (*astadr)(__unknown_params),  int 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 audit completes. The *efn* argument is a longword containing the number of the event flag; however, $AUDIT_EVENT uses only the low-order byte. If *efn* is not specified, event flag 0 is used.

Upon request initiation, $AUDIT_EVENT clears the specified event flag.

flags

OpenVMS usage:	mask_longword

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

Flags specifying options for the $AUDIT_EVENT system operation. The *flags* argument is a longword bit mask, where each bit corresponds to an option.

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

Symbolic Name	Description
NSA$M_ACL	Specifies an event generated by an Alarm ACE or Audit ACE. This flag is reserved to OpenVMS.
NSA$M_FLUSH	Specifies that all messages in the audit server buffer be written to the audit log file.
NSA$M_INTERNAL	Specifies that the $AUDIT_EVENT call originates in the context of a trusted computing base (TCB) component. The auditing components use this flag to indicate that internal auditing failures should result in a SECAUDTCB bugcheck. This flag is reserved to OpenVMS.
NSA$M_MANDATORY	Specifies that an audit is to be performed, regardless of system alarm and audit settings.
NSA$M_NOEVTCHECK	Specifies that an audit is to be performed, regardless of the system alarm or audit settings. This flag is similar to the NSA$M_MANDATORY bit but, unlike the NSA$M_MANDATORY bit, this flag is not reflected in the NSA$W_FLAGS field in the resulting audit record on disk.
NSA$M_SERVER	Indicates that the call originates in a TCB server process and that the event should be audited regardless of the state of a process-specific no-audit bit. Trusted servers use this flag to override the no-audit bit when they want to perform explicit auditing on behalf of a client process. This flag is reserved to OpenVMS.

itmlst

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

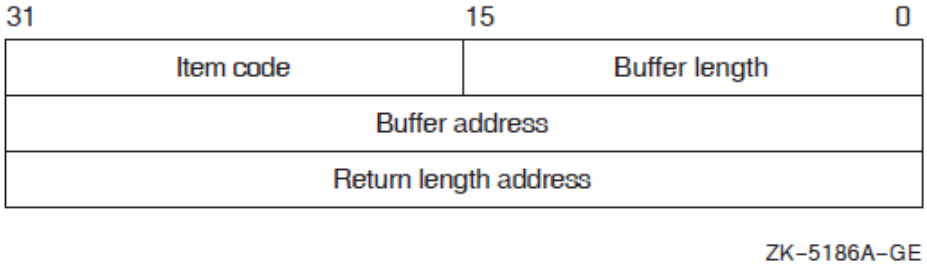
Item list specifying information to include in the audit record. The *itmlst* argument is the address of a list of item descriptors. The list of item descriptors is terminated by a longword of 0.

The item list for all calls to $AUDIT_EVENT must include the following item codes:

- NSA$_EVENT_TYPE (see Table 17)
- NSA$_EVENT_SUBTYPE (see Table 17)
- At least one of the NSA$_ALARM_NAME item code or the NSA$_AUDIT_NAME item code.

- If the event being reported is an object access (NSA$C_MSG_OBJ_ACCESS) or an object delete (NSA$C_MSG_OBJ_DELETE), the NSA$_FINAL_STATUS, NSA$_ACCESS_DESIRED, and NSA$_OBJECT_CLASS item codes must be specified.
- If the event being reported is an object create (NSA$C_MSG_OBJ_CREATE), the NSA$_FINAL_STATUS and NSA$_OBJECT_CLASS item codes must be specified.
- If the event being reported is a privilege audit (NSA$C_MSG_PRVAUD), the NSA$_PRIVS_USED or the NSA$_PRIVS_MISSING item code must be specified.
- If the audit event being reported is a deaccess event (NSA$C_MSG_OBJ_DEACCESS), the NSA$_OBJECT_CLASS item code must be specified.

The item list is a standard format item list. The following diagram depicts the general structure of an item descriptor.



The following table defines the item descriptor fields.

Descriptor Field	Definition
Buffer length	A word specifying the length (in bytes) of the buffer; the buffer supplies information to be used by $AUDIT_EVENT. The required length of the buffer varies, depending on the item code specified; each item code description specifies the required length.
Item code	A word containing a symbolic code describing the nature of the information currently in the buffer. The location of the buffer is pointed to by the buffer address field. Each item code has a symbolic name. This section provides a detailed description of item codes following the description of arguments.
Buffer address	A longword containing the address of the buffer that specifies the information.
Return length address	Not currently used; this field is reserved to OpenVMS. You must specify 0.

See the section called “Item Codes” for a description of the $AUDIT_EVENT item codes.

audsts

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

Longword condition value that receives the final completion status from the operation. If a security audit is required, the final completion status represents either the successful completion of the resulting

security audit or any failing status that occurred while the security audit was performed within the audit server process.

The *audsts* argument is valid only when the service returns success and the status is not SS$_EVTNOTENAB. In addition, the caller must either make use of the *astadr* argument or use the $AUDIT_EVENTW service before attempting to access *audsts*.

astadr

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

Asynchronous system trap (AST) routine to be executed after the *audsts* is updated. The *astadr* argument, which is the address of a longword value, is the procedure value of the AST routine.

The AST routine executes in the access mode of the caller of $AUDIT_EVENT.

astprm

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

Asynchronous system trap (AST) parameter passed to the AST service routine. The *astprm* argument is a longword value containing the AST parameter.

Item Codes

This section provides a list of item codes that can be used to affect auditing.

NSA$_ALARM_NAME is a string of 1 to 32 characters specifying an alarm journal name to receive the record. To direct an event to the system alarm journal (that is, all enabled security operator terminals), use the string SECURITY.

NSA$_AUDIT_NAME is a string of 1 to 65 characters specifying the journal file to receive the audit record. To direct an event to the system audit journal, use the string SECURITY.

NSA$_CHAIN is a longword value specifying the item list to process immediately after the current one. The buffer address field in the item descriptor specifies the address of the next item list to be processed. Anything after NSA$_CHAIN is ignored.

NSA$_EVENT_FACILITY is a word value specifying the facility generating the event. All operating system events are audited as facility zero.

NSA$_EVENT_SUBTYPE is a longword value specifying an event message subtype. See Table 17 for a list of valid event subtypes.

NSA$_EVENT_TYPE is a longword value specifying an event message type. See Table 17 for a list of valid event types.

Table 17. ***Description of $AUDIT_EVENT Types and Subtypes***

Symbol of Event Type	**Meaning**
NSA$C_MSG_AUDIT	Systemwide change to auditing
Subtype and Meaning	
NSA$C_AUDIT_DISABLED	Audit events disabled
NSA$C_AUDIT_ENABLED	Audit events enabled
NSA$C_AUDIT_INITIATE	Audit server startup
NSA$C_AUDIT_TERMINATE	Audit server shutdown
NSA$C_AUDIT_LOG_FINAL	Final entry in audit log (forward link)
NSA$C_AUDIT_LOG_FIRST	First entry in audit log (backward link)
NSA$C_MSG_BREAKIN	Break-in attempt detected
Subtype and Meaning	
NSA$C_DETACHED	Detached process
NSA$C_DIALUP	Dialup interactive process
NSA$C_LOCAL	Local interactive process
NSA$C_NETWORK	Network server process
NSA$C_REMOTE	Interactive process from another network node
NSA$C_MSG_CONNECTION	Logical link connection or termination
Subtype and Meaning	
NSA$C_CNX_ABORT	Connection aborted
NSA$C_CNX_ACCEPT	Connection accepted
NSA$C_CNX_DECNET_CREATE	DECnet for OpenVMS logical link created
NSA$C_CNX_DECNET_DELETE	DECnet for OpenVMS logical link disconnected
NSA$C_CNX_DISCONNECT	Connection disconnected
NSA$C_CNX_IPC_CLOSE	Interprocess communication association closed
NSA$C_CNX_IPC_OPEN	Interprocess communication association opened
NSA$C_CNX_REJECT	Connection rejected
NSA$C_CNX_REQUEST	Connection requested
NSA$C_CNX_INC_REQUEST	Incoming connection requested
NSA$C_CNX_INC_ACCEPT	Incoming connection accepted
NSA$C_CNX_INC_REJECT	Incoming connection rejected
NSA$C_CNX_INC_DISCONNECT	Incoming connection disconnected
NSA$C_CNX_INC_ABORT	Incoming connection aborted
NSA$C_MSG_INSTALL	Use of the Install utility (INSTALL)
Subtype and Meaning	
NSA$C_INSTALL_ADD	Known image installed
NSA$C_INSTALL_REMOVE	Known image deleted

Symbol of Event Type	Meaning
NSA$C_MSG_LOGFAIL	Login failure
Subtype and Meaning	
NSA$C_BATCH	Batch process
NSA$C_DETACHED	Detached process
NSA$C_DIALUP	Dialup interactive process
NSA$C_LOCAL	Local interactive process
NSA$C_NETWORK	Network server process
NSA$C_REMOTE	Interactive process from another network node
NSA$C_SUBPROCESS	Subprocess
NSA$C_MSG_LOGIN	Successful login
Subtype and Meaning	
See subtypes for NSA$C_MSG_ LOGFAIL	
NSA$C_MSG_LOGOUT	Successful logout
Subtype and Meaning	
See subtypes for NSA$C_MSG_ LOGFAIL	
NSA$C_MSG_MOUNT	Volume mount or dismount
Subtype and Meaning	
NSA$C_VOL_DISMOUNT	Volume dismount
NSA$C_VOL_MOUNT	Volume mount
NSA$C_MSG_NCP	Modification to network configuration database
Subtype and Meaning	
NSA$C_NCP_COMMAND	Network Control Program (NCP) command issued
NSA$C_MSG_NETPROXY	Modification to network proxy database
Subtype and Meaning	
NSA$C_NETPROXY_ADD	Record added to network proxy database
NSA$C_NETPROXY_DELETE	Record removed from network proxy database
NSA$C_NETPROXY_MODIFY	Record modified in network proxy database
NSA$C_MSG_OBJ_ACCESS	Object access attempted
Subtype and Meaning	
NSA$C_OBJ_ACCESS	Object access attempted

Symbol of Event Type	Meaning
NSA$C_MSG_OBJ_CREATE	Object created
Subtype and Meaning	
NSA$C_OBJ_CREATE	Object created
NSA$C_MSG_OBJ_DEACCESS	Object deaccessed
Subtype and Meaning	
NSA$C_OBJ_DEACCESS	Object deaccessed
NSA$C_MSG_OBJ_DELETE	Object deleted
Subtype and Meaning	
NSA$C_OBJ_DELETE	Object deleted
NSA$C_MSG_PROCESS	Process control system service issued
Subtype and Meaning	
NSA$C_PRC_CANWAK	Process wakeup canceled
NSA$C_PRC_CREPRC	Process created
NSA$C_PRC_DELPRC	Process deleted
NSA$C_PRC_FORCEX	Process exit forced
NSA$C_PRC_GETJPI	Process information gathered
NSA$C_PRC_GRANTID	Process identifier granted
NSA$C_PRC_RESUME	Process resumed
NSA$C_PRC_REVOKID	Process identifier revoked
NSA$C_PRC_SCHDWK	Process wakeup scheduled
NSA$C_PRC_SETPRI	Process priority altered
NSA$C_PRC_SIGPRC	Process exception issued
NSA$C_PRC_SUSPND	Process suspended
NSA$C_PRC_WAKE	Process wakeup issued
NSA$C_PRC_PRCTERM	Process termination notification requested
NSA$C_MSG_PRVAUD	Attempt to use privilege
Subtype and Meaning	
NSA$C_PRVAUD_FAILURE	Unsuccessful use of privilege
NSA$C_PRVAUD_SUCCESS	Successful use of privilege
NSA$C_MSG_RIGHTSDB	Modification to rights database
Subtype and Meaning	
NSA$C_RDB_ADD_ID	Identifier added to rights database
NSA$C_RDB_CREATE	Rights database created

Symbol of Event Type	Meaning
NSA$C_RDB_GRANT_ID	Identifier given to user
NSA$C_RDB_MOD_HOLDER	List of identifier holders modified
NSA$C_RDB_MOD_ID	Identifier name or attributes modified
NSA$C_RDB_REM_ID	Identifier removed from rights database
NSA$C_RDB_REVOKE_ID	Identifier revoked from user
NSA$C_MSG_SYSGEN	Modification of a system parameter using the System Generation utility (SYSGEN)
Subtype and Meaning	
NSA$C_SYSGEN_SET	System parameter modified
NSA$C_MSG_SYSTIME	Modification to system time
Subtype and Meaning	
NSA$C_SYSTIM_SET	System time set
NSA$C_SYSTIM_CAL	System time calibrated
NSA$C_MSG_SYSUAF	Modification to system user authorization file (SYSUAF)
Subtype and Meaning	
NSA$C_SYSUAF_ADD	Record added to SYSUAF
NSA$C_SYSUAF_COPY	Record copied in SYSUAF
NSA$C_SYSUAF_DELETE	Record deleted from SYSUAF
NSA$C_SYSUAF_MODIFY	Record modified in SYSUAF
NSA$C_SYSUAF_RENAME	Record renamed in SYSUAF

NSA$_FIELD_NAME is a string of 1 to 256 characters specifying the name of the field being modified. This is used in combination with NSA$_ORIGINAL_DATA and NSA$_NEW_DATA.

NSA$_MESSAGE specifies a system message code. The $FORMAT_AUDIT service will use the $GETMSG service to translate the message into text. The resulting text is inserted into the formatted audit message, with the “Event information:” prefix. For example, the operating system uses this item code to supply the privilege audit text associated with privilege audit events; this keeps the audit records small. By default, the $GETMSG service can only translate resident system messages. You can use the NSA$_MSGFILNAM item code to specify the name of an application or site-specific message file.

NSA$_MSGFILNAM is a string of 1 to 255 characters specifying the message file containing the translation for the message code in NSA$_MESSAGE. The default file specification is SYS$MESSAGE:.EXE. By default, $FORMAT_AUDIT uses the resident system message file.

NSA$_NEW_DATA is a string of 1 to *n* characters specifying the contents of the field named in NSA$_FIELD_NAME after the event occurred. NSA$_ORIGINAL_DATA contains the field contents prior to the event.

NSA$_NOP specifies that the item list entry should be ignored. This item code allows you to build a static item list and then remove those entries that do not apply to the current event.

NSA$_ORIGINAL_DATA is a string of 1 to *n* characters specifying the contents of the field named in NSA$_FIELD_NAME before the event occurred. NSA$_NEW_DATA contains the field contents following the event.

NSA$_SENSITIVE_FIELD_NAME is a string of 1 to 256 characters specifying the name of the field being modified. This is used in combination with NSA$_SENSITIVE_ORIG_DATA and NSA$_SENSITIVE_NEW_DATA. Use NSA$_SENSITIVE_FIELD_NAME to prevent sensitive information, such as passwords, from being displayed in an alarm message. Sensitive information is written to the audit log.

NSA$_SENSITIVE_NEW_DATA is a string of 1 to *n* characters specifying the contents of the field named in NSA$_SENSITIVE_FIELD_NAME after the event occurred. NSA$_SENSITIVE_ORIG_DATA contains the field contents prior to the event. Use NSA$_SENSITIVE_NEW_DATA to prevent sensitive information from being displayed in an alarm message. Sensitive information is written to the audit log.

NSA$_SENSITIVE_ORIG_DATA is a string of 1 to *n* characters specifying the contents of the field named in NSA$_SENSITIVE_FIELD_NAME before the event occurred. NSA$_SENSITIVE_NEW_DATA contains the field contents following the event. Use NSA$_SENSITIVE_FIELD_NAME to prevent sensitive information from being displayed in an alarm message. Sensitive information is written to the audit log.

NSA$_SUPPRESS is a longword bitmask directing $AUDIT_EVENT to ignore the defaults for the following values and either omit the information from the event record or use the value provided in another parameter. The bits in the mask inhibit the use of default values for the following item codes:

NSA$V_ACCOUNT_NAME	NAS$V_PROCESS_NAME
NSA$V_FINAL_STATUS	NSA$V_SUBJECT_CLASS
NSA$V_IMAGE_NAME	NSA$V_SUBJECT_OWNER
NSA$V_PARENT_ID	NSA$V_SYSTEM_ID
NSA$V_PARENT_NAME	NSA$V_SYSTEM_OWNER
NSA$V_PARENT_OWNER	NSA$V_TERMINAL
NSA$V_PARENT_USERNAME	NSA$V_TIME_STAMP
NSA$V_PROCESS_ID	NSA$V_USERNAME

Use NSA$_SUPPRESS, for example, when auditing events from server processes when the default values for many of these items need to explicitly reference the client context rather than be defaulted from the environment of the server.

The following is a list of additional item codes that are valid as an item descriptor in the `itmlst` argument.

NSA$_ACCESS_DESIRED is a longword value specifying the access request mask as defined in $ARMDEF.

NSA$_ACCESS_MODE is a byte value specifying an access mode associated with the event.

NSA$_ACCOUNT is a string of 1 to 32 characters specifying the account name associated with the event.

NSA$_ASSOCIATION_NAME is a string of 1 to 256 characters specifying an association name.

NSA$_COMMAND_LINE is a string of 1 to 2048 characters specifying a command line.

NSA$_CONNECTION_ID is a longword value specifying a connection identification.

NSA$_DECNET_LINK_ID is a longword value specifying a DECnet for OpenVMS logical link identification.

NSA$_DECNET_OBJECT_NAME is a string of 1 to 16 characters specifying a DECnet for OpenVMS object name.

NSA$_DECNET_OBJECT_NUMBER is a longword value specifying a DECnet for OpenVMS object number.

NSA$_DEFAULT_USERNAME is a string of 1 to 32 characters specifying a default local user name for incoming network proxy requests.

NSA$_DEVICE_NAME is a string of 1 to 64 characters specifying the name of the device where the volume resides.

NSA$_DIRECTORY_ENTRY is a string of 1 to 256 characters specifying the name of the directory entry associated with an XQP operation.

NSA$_DIRECTORY_ID is an array of three words specifying the directory file identification.

NSA$_DISMOUNT_FLAGS is a longword value specifying the dismount flags that are defined by the $DMTDEF macro in STARLET.

NSA$_EFC_NAME is a string of 1 to 16 characters specifying the event flag cluster name.

NSA$_FILE_ID is an array of three words specifying the file identification.

NSA$_FINAL_STATUS is a longword value specifying the successful or unsuccessful status that caused the auditing facility to be invoked.

NSA$_HOLDER_NAME is a string of 1 to 32 characters specifying the name of the user holding the identifier.

NSA$_HOLDER_OWNER is a longword value specifying the owner (UIC) of the holder.

NSA$_ID_ATTRIBUTES is a longword value specifying the attributes of the identifier, which are defined by the $KGBDEF macro in STARLET.

NSA$_IDENTIFIERS_USED is an array of longwords specifying the identifiers (from the access control entry [ACE] granting access) that were used to gain access to the object.

NSA$_ID_NAME is a string of 1 to 32 characters specifying the name of the identifier.

NSA$_ID_NEW_ATTRIBUTES is a longword value specifying the new attributes of the identifier, which are defined by the $KGBDEF macro in STARLET.

NSA$_ID_NEW_NAME is a string of 1 to 32 characters specifying the new name of the identifier.

NSA$_ID_NEW_VALUE is a longword value specifying the new value of the identifier.

NSA$_ID_VALUE is a longword value specifying the value of the identifier.

NSA$_ID_VALUE_ASCII is a longword specifying the value of the identifier.

NSA$_IMAGE_NAME is a string of 1 to 1024 characters specifying the name of the image being executed when the event took place.

NSA$_INSTALL_FILE is a string of 1 to 255 characters specifying the name of the installed file.

NSA$_INSTALL_FLAGS is a longword value specifying the INSTALL flags. They correspond to qualifiers for the Install utility; for example, NSA$M_INS_EXECUTE_ONLY.

NSA$_LNM_PARENT_NAME is a string of 1 to 31 characters specifying the name of the parent logical name table.

NSA$_LNM_TABLE_NAME is a string of 1 to 31 characters specifying the name of the logical name table.

NSA$_LOCAL_USERNAME is a string of 1 to 32 characters specifying user names of the accounts available for incoming network proxy requests.

NSA$_LOGICAL_NAME is a string of 1 to 255 characters specifying the logical name associated with the device.

NSA$_MAILBOX_UNIT is a longword value specifying the mailbox unit number.

NSA$_MATCHING_ACE is an array of bytes specifying the ACE granting or denying access.

NSA$_MOUNT_FLAGS is a quadword value specifying mount flags that are defined by the $MNTDEF macro in STARLET.

NSA$_NEW_IMAGE_NAME is a string of 1 to 1024 characters specifying the name of the new image.

NSA$_NEW_OWNER is a longword value specifying the new process owner (UIC).

NSA$_NEW_PRIORITY is a longword value specifying the new process priority.

NSA$_NEW_PRIVILEGES is a quadword privilege mask specifying the new privileges. The $PRVDEF macro defines the list of available privileges.

NSA$_NEW_PROCESS_ID is a longword value specifying the new process identification.

NSA$_NEW_PROCESS_NAME is a string of 1 to 15 characters specifying the name of the new process.

NSA$_NEW_PROCESS_OWNER is a longword value specifying the owner (UIC) of the new process.

NSA$_NEW_USERNAME is a string of 1 to 32 characters specifying the new user name.

NSA$_OBJECT_CLASS is a string of 1 to 23 characters specifying the security object class associated with the event; for example, FILE.

NSA$_OBJECT_ID is an array of three words specifying the unique object identification code, which is currently applicable only to files; therefore, it is the file identification.

NSA$_OBJECT_MAX_CLASS is a 20-byte record specifying the maximum access classification of the object.

NSA$_OBJECT_MIN_CLASS is a 20-byte record specifying the minimum access classification of the object.

NSA$_OBJECT_NAME is a string of 1 to 255 characters specifying an object's name.

NSA$_OBJECT_NAME_2 is a string of 1 to 255 characters specifying an alternate object name; currently it applies to file-backed global sections where the alternate name of a global section is the file name.

NSA$_OBJECT_OWNER is a longword value specifying the UIC or general identifier of the process causing the auditable event.

NSA$_OBJECT_PROTECTION is a word, or an array of four longwords, specifying the UIC-based protection of the object.

NSA$_OLD_PRIORITY is a longword value specifying the former process priority.

NSA$_OLD_PRIVILEGES is a quadword privilege mask specifying the former privileges. The $PRVDEF macro defines the list of available privileges.

NSA$_PARAMS_INUSE is a string of 1 to 255 characters specifying the name of the parameter file given to the SYSGEN command USE.

NSA$_PARAMS_WRITE is a string of 1 to 255 characters specifying the file name for the SYSGEN command WRITE.

NSA$_PARENT_ID is a longword value specifying the process identification (PID) of the parent process. It is used only when auditing events pertaining to a subprocess.

NSA$_PARENT_NAME is a string of 1 to 15 characters specifying the parent's process name. It is used only when auditing events pertaining to a subprocess.

NSA$_PARENT_OWNER is longword value specifying the owner (UIC) of the parent process. It is used only when auditing events pertaining to a subprocess.

NSA$_PARENT_USERNAME is a string of 1 to 32 characters specifying the user name associated with the parent process. It is used only when auditing events pertaining to a subprocess.

NSA$_PASSWORD is a string of 1 to 32 characters specifying the password used in an unsuccessful break-in attempt. By default, system security alarms do not include break-in passwords.

NSA$_PRIVILEGES is a quadword privilege mask specifying the privileges used to gain access. The $PRVDEF macro defines the list of available privileges.

NSA$_PRIVS_MISSING is a longword or a quadword privilege mask specifying the privileges that are needed. The privileges are defined by a macro in STARLET; see the $CHPDEF macro for definition as a longword mask and see the $PRVDEF macro for definition as a quadword privilege mask.

NSA$_PRIVS_USED is a longword or a quadword privilege mask specifying the privileges used to gain access to the object. The privileges are defined by a macro in STARLET; see the $CHPDEF macro for definition as a longword mask and see the $PRVDEF macro for definition as a quadword privilege mask.

NSA$_PROCESS_ID is a longword value specifying the PID of the process causing the auditable event.

NSA$_PROCESS_NAME is a string of 1 to 15 characters specifying the process name that caused the auditable event.

NSA$_REM_ASSOCIATION_NAME is a string of 1 to 256 characters specifying the interprocess communication (IPC) remote association name.

NSA$_REMOTE_LINK_ID is a longword value specifying the remote logical link ID.

NSA$_REMOTE_NODE_FULLNAME is a string of 1 to 255 characters specifying the fully expanded DECnet for OpenVMS node name of the remote process.

NSA$_REMOTE_NODE_ID is a string of 4 to 24 characters specifying the DECnet for OpenVMS node address of the remote process. A value 4 bytes in length is a DECnet Phase IV node address. A value with length greater than 4 bytes is a DECnet/OSI NSAP address.

NSA$_REMOTE_NODENAME is a string of 1 to 6 characters specifying the DECnet for OpenVMS node name of the remote process.

NSA$_REMOTE_USERNAME is a string of 1 to 32 characters specifying the user name of the remote process.

NSA$_REQUEST_NUMBER is a longword value specifying the request number associated with the system service call.

NSA$_RESOURCE_NAME is a string of 1 to 32 characters specifying the lock resource name.

NSA$_SECTION_NAME is a string of 1 to 42 characters specifying the global section name.

NSA$_SNAPSHOT_BOOTFILE is a string of 1 to 255 characters specifying the name of the snapshot boot file, the saved system image file from which the system just booted.

NSA$_SNAPSHOT_SAVE_FILNAM is a string of 1 to 255 characters specifying the name of the snapshot save file, which is the original location of the snapshot file at the time that the system was saved.

NSA$_SNAPSHOT_TIME is a quadword value specifying the time the picture of the configuration was taken and saved in the snapshot boot file.

NSA$_SOURCE_PROCESS_ID is a longword value specifying the process identification of the process originating the request.

NSA$_SUBJECT_CLASS is a 20-byte record specifying the current access class of the process causing the auditable event.

NSA$_SUBJECT_OWNER is a longword value specifying the owner (UIC) of the process causing the event.

NSA$_SYSTEM_ID is a longword value specifying the SCS identification of the cluster node where the event took place (system parameter SCSSYSTEMID).

NSA$_SYSTEM_NAME is a string of 1 to 6 characters specifying the System Communications Services (SCS) node name where the event took place (system parameter SCSNODE).

NSA$_SYSTEM_SERVICE_NAME is a string of 1 to 256 characters specifying the name of the system service associated with the event.

NSA$_SYSTIM_NEW is a quadword value specifying the new system time.

NSA$_SYSTIM_OLD is a quadword value specifying the old system time.

NSA$_TARGET_DEVICE_NAME is a string of 1 to 64 characters specifying the target device name.

NSA$_TARGET_PROCESS_CLASS is a 20-byte record specifying the target process classification.

NSA$_TARGET_PROCESS_ID is a longword value specifying the target process identifier (PID).

NSA$_TARGET_PROCESS_NAME is a string of 1 to 64 characters specifying the target process name.

NSA$_TARGET_PROCESS_OWNER is a longword value specifying the target owner (UIC).

NSA$_TARGET_USERNAME is a string of 1 to 32 characters specifying the target process user name.

NSA$_TERMINAL is a string of 1 to 256 characters specifying the name of the terminal to which the process was connected when the auditable event occurred.

NSA$_TIME_STAMP is a quadword value specifying the time when the event occurred.

NSA$_TRANSPORT_NAME is a string of 1 to 256 characters specifying the name of the transport: interprocess communication, DECnet for OpenVMS, or System Management Integrator (SMI), which handles requests from SYSMAN (ASCII string).

NSA$_UAF_ADD is a string of 1 to 32 characters specifying the name of the authorization record being added.

NSA$_UAF_COPY is a string of 1 to 32 characters specifying the new name of the authorization record being copied from NSA$_UAF_SOURCE.

NSA$_UAF_DELETE is a string of 1 to 32 characters specifying the name of the authorization record being removed.

NSA$_UAF_MODIFY is a string of 1 to 32 characters specifying the name of the authorization record being modified.

NSA$_UAF_RENAME is a string of 1 to 32 characters specifying the name of the authorization record being renamed.

NSA$_UAF_SOURCE is a string of 1 to 32 characters specifying the user name of the source record for an Authorize utility (AUTHORIZE) copy operation.

NSA$_USERNAME is a string of 1 to 32 characters specifying the user name of the process causing the auditable event.

NSA$_VOLUME_NAME is a string of 1 to 15 characters specifying a volume name.

NSA$_VOLUME_SET_NAME is a string of 1 to 15 characters specifying a volume set name.

Description

The Audit Event service can be called by any program that enforces a security policy in order to append an event message to the audit log file or send an alarm to an operator terminal. For example, AUTHORIZE calls $AUDIT_EVENT whenever a UAF record is altered and LOGINOUT calls the service whenever a user logs in.

$AUDIT_EVENT takes the event message, checks the auditing database to determine whether a class of event is being audited, and, if the event class is enabled, creates an alarm or audit record.

$AUDIT_EVENT completes asynchronously; that is, it does not wait for final status. For synchronous completion, use the $AUDIT_EVENTW service.

Required Access or Privileges

AUDIT

Required Quota

None

Related Services

$CHECK_ACCESS, $CHECK_PRIVILEGE, $CHKPRO

Condition Values Returned

SS$_NORMAL

The service completed successfully.

SS$_ACCVIO

A parameter is not accessible.

SS$_BADBUFADR

The buffer address is invalid or not readable.

SS$_BADBUFLEN

The specified buffer length is invalid or out of range.

SS$_BADCHAIN

The address of the next item list to be processed, as identified in the buffer address field, is either not readable or points to itself.

SS$_BADITMCOD

The specified item code is invalid or out of range.

SS$_EVTNOTENAB

The event is not enabled.

SS$_INSFARG

A required item code or parameter is missing.

SS$_INVAJLNAM

The alarm or audit journal name is invalid.

SS$_IVSTSFLG

The specified system service flags are invalid.

SS$_NOAUDIT

The caller does not have the required privilege to perform the audit.

SS$_OVRMAXAUD

There is insufficient memory to perform the audit.

SS$_SYNCH

An audit was not required.

$AUDIT_EVENTW

Audit Event and Wait — Determines whether a security-related event should be reported. If the event should be reported, the service sends the event report to the audit server. The $AUDIT_EVENTW service completes synchronously; that is, it returns only after receiving an explicit confirmation from the audit server that the associated audit, if enabled, has been performed. For asynchronous completion, use the Audit Event ($AUDIT_EVENT) service. In all other respects, $AUDIT_EVENTW is identical to $AUDIT_EVENT. For additional information about $AUDIT_EVENTW, see the $AUDIT_EVENT service.

Format

```
SYS$AUDIT_EVENTW efn ,[flags] ,itmlst ,audsts  ,[astadr] ,[astprm]
```

C Prototype

```
int sys$audit_eventw
   (unsigned int efn, unsigned int flags, void *itmlst,
    unsigned int *audsts, void (*astadr)(__unknown_params), int astprm);
```

$AVOID_PREEMPT

Avoid Process Preemption — Requests that the EXEC avoid preempting the calling process or thread.

Format

```
SYS$AVOID_PREEMPT enable
```

C Prototype

```
int sys$avoid_preempt (int enable);
```

Arguments

enable

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

Enables or disables preemption avoidance. If the *enable* argument is set to 1, preemption avoidance is enabled; if 0, preemption avoidance is disabled.

Description

The Avoid Process Preemption service is a caller's mode service that sets a thread-specific bit that informs the scheduler that this thread desires to avoid preemption. Before setting the bit, it checks if the process or thread has already benefited from preemption avoidance during this time on the processor, and if it has, calls the $RESCHED system service to give up the processor.

If quantum end is reached when this bit is set, the scheduler will “borrow” the next quantum for this process or thread. It will give the process or thread another quantum immediately and allow it to resume execution. The next time that the process or thread is eligible for scheduling, it will be placed at the end of the scheduling queue without any execution time, skipping its next quantum.

If another process or thread of the same base priority attempts to preempt a process or thread that has this bit set, this preemption can be avoided if the process had the ALTPRI privilege when the $SETUP_AVOID_PREEMPT service was called. In this case, the priority of the current thread is boosted to the same level as the thread attempting preemption, denying the attempted preemption.

Required Access or Privileges

ALTPRI

Required Quota

None

Related Services

$RESCHED, $SETUP_AVOID_PREEMPT

Condition Values Returned

SS$_NORMAL

The service completed successfully.

$BINTIM

Convert ASCII String to Binary Time — Converts an ASCII string to an absolute or delta time value in the system 64-bit time format suitable for input to the Set Timer ($SETIMR) or Schedule Wakeup ($SCHDWK) service. On Alpha and Integrity server systems, this service accepts 64-bit addresses.

Format

```
SYS$BINTIM timbuf ,timadr
```

C Prototype

```
int sys$bintim (void *timbuf, struct _generic_64 *timadr);
```

Arguments

timbuf

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

Buffer that holds the ASCII time to be converted. The *timbuf* argument specifies the 32- or 64-bit address of a character string descriptor pointing to the time string. The time string specifies the absolute or delta time to be converted by $BINTIM. The data type table describes the time string.

timadr

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

Time value that $BINTIM has converted. The *timadr* argument is the 32- or 64-bit address of the quadword system time, which receives the converted time.

Description

The Convert ASCII String to Binary Time service converts an ASCII string to an absolute or delta time value in the system 64-bit time format suitable for input to the Set Timer ($SETIMR) or Schedule Wakeup ($SCHDWK) service. The service executes at the access mode of the caller and does not check whether address arguments are accessible before it executes. Therefore, an access violation causes an exception condition if the input buffer or buffer descriptor cannot be read or the output buffer cannot be written.

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

The required ASCII input strings have the following format:

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

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

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

Month abbreviations must be uppercase.

The hundredths-of-second field represents a true fraction. For example, the string .1 represents ten-hundredths of a second (one-tenth of a second) and the string .01 represents one-hundredth of a second. Also, you can add a third digit to the hundredths-of-second field; this thousandths-of-second digit is used to round the hundredths-of-second value. Digits beyond the thousandths-of-second digit are ignored.

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

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

  Leaving out any element, however, including hundredths of a second, results in the $BINTIM service supplying the current base time for the missing element. Trailing fields can be truncated. If leading fields are omitted, you must specify the punctuation (hyphens, blanks, colons, periods). For example, the following string results in an absolute time of 12:00 on the current day:

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

  For delta time values, the $BINTIM service uses a default value of 0 for unspecified hours, minutes, and seconds fields. Trailing fields can be truncated. If you omit leading fields from the time value,

you must specify the punctuation (blanks, colons, periods). If the number of days in the delta time is 0, you must specify a 0. For example, the following string results in a delta time of 10 seconds:

```
0 ::10
```

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

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

Required Access or Privileges

None

Required Quota

None

Related Services

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

Condition Values Returned

SS$_NORMAL

The service completed successfully.

SS$_IVTIME

The syntax of the specified ASCII string is invalid, or the time component is out of range.

Example

Column 1 of the following table lists legal input strings to the $BINTIM service; column 2 lists the $BINTIM output of these strings translated through the Convert Binary Time to ASCII String ($ASCTIM) system service. The current date is assumed to be 30-DEC-2003 04:15:28.00.

Input to $BINTIM	$ASCTIM Output String
–: 50	30-DEC-2003 04:50:28.00
– 2003 0:0:0.0	29-DEC-2003 00:00:00.00
30-DEC-2003 12:32:1.1161	30-DEC-2003 12:32:01.12
29-DEC-2003 16:35:0.0	29-DEC-2003 16:35:00.00
0 ::.1	0 00:00:00.10
0 ::.06	0 00:00:00.06
5 3:18:32.068	5 03:18:32:07
20 12:	20 12:00:00.00
0 5	0 05:00:00.00

$BINUTC

Convert ASCII String to UTC Binary Time — Converts an ASCII string to an absolute time value in the 128-bit UTC format. On Alpha and Integrity server systems, this service accepts 64-bit addresses.

Format

```
SYS$BINUTC timbuf ,utcadr
```

C Prototype

```
int sys$binutc (void *timbuf, unsigned int *utcadr [4]);
```

Arguments

timbuf

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

Buffer that holds the ASCII time to be converted. The *timbuf* argument specifies the 32- or 64-bit address of a character string descriptor pointing to a local time string. The time string specifies the absolute time to be converted by $BINUTC.

utcadr

OpenVMS usage:	coordinated universal time
type:	utc_date_time
access:	write only
mechanism:	by 32- or 64-bit reference

Time value that $BINUTC has converted. The *utcadr* argument is the 32- or 64-bit address of a 16-byte location to receive the converted time.

Description

The Convert ASCII String to UTC Binary Time service converts an ASCII string to an absolute time in the 128-bit UTC format. The service executes at the access mode of the caller and does not check whether address arguments are accessible before it executes. Therefore, an access violation causes an exception condition if the input buffer or buffer descriptor cannot be read or the output buffer cannot be written.

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

$BINUTC uses the time zone differential factor of the local system to encode the 128-bit UTC.

The required ASCII input strings have the following format:

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

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

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

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

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

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

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

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

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

Required Access or Privileges

None

Required Quota

None

Related Services

$ASCUTC, $GETUTC, $NUMUTC, $TIMCON

Condition Values Returned

SS$_NORMAL

The service completed successfully.

SS$_IVTIME

The syntax of the specified ASCII string is invalid, the specified time is a delta time, or the time component is out of range.

$BRKTHRU

Breakthrough — Sends a message to one or more terminals. The $BRKTHRU service completes asynchronously; that is, it returns to the caller after queuing the message request, without waiting for the message to be written to the specified terminals. For synchronous completion, use the Breakthrough and Wait ($BRKTHRUW) service. The $BRKTHRUW service is identical to the $BRKTHRU service in every way except that $BRKTHRUW returns to the caller after the message is written to the specified terminals. For additional information about system service completion, refer to the Synchronize ($SYNCH) service. The $BRKTHRU service supersedes the Broadcast ($BRDCST) service. When writing new programs, you should use $BRKTHRU instead of $BRDCST. When updating old programs, you should change all uses of $BRDCST to $BRKTHRU.

Format

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

C Prototype

```
int sys$brkthru
   (unsigned int efn, void *msgbuf, void *sendto,  unsigned int sndtyp,
    struct _iosb *iosb, unsigned int carcon, unsigned int  flags,
    unsigned int reqid, unsigned int timout,
    void (*astadr)(__unknown_params), int 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 message has been written to the specified terminals. The *efn* argument is a longword containing this number; however, $BRKTHRU uses only the low-order byte.

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

msgbuf

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

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

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

sendto

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

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

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

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

sndtyp

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

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

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

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

iosb

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

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

When the *iosb* argument is specified, $BRKTHRU sets the quadword to 0 when it queues the message request. Then, after the message is sent to the specified terminals, $BRKTHRU returns four informational items, one item per word, in the quadword I/O status block.

These informational items indicate the status of the messages sent only to terminals and mailboxes on the local node; these items do not include the status of messages sent to terminals and mailboxes on other nodes in an OpenVMS Cluster system.

The following table shows each word of the quadword block and the informational item it contains.

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

carcon

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

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

For a list of the carriage control specifiers that you can use in the *carcon* argument, refer to the *VSI OpenVMS I/O User's Reference Manual*.

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

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

flags

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

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

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

Symbolic Name	Description
BRK$V_ERASE_LINES	When specified with the BRK$M_SCREEN flag, BRK$V_ERASE_LINES causes a specified number of lines to be cleared from the screen before the message is displayed. When BRK$M_SCREEN is not also specified, BRK$V_ERASE_LINES is ignored. Unlike the other Boolean flags, BRK$V_ERASE_LINES specifies a 1-byte integer in the range 0 to 24. It occupies the first byte in the longword flag mask. In coding the call to $BRKTHRU, specify the desired integer value in the OR operation with any other desired flags.
BRK$M_SCREEN	When specified, $BRKTHRU sends screen-formatted messages as well as messages formatted through the use of the *carcon* argument. $BRKTHRU sends screen-formatted messages to terminals with the DEC_CRT characteristic, and it sends messages formatted by *carcon* to those without the DEC_CRT characteristic. You set the

Symbolic Name	Description
	DEC_CRT characteristic for the terminal by using the DCL command SET TERMINAL/DEC_CRT. A screen-formatted message is displayed at the top of the terminal screen, and the cursor is repositioned at the point it was prior to the broadcast message. However, the BRK$V_ERASE_LINES and BRK$M_BOTTOM flags also affect the display.
BRK$M_BOTTOM	When BRK$M_BOTTOM is specified and BRK$M_SCREEN is also specified, $BRKTHRU writes the message to the bottom of the terminal screen instead of the top. BRK$M_BOTTOM is ignored if the BRK$M_SCREEN flag is not set.
BRK$M_NOREFRESH	When BRK$M_NOREFRESH is specified, $BRKTHRU, after writing the message to the screen, does not redisplay the last line of a read operation that was interrupted by the broadcast message. This flag is useful only when the BRK$M_SCREEN flag is not specified, because BRK$M_NOREFRESH is the default for screen-formatted messages.
BRK$M_CLUSTER	Specifying BRK$M_CLUSTER enables $BRKTHRU to send the message to terminals or mailboxes on other nodes in an OpenVMS Cluster system. If BRK$M_CLUSTER is not specified, $BRKTHRU sends messages only to terminals or mailboxes on the local node.

reqid

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

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

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

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

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

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

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

Class Name	Component
BRK$C_GENERAL	This class name is used by the image invoked by the DCL command REPLY and the callers of the $BRKTHRU service. This is the default if the *reqid* argument is not specified.
BRK$C_PHONE	This class name is used by the OpenVMS Phone utility.
BRK$C_MAIL	This class name is used by the OpenVMS Mail utility.
BRK$C_DCL	This class name is used by the DIGITAL Command Language (DCL) interpreter for the Ctrl/T command, which displays the process status.
BRK$C_QUEUE	This class name is used by the queue manager, which manages print and batch jobs.
BRK$C_SHUTDOWN	This class name is used by the system shutdown image, which is invoked by the DCL command REPLY/ID=SHUTDOWN.
BRK$C_URGENT	This class name is used by the image invoked by the DCL command REPLY/ID=URGENT.
BRK$C_USER1 through BRK$C_USER16	These class names can be used by user-written images.

timout

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

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

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

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

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

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

astadr

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

AST service routine to be executed after $BRKTHRU has sent the message to the specified terminals. 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 $BRKTHRU.

astprm

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

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

Description

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

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

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

- If the terminal is performing a read operation when $BRKTHRU sends the message, the read operation is suspended, the message is displayed, and then the line that was being read when the read operation was suspended is redisplayed (equivalent to the action produced by Ctrl/R).
- If the terminal is performing a write operation when $BRKTHRU sends the message, the message is displayed after the current write operation has completed.
- If the terminal has the NOBROADCAST characteristic set for all images, or if you have disabled the receiving of messages from the image that is issuing the $BRKTHRU call (see the description of the *reqid* argument), the message is not displayed.

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

Required Access or Privileges

The calling process must have OPER privilege for the following conditions:

- To send a message to more than one terminal
- To send a message to a terminal that is allocated to another user
- To send a message to a specific user that has a different user name than the current process

To send a message to a specific user that is the same as your process requires no privileges.

Required Quota

The $BRKTHRU service allows the message text to be as long as 16,350 bytes; however, both the system parameter MAXBUF and the caller's available process buffered I/O byte count limit (BYTLM) quota must be sufficient to handle the message.

Related Services

$ALLOC, $ASSIGN, $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 message buffer, message buffer descriptor, device name string, or device name string descriptor cannot be read by the caller.

SS$_BADPARAM

The message length exceeds 16,350 bytes; the process's buffered I/O byte count limit (BYTLM) quota is insufficient; the message length exceeds the value specified by the system parameter MAXBUF; the value of the TIMOUT parameter is nonzero and less than 4 seconds; the value of the REQID is outside the range 0 to 63; or the value of the SNDTYP is not one of the legal ones listed.

SS$_EXQUOTA

The process has exceeded its buffer space quota and has disabled resource wait mode with the Set Resource Wait Mode ($SETRWM) service.

SS$_INSFMEM

The system dynamic memory is insufficient for completing the request and the process has disabled resource wait mode with the Set Resource Wait Mode ($SETRWM) service.

SS$_NONLOCAL

The device is on a remote node.

SS$_NOOPER

The process does not have the necessary OPER privilege.

SS$_NOSUCHDEV

The specified terminal does not exist, or it cannot receive the message.

Condition Values Returned in the I/O Status Block

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

$BRKTHRUW

Breakthrough and Wait — Sends a message to one or more terminals. The $BRKTHRUW service operates synchronously; that is, it returns to the caller after the message has been sent to the specified terminals. For asynchronous operations, use the Breakthrough ($BRKTHRU) service; $BRKTHRU returns to the caller after queuing the message request, without waiting for the message to be delivered. Aside from the preceding, $BRKTHRUW is identical to $BRKTHRU. For all other information about the $BRKTHRUW service, refer to the description of $BRKTHRU. For additional information about system service completion, refer to the documentation of the Synchronize ($SYNCH) service. The $BRKTHRU and $BRKTHRUW services supersede the Broadcast ($BRDCST) service. When writing new programs, you should use $BRKTHRU or $BRKTHRUW instead of $BRDCST. When updating old programs, you should change all uses of $BRDCST to $BRKTHRU or $BRKTHRUW. $BRDCST is now an obsolete system service and is no longer being enhanced.

Format

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

C Prototype

```
int sys$brkthruw
   (unsigned int efn, void *msgbuf, void *sendto,  unsigned int sndtyp,
    struct _iosb *iosb, unsigned int carcon, unsigned int flags,
    unsigned int reqid, unsigned int timout,
    void (*astadr)(__unknown_params),int astprm);
```

$CANCEL

Cancel I/O on Channel — Cancels all pending I/O requests on a specified channel. In general, this includes all I/O requests that are queued, as well as the request currently in progress.

Format

```
SYS$CANCEL chan
```

C Prototype

```
int sys$cancel (unsigned short int chan);
```

Argument

chan

OpenVMS usage:	channel
type:	word (unsigned)

access:	read only
mechanism:	by value

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

Description

The Cancel I/O on Channel service cancels all pending I/O requests on a specified channel. In general, this includes all I/O requests that are queued, as well as the request currently in progress.

When you cancel a request currently in progress, the driver is notified immediately. The actual cancellation might occur immediately, depending on the logical state of the driver. When cancellation does occur, the following action for I/O in progress, similar to that for queued requests, takes place:

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

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

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

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

Required Access or Privileges

To cancel I/O on a channel, the access mode of the calling process must be equal to or more privileged than the access mode that the process had when it originally made the channel assignment.

Required Quota

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

Related Services

$ALLOC, $ASSIGN, $BRKTHRU, $BRKTHRUW, $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$_EXQUOTA

The process has exceeded its buffered I/O limit (BIOLM) quota.

SS$_INSFMEM

The system dynamic memory is insufficient for canceling the I/O.

SS$_IVCHAN

You specified an invalid channel, that is, a channel number of 0 or a number larger than the number of channels available.

SS$_NOPRIV

The specified channel is not assigned or was assigned from a more privileged access mode.

$CANEXH

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

Format

```
SYS$CANEXH [desblk]
```

C Prototype

```
int sys$canexh (void *desblk);
```

Argument

desblk

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

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

Condition Values Returned

SS$_NORMAL

The service completed successfully.

SS$_ACCVIO

The first longword of the exit control block or the first longword of a previous exit control block in the list cannot be read by the caller, or the first longword of the preceding control block cannot be written by the caller.

SS$_IVSSRQ

The call to the service is invalid because it was made from kernel mode.

SS$_NOHANDLER

The specified exit handler does not exist.

$CANTIM

Cancel Timer — Cancels all or a selected subset of the Set Timer requests previously issued by the current image executing in a process. Cancellation is based on the request identification specified in the Set Timer ($SETIMR) service. If you give the same request identification to more than one timer request, all requests with that request identification are canceled.

Format

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

C Prototype

```
int sys$cantim (unsigned __int64 reqidt, unsigned int acmode);
```

Arguments

reqidt

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

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

acmode

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

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

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

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

Description

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

Outstanding timer requests are automatically canceled at image exit.

Required Access or Privileges

The calling process can cancel only timer requests that are issued by a process whose access mode is equal to or less privileged than that of the calling process.

Required Quota

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

Related Services

$ASCTIM, $BINTIM, $CANWAK, $GETTIM, $GETTIM_PREC, $NUMTIM, $SCHDWK, $SETIME, $SETIMR

Condition Values Returned

SS$_NORMAL

The service completed successfully.

$CHECK_ACCESS

Check Access — Determines on behalf of a third-party user whether a named user can access the object specified.

Format

```
SYS$CHECK_ACCESS
```

```
   [objtyp], [objnam], [usrnam], itmlst, [contxt], [clsnam], [objpro],
   [usrpro]
```

C Prototype

```
int sys$check_access
   (unsigned int *objtyp, void *objnam,  void *usrnam, void *itmlst,
    unsigned int *contxt, void *clsnam, void *objpro,  void *usrpro);
```

Arguments

objtyp

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

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

Symbol	Meaning
ACL$C_CAPABILITY	Object is a restricted resource; use the reserved name VECTOR.
ACL$C_DEVICE	Object is a device.
ACL$C_FILE	Object is a Files-11 On-Disk Structure Level 2 file.
ACL$C_GROUP_GLOBAL_SECTION	Object is a group global section.
ACL$C_JOBCTL_QUEUE	Object is a batch, print, or server queue.
ACL$C_LOGICAL_NAME_TABLE	Object is a logical name table.
ACL$C_SYSTEM_GLOBAL_SECTION	Object is a system global section.

For further information about these symbols, see the description of the *clsnam* argument.

objnam

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

Name of the object being accessed. The *objnam* argument is the address of a character-string descriptor pointing to the object name.

usrnam

OpenVMS usage:	char_string

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

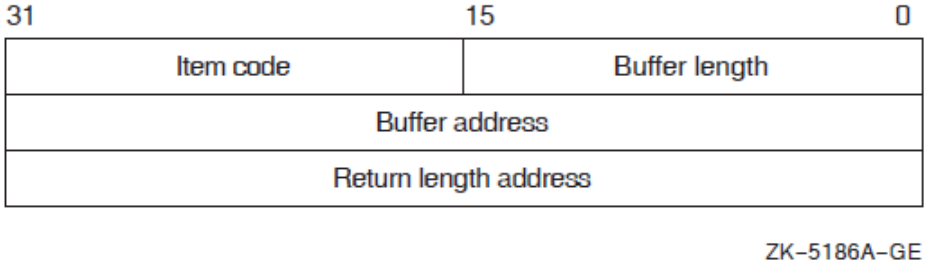
Name of the user attempting access. The *usrnam* argument is the address of a descriptor that points to a character string that contains the name of the user attempting to gain access to the specified object. The user name string can contain a maximum of 12 alphanumeric characters.

itmlst

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

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

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



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 associated buffer. The length of the buffer needed depends on the item code specified in the item code field of the item descriptor. If the value of buffer length is too small, the service truncates the data.
Item code	A word containing a user-supplied symbolic code specifying the item of information in the associated buffer.
Buffer address	A longword containing the user-supplied address of the buffer.
Return length address	A longword containing the address of a word in which $CHECK_ACCESS writes the number of bytes written to the buffer pointed to by *bufadr*. If the buffer pointed to by *bufadr* is used to pass information to $CHECK_ACCESS, *retlenadr* is ignored but must be included.

contxt

OpenVMS usage:	longword
type:	longword (unsigned)

access:	read-write
mechanism:	by reference

Longword used to maintain the user authorization file (UAF) context. The *contxt* argument is the address of a longword to receive a UAI context longword. 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.

Using the *contxt* argument keeps the UAF open across all calls, thereby improving the performance of the system on subsequent calls. To close the UAF, you must run down the image.

The resulting *contxt* value from a $CHECK_ACCESS call can also be used as the input *contxt* argument to the $GETUAI system service, and vice versa.

clsnam

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

Object class name associated with the protected object. The *clsnam* argument is the address of a descriptor pointing to the name of the object class associated with the object specified by either the *objnam* or the *objpro* argument. The *clsnam* and *objtyp* arguments are mutually exclusive. The *clsnam* argument is the preferred argument to $CHECK_ACCESS. The following object class names are valid:

CAPABILITY	QUEUE
COMMON_EVENT_CLUSTER	RESOURCE_DOMAIN
DEVICE	SECURITY_CLASS
FILE	SYSTEM_GLOBAL_SECTION
GROUP_GLOBAL_SECTION	VOLUME
LOGICAL_NAME_TABLE	

objpro

OpenVMS usage:	char_string
type:	opaque byte stream or object handle
access:	read only
mechanism:	by descriptor

Buffer containing an object security profile or object handle. The *objpro* argument is the address of a descriptor pointing to a buffer that contains an encoded object security profile or the address of a descriptor pointing to an object handle.

Object handles vary according to the associated security object class. Currently, the only supported object handles are for the file and device class objects where the object handle is a word or longword channel.

The *objpro* and *objnam* arguments are mutually exclusive unless the *objpro* argument is a simple object handle. The *objpro* and *usrpro* arguments are also mutually exclusive unless the *objpro* argument is an object handle.

usrpro

OpenVMS usage:	char_string
type:	opaque byte stream
access:	read only
mechanism:	by descriptor

Buffer containing a user security profile. The *usrpro* argument is the address of a descriptor pointing to a buffer that contains an encoded user security profile.

The $CREATE_USER_PROFILE service can be used to construct a user security profile. The *usrpro* and *usrnam* arguments are mutually exclusive. The *objpro* and *usrpro* arguments are also mutually exclusive unless the *objpro* argument is an object handle.

The item codes used with $CHECK_ACCESS are described in the following list and are defined in the $CHPDEF system macro library.

Item Codes

CHP$_ACCESS

A longword bit mask that represents the desired access ($ARMDEF). Only those bits set in CHP$_ACCESS are checked against the protection of the object to determine whether access is granted.

The default for CHP$_ACCESS is read. Symbolic representations for the access types associated with the built-in protected classes are found in the $ARMDEF macro.

For example, ARM$M_MANAGE specifies Manage access for the queue class object. Access type names are object class specific and vary from class to class. Because $CHECK_ACCESS performs only a bitwise comparison of access desired to object protection, the original Read, Write, Execute, and Delete names can also be used to specify the first four access types for any object class.

The following table shows the access types available and lists their common interpretations. These symbols are defined in the $ARMDEF system macro library. For more information, see the *VSI OpenVMS Guide to System Security*.

Access Type	Access Permitted
ARM$M_READ	Allows holders to read an object, perform wildcard directory lookups, display jobs in a queue, or use an associated vector processor.
ARM$M_WRITE	Allows holders to alter the contents of an object, remove a directory entry, write or extend existing files on a volume, or submit a job to a queue.
ARM$M_EXECUTE	Allows holders to run an image or command procedure, perform exact directory lookups, issue physical I/O requests to a device, create new files on a volume, or act as operator for a queue.
ARM$M_DELETE	Allows holders to delete an object, perform logical I/O to a device, or delete a job in a queue.

Access Type	Access Permitted
ARM$M_CONTROL	Allows holders to display or alter the security characteristics of an object.

CHP$_ACMODE

A byte that defines the accessor's processor access mode ($PSLDEF). The following access modes and their symbols are defined in the system macro library ($PSLDEF). Objects supported by the operating system do not consider access mode in determining object access.

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

If CHP$_ACMODE is not specified, access mode is not used to determine access.

CHP$_ALARMNAME

Address of a buffer to receive the alarm name from any Alarm ACE contained in the object's ACL. Currently, if a matching Alarm ACE exists, the string SECURITY will be returned. The string returned by CHP$_ALARMNAME can be used as input to the $AUDIT_EVENT system service, using the NSA$_ALARM_NAME item code.

CHP$_AUDIT_LIST

A list containing information to be added to any resulting security audit. The *bufadr* argument points to the beginning of an $AUDIT_EVENT item list. See the *itmlst* argument of the $AUDIT_EVENT system service for a list of valid security auditing item codes. Note that the NSA$_EVENT_TYPE and NSA$_EVENT_SUBTYPE items are ignored when auditing with $CHECK_ACCESS. The CHP$V_AUDIT flag must be specified.

CHP$_AUDITNAME

Address of a buffer to receive the audit name from any Audit ACE contained in the object's ACL. Currently, if a matching Audit ACE exists, the string SECURITY will be returned. The string returned by CHP$_AUDITNAME can be used as input to the $AUDIT_EVENT system service, using the NSA$_AUDIT_NAME item code.

CHP$_FLAG

A longword that controls various aspects of the protection check. The symbols in the following table are offsets to the bits within the longword. You can also obtain the values as masks with the appropriate bit set by using the prefix CHP$M rather than CHP$V. These symbols are defined in the system macro library ($CHPDEF).

Symbol	Access
CHP$V_ALTER	Accessor desires write access to object.
CHP$V_AUDIT	Access audit requested.

Symbol	Access
CHP$V_CREATE	Perform the audit as an object creation event.
CHP$V_DELETE	Perform the audit as an object deletion event.
CHP$V_FLUSH	Force audit buffer flush.
CHP$V_INTERNAL	Audit on behalf of the Trusted Computing Base (TCB). Reserved to OpenVMS.
CHP$V_MANDATORY	Force the object access event to be audited.
CHP$V_NOFAILAUD	Do not perform audits for failed access.
CHP$V_NOSUCCAUD	Do not perform audits for successful access.
CHP$V_OBSERVE	Accessor desires read access to object.
CHP$V_SERVER	Audit on behalf of a TCB server process.
CHP$V_USEREADALL	Accessor is eligible for READALL privilege.

The default for CHP$_FLAG is CHP$V_OBSERVE.

The primary purpose of the CHP$V_OBSERVE and CHP$V_ALTER flags is as latent support for a mandatory (lattice) security policy, such as that provided by the Security Enhanced VMS (SEVMS) offering.

CHP$_MATCHEDACE

A variable-length data structure containing the first Identifier ACE in the ACL that granted or denied access to the object. The $FORMAT_ACL system service describes the format of an Identifier ACE.

CHP$_PRIVUSED

A longword mask of flags that represent the privileges used to gain access.

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

Symbol	Meaning
CHP$V_SYSPRV	SYSPRV was used to gain the requested access.
CHP$V_GRPPRV	GRPPRV was used to gain the requested access.
CHP$V_BYPASS	BYPASS was used to gain the requested access.
CHP$V_READALL	READALL was used to gain the requested access.
CHP$V_OPER	OPER was used to gain the requested access.
CHP$V_GRPNAM	GRPNAM was used to gain the requested access.
CHP$V_SYSNAM	SYSNAM was used to gain the requested access.
CHP$V_GROUP	GROUP was used to gain the requested access.
CHP$V_WORLD	WORLD was used to gain the requested access.
CHP$V_PRMCEB	PRMCEB was used to gain the requested access.
CHP$V_UPGRADE	UPGRADE was used to gain the requested access.
CHP$V_DOWNGRADE	DOWNGRADE was used to gain the requested access.

Description

The Check Access service invokes the operating system control protection check mechanism, $CHKPRO, to determine whether a named user is allowed the described access to the named object. A file server, for example, might check the access attributes of a user who attempts to access a file (the object).

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

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

The caller can also request that an object access audit be performed if security auditing has been enabled for the object class or if Audit ACEs are contained in the object's ACL. Auditing ACEs include both Alarm ACEs and Audit ACEs. The CHP$V_AUDIT flag requests an access audit. This requires that the caller be in executive or kernel mode or possess the AUDIT privilege.

Normally, $CHECK_ACCESS generates an object access audit when an audit is required. The caller can specify the CHP$V_CREATE flag to force an object creation audit instead of an object access audit. Similarly, the CHP$V_DELETE flag forces an object deletion audit. The CHP$_AUDIT_LIST item code can be used to specify additional information to be included in any resulting audit records.

With certain types of devices, $CHECK_ACCESS can return a false negative, but never a false positive. This is due to additional LOG_IO and PHY_IO privilege checking in the $QIO system service that might override an otherwise unsuccessful access attempt. These privilege checks are not mirrored by the $CHECK_ACCESS system service. The affected devices are those that are non-file-structured or mounted foreign and also either spooled, file-oriented, or shareable. For example, mailbox devices fall into this category because they are non-file-structured and shareable. To accurately duplicate the result that would be obtained if the user had issued a read or write against these devices, it might be necessary to test for these additional privileges using the $CHECK_PRIVILEGE system service. For more information about access requirements for devices, see the *VSI OpenVMS I/O User's Reference Manual*.

Required Access or Privileges

Access to SYSUAF.DAT and RIGHTSLIST.DAT is required. AUDIT privilege is required when requesting a user mode audit.

Required Quota

None

Related Services

$CHKPRO, $CREATE_USER_PROFILE, $FORMAT_ACL

Condition Values Returned

SS$_NORMAL

The service completed successfully; the desired access is granted.

SS$_ACCVIO

The item list cannot be read by the caller, one of the buffers specified in the item list cannot be written by the caller, or one of the arguments could not be read or written.

SS$_BADPARAM

Invalid or conflicting combination of parameters.

SS$_INSFARG

Insufficient information to identify object or user.

SS$_INSFMEM

Insufficient process memory to execute service.

SS$_NOAUDIT

Caller lacks privilege to request audit.

SS$_NOCALLPRIV

Caller lacks privilege to access authorization database.

SS$_NOCLASS

No matching object class was located.

SS$_NOPRIV

The desired access is not granted.

SS$_UNSUPPORTED

Operations on remote object are not supported.

If CHP$V_AUDIT is specified, any error from the $AUDIT_EVENT system service can also be returned.

$CHECK_FEN (Alpha and Integrity servers)

Check Floating Point — On Alpha and Integrity server systems, indicates whether floating point is enabled for the current image.

Format

```
SYS$CHECK_FEN [flags]
```

C Prototype

```
int sys$check_fen (unsigned int *flags);
```

Arguments

flags

OpenVMS usage:	mask longword
type:	longword (unsigned)
access:	write only
mechanism:	by 32- or 64-bit reference (Alpha and Integrity servers)

For architectures that have multiple floating-point resources that can be enabled separately, this longword is returned with a bitmask indicating which resources are enabled. On Alpha systems, no separate resources exist; nothing is returned. On Integrity server systems, the bitmask has two bits: bit 0 for the low floating-point bank and bit 1 for the high floating-point bank.

Description

The Check Floating Point service returns a Boolean value in R0 indicating whether any floating point resources are enabled for the current image.

The $CHECK_FEN service returns a value of 1 if the floating point is enabled for the current image. A value of 0 is returned if the floating point is disabled.

An optional longword, passed by reference, can be specified to receive architecture-dependent information about the floating-point resources in use.

Required Access or Privileges

None

Required Quota

None

$CHECK_PRIVILEGE

Check Privilege — Determines whether the caller has the specified privileges or identifier. In addition to checking for a privilege or an identifier, $CHECK_PRIVILEGE determines if the caller's use of privilege needs to be audited.

Format

```
SYS$CHECK_PRIVILEGE
   [efn] ,prvadr ,[altprv] ,[flags]  ,[itmlst] ,[audsts] ,[astadr]
  ,[astprm]
```

C Prototype

```
int sys$check_privilege
   (unsigned int efn, struct _generic_64  *prvadr,
```

```
struct _generic_64 *altprv, unsigned int flags,
void *itmlst,  unsigned int *audsts,
void (*astadr)(__unknown_params), int 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 audit completes. The *efn* argument is a longword containing the number of the event flag; however, $CHECK_PRIVILEGE uses only the low-order byte. If *efn* is not specified, event flag 0 is used.

Upon request initiation, $CHECK_PRIVILEGE clears the specified event flag.

prvadr

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

The privilege, privileges, or identifier that the calling process must possess.

The *prvadr* argument is either the address of a quadword bit array, where each bit corresponds to a privilege, or the address of a quadword identifier.

When the array lists privileges, each bit has a symbolic name. The $PRVDEF macro defines these names. You form the bit array by specifying the symbolic name of each desired privilege in a logical OR operation. See the $SETPRV system service for the symbolic name and description of each privilege.

If the caller passes an identifier, the caller must set the NSA$M_IDENTIFIER bit in the *flags* longword. The identifier structure is defined by the $KGBDEF macro. The identifier attributes (KGB$) are reserved for future use and should be set to 0.

altprv

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

Alternate privilege mask to check against. The *altprv* argument is the address of a quadword privilege mask, where each bit corresponds to a privilege. This argument and the flags NSA$M_AUTHPRIV, NSA$M_IDENTIFIER, and NSA$M_PROCPRIV are mutually exclusive.

With this argument, $CHECK_PRIVILEGE uses the supplied set of privileges instead of the current, active privileges. Each bit in the mask has a symbolic name, defined by the $PRVDEF macro. You form the bit array by specifying the symbolic name of each desired privilege in a logical OR operation. See the $SETPRV system service for the symbolic name and description of each privilege.

flags

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

Flags that specify options for the $CHECK_PRIVILEGE operation. The *flags* argument is a longword bit mask, where each bit corresponds to an option.

Each flag option has a symbolic name. The $NSADEF macro defines the following symbolic names. Be aware that the flags NSA$M_AUTHPRIV, NSA$M_IDENTIFIER, and NSA$M_PROCPRIV are mutually exclusive; therefore, you can specify only one of these flag options.

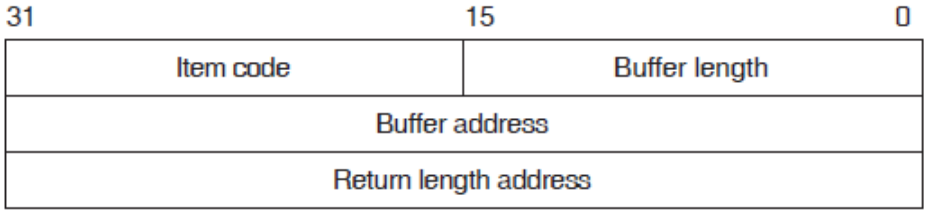
Symbolic Name	Description
NSA$M_AUTHPRIV	Checks the authorized privileges of the process instead of the current (active) privileges.
NSA$M_FLUSH	Specifies that all messages in the audit server buffer be written to the audit log file.
NSA$M_IDENTIFIER	Interprets the *prvadr* argument as the address of an identifier instead of a privilege mask.
NSA$M_INTERNAL	Specifies that the $CHECK_PRIVILEGE call originates in the context of a trusted computing base (TCB) component. The auditing components use this flag to indicate that internal auditing failures should result in a SECAUDTCB bugcheck. This flag is reserved to OpenVMS.
NSA$M_MANDATORY	Specifies that an audit is to be performed, regardless of system alarm and audit settings.
NSA$M_PROCPRIV	Checks the permanent privileges of the process, instead of the privileges in the current (active) mask.
NSA$M_SERVER	Indicates that the call originates in a TCB server process and that the event should be audited regardless of the state of a process-specific no-audit bit. Trusted servers use this flag to override the no-audit bit when they want to perform explicit auditing on behalf of a client process. This flag is reserved to OpenVMS.

itmlst

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

Item list specifying additional security auditing information to be included in any security audit that is generated by the service. 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 item list is a standard format item list. The following diagram depicts the format of a single item descriptor.



The following table defines the item descriptor fields.

Descriptor Field	Definition
Buffer length	A word specifying the length of the buffer in bytes. The buffer supplies information to be used by $CHECK_PRIVILEGE. The required length of the buffer varies, depending on the item code specified; each item code description specifies the required length.
Item code	A word containing a symbolic code describing the nature of the information currently in the buffer or to be returned in the buffer. The location of the buffer is pointed to by the buffer address field. Each item code has a symbolic name.
Buffer address	A longword containing the address of the buffer that specifies or receives the information.
Return length address	Not currently used; this field is reserved to OpenVMS. You should specify 0.

All item codes listed in the Item Codes section of the $AUDIT_EVENT service are valid within the item list used by the $CHECK_PRIVILEGE service except for the NSA$_EVENT_TYPE and NSA$_EVENT_SUBTYPE item codes, which are supplied internally by the $CHECK_PRIVILEGE service.

$CHECK_PRIVILEGE should be called with an item list identifying the alarm and audit journals, and does not need to use the NSA$_PRIVS_USED item code. NSA$_PRIVS_USED is supplied automatically by the $CHECK_PRIVILEGE service. Note that $CHECK_PRIVILEGE returns SS$_BADPARAM if you supply either NSA$_EVENT_TYPE or NSA$_EVENT_SUBTYPE. These items are supplied internally by $CHECK_PRIVILEGE.

audsts

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

Longword condition value that receives a final completion status from the operation. If a security audit is required, the final completion status represents either the successful completion of the resulting security audit or any failing status that occurred while the security audit was performed within the AUDIT_SERVER process.

The *audsts* argument is valid only when the service returns success and the status is not SS$_EVTNOTENAB. In addition, the caller must either make use of the *astadr* argument or use the $CHECK_PRIVILEGEW service before attempting to access *audsts*.

astadr

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

Asynchronous system trap (AST) routine to be executed after the *audsts* argument is written. The *astadr* argument, which is the address of a longword value, is the procedure value of the AST routine.

The AST routine executes in the access mode of the caller of $CHECK_PRIVILEGE.

astprm

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

Asynchronous system trap (AST) parameter passed to the AST service routine. The *astprm* argument is a longword value containing the AST parameter.

Description

The Check Privilege service determines whether a user has the privileges or identifier that an operation requires. In addition, $CHECK_PRIVILEGE audits the use of privilege if privilege auditing has been enabled by the site security administrator. The caller does not need to determine whether privilege auditing has been enabled.

Required Access or Privileges

AUDIT privilege is required.

Required Quota

None

Related Services

$AUDIT_EVENT, $SETPRV

Condition Values Returned

SS$_NORMAL

The service completed successfully.

SS$_ACCVIO

The specified parameter of the item list buffer is not accessible.

SS$_BADBUFADR

The buffer address is invalid or not readable.

SS$_BADBUFLEN

The specified buffer length is invalid or out of range.

SS$_BADCHAIN

The address of the next item list to be processed, as identified in the buffer address field, is either not readable or points to itself.

SS$_BADITMCOD

The specified item code is invalid or out of range.

SS$_BADPARAM

The specified list entry is invalid or out of range.

SS$_EVTNOTENAB

No audit required; privilege granted.

SS$_ILLEFC

You specified an illegal event flag number.

SS$_INSFARG

The argument list contains too few arguments for the service.

SS$_INVAJLNAM

The alarm or audit journal name is invalid.

SS$_IVSTSFLG

The specified system service flags are invalid.

SS$_NOAUDIT

The caller does not have the required privilege to perform the audit.

SS$_NOPRIV

The subject does not have the required privileges or identifier.

SS$_NO[privilege-name]

The subject does not have a specific privilege.

SS$_OVRMAXAUD

There is insufficient memory to perform the audit.

SS$_TOOMANYAJL

Too many alarm or audit journals were specified.

SS$_UNASEFC

An unassociated event flag cluster was specified.

$CHECK_PRIVILEGEW

Check Privilege and Wait — Determines whether the caller has the specified privileges or identifier. In addition to checking for a privilege or an identifier, the Check Privilege and Wait service determines if the caller's use of privilege needs to be audited. $CHECK_PRIVILEGEW completes synchronously; that is, it returns the final status to the caller only after receiving an explicit confirmation from the audit server that the associated audit, if enabled, has been performed.

Format

```
SYS$CHECK_PRIVILEGEW
   efn ,prvadr ,[altprv] ,[flags]  ,[itmlst] ,audsts ,[astadr] ,[astprm]
```

C Prototype

```
int sys$check_privilegew
   (unsigned int efn, struct _generic_64  *prvadr, struct _generic_64
 *altprv,
    unsigned int flags, void *itmlst,  unsigned int *audsts,
    void (*astadr)(__unknown_params), int astprm);
```

$CHKPRO

Check Access Protection — Determines whether an accessor with the specified rights and privileges can access an object with the specified attributes.

Format

```
SYS$CHKPRO itmlst ,[objpro] ,[usrpro]
```

C Prototype

```
int sys$chkpro (void *itmlst, void *objpro, void *usrpro);
```

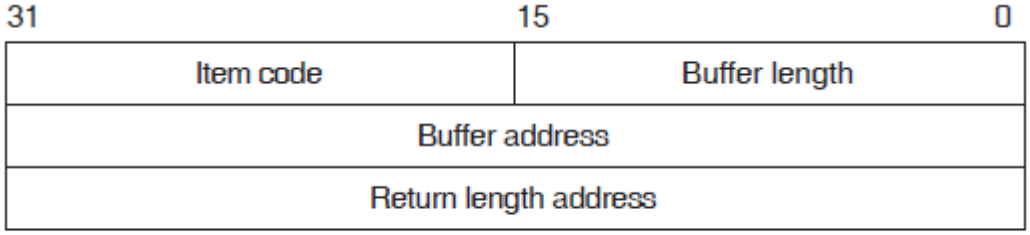
Argument

itmlst

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

Protection attributes of the object and the rights and privileges of the accessor. The *itmlst* argument is the address of an item list of descriptors used to specify the protection attributes of the object and the rights and privileges of the accessor.

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



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 associated buffer. The length of the buffer needed depends on the item code specified in the item code field of the item descriptor. If the value of buffer length is too small, the service truncates the data.
Item code	A word containing a user-supplied symbolic code specifying the item of information in the associated buffer. The item codes are defined in the $ACLDEF system macro library.
Buffer address	A longword containing the user-supplied address of the buffer.
Return length address	A longword that normally contains the user-supplied address of a word in which the service writes the length in bytes of the information it returned. This is not used by $CHKPRO and should contain a 0.

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

objpro

OpenVMS usage:	char_string
type:	opaque byte stream
access:	read only
mechanism:	by descriptor

Buffer containing an object security profile. The *objpro* argument is the address of a descriptor pointing to a buffer that contains an encoded object security profile. The *objpro* argument eliminates

the need to supply all of the component object protection attributes with the $CHKPRO item list. The *objpro* argument is currently reserved to OpenVMS.

usrpro

OpenVMS usage:	char_string
type:	opaque byte stream
access:	read only
mechanism:	by descriptor

Buffer containing a user security profile. The *usrpro* argument is the address of a descriptor pointing to a buffer that contains an encoded user security profile. The *usrpro* argument eliminates the need to supply all of the component user security attributes with the $CHKPRO item list.

The $CREATE_USER_PROFILE service can be used to construct a user security profile. When the *usrpro* argument is specified, any component user profile attributes specified in the $CHKPRO item list replace those contained in the user security profile.

The item codes used with $CHKPRO are described in the following list and are defined in the $CHPDEF system macro library.

Item Codes

CHP$_ACCESS

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

CHP$_ACL

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

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

CHP$_ACMODE

A byte that defines the accessor's processor access mode. The following access modes and their symbols are defined in the $PSLDEF macro.

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

Symbol	Access Mode
PSL$C_KERNEL	Kernel

If CHP$_ACMODE is not specified, access mode is not used to determine access.

CHP$_ADDRIGHTS

A vector that points to an additional rights list segment to be appended to the existing rights list. Each entry of the rights list is a quadword data structure consisting of a longword containing the identifier value, followed by a longword containing a mask identifying the attributes of the holder. The $CHKPRO service ignores the attributes.

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

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

- CHP$_RIGHTS must come first.
- The accessor's UIC is the identifier of the first entry in the rights list specified by the CHP$_RIGHTS item code.
- The accessor's rights list consists of the rights list specified by the CHP$_RIGHTS item code and the CHP$_ADDRIGHTS item codes.

CHP$_ALARMNAME

Address of a buffer to receive the alarm name from any Alarm ACE contained in the object's ACL. If the object does not have security alarms enabled, $CHKPRO returns *retlenadr* as 0. If a matching Alarm ACE exists, the string SECURITY will be returned.

CHP$_AUDIT_LIST

A security auditing item list containing additional information to be included in any resulting security audit. The *bufadr* argument points to the beginning of an $AUDIT_EVENT item list. See the *itmlst* argument of the $AUDIT_EVENT system service for a list of valid security auditing item codes. Note that the NSA$_EVENT_TYPE and NSA$_EVENT_SUBTYPE items are ignored when auditing with $CHKPRO. The CHP$V_AUDIT flag must be specified.

CHP$_AUDITNAME

Address of a buffer to receive the audit name from any Audit ACE contained in the object's ACL. If the object does not have auditing enabled, $CHKPRO returns *retlenadr* as 0. If a matching Audit ACE exists, the string SECURITY will be returned.

CHP$_FLAGS

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

Symbol	Access
CHP$V_ALTER	Accessor desires write access to object.

Symbol	Access
CHP$V_AUDIT	Access audit requested.
CHP$V_CREATE	Perform the audit as an object creation event.
CHP$V_DELETE	Perform the audit as an object deletion event.
CHP$V_FLUSH	Force audit buffer flush.
CHP$V_INTERNAL	Audit on behalf of the Trusted Computing Base (TCB). Reserved to OpenVMS.
CHP$V_MANDATORY	Force the object access event to be audited.
CHP$V_NOFAILAUD	Do not perform audits for failed access.
CHP$V_NOSUCCAUD	Do not perform audits for successful access.
CHP$V_OBSERVE	Accessor desires read access to object.
CHP$V_SERVER	Audit on behalf of a TCB server process.
CHP$V_USEREADALL	Accessor is eligible for READALL privilege.

The default for CHP$_FLAG is CHP$M_OBSERVE and CHP$M_ALTER.

The primary purpose of the CHP$V_OBSERVE and CHP$V_ALTER flags is as latent support for a mandatory (lattice) security policy, such as that provided by the Security Enhanced VMS (SEVMS) offering.

CHP$_MATCHEDACE

This output item is a variable-length data structure containing the first Identifier ACE in the object's ACL that allowed or denied the accessor to access the object. See the $FORMAT_ACL system service for a description of an Identifier ACE format.

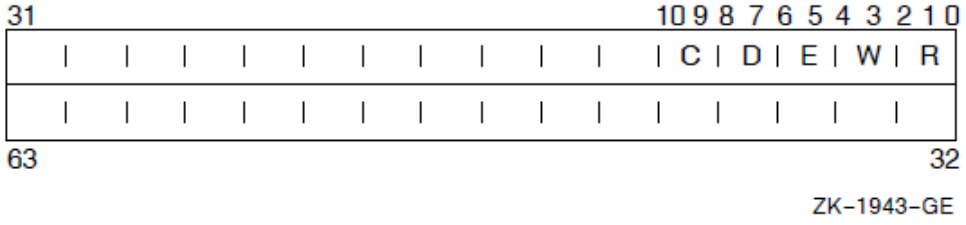
CHP$_MODE

A byte that defines the object's owner access mode. The following access modes of the object's owner and their symbols are defined in the system macro library ($PSLDEF).

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

CHP$_MODES

A quadword that defines the object's access mode protection. You specify a 2-bit access mode as shown in CHP$_MODE for each possible access type. The following diagram illustrates the format of an access mode vector for bit numbers.



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

CHP$_OBJECT_CLASS

A character string containing the protected object class associated with the object. The object class string is used to determine whether any security auditing is enabled for the object access event. This item code is required when the CHP$_AUDIT flag is specified.

CHP$_OBJECT_NAME

A character string containing the object name associated with the protection check. The object name string is included in any resulting security audit. If an object name string is not specified, the string "<not available>" is substituted in any security audit for all protected object classes other than FILE. For FILE class audits, it is assumed that the caller has supplied an object name by using the auditing item list (NSA$_OBJECT_NAME).

CHP$_OWNER

A longword describing the object's owner identifier (UIC or general identifier). This might be either a UIC format identifier or a general identifier.

Note

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

CHP$_PRIV

A quadword that defines an accessor's privilege mask. Each bit in the mask has a symbolic name, defined by the $PRVDEF macro. You form the bit array by specifying the symbolic name of each privilege in a logical OR operation. See the $SETPRV system service for the symbolic name and description of each privilege.

CHP$_PRIVUSED

A longword mask of flags representing privileges used to gain the requested access.

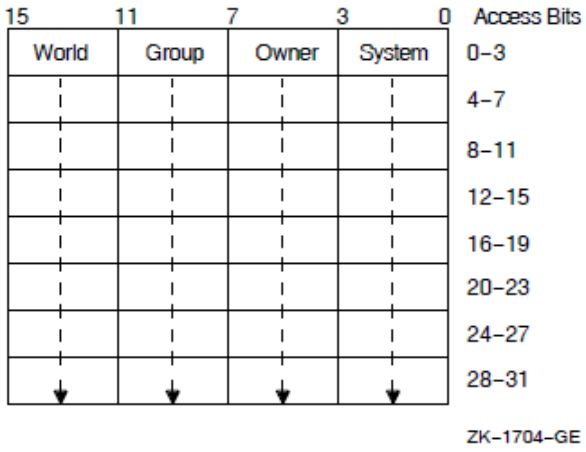
You can also obtain the values as masks with the appropriate bit set by using the prefix CHP$M rather than CHP$V. The symbols are defined in the system macro library ($CHPDEF). The following symbols are used as offsets to the bits within the longword.

Symbol	Meaning
CHP$V_SYSPRV	SYSPRV was used to gain the requested access.
CHP$V_GRPPRV	GRPPRV was used to gain the requested access.
CHP$V_BYPASS	BYPASS was used to gain the requested access.
CHP$V_READALL	READALL was used to gain the requested access.
CHP$V_OPER	OPER was used to gain the requested access.
CHP$V_GRPNAM	GRPNAM was used to gain the requested access.
CHP$V_SYSNAM	SYSNAM was used to gain the requested access.

Symbol	Meaning
CHP$V_GROUP	GROUP was used to gain the requested access.
CHP$V_WORLD	WORLD was used to gain the requested access.
CHP$V_PRMCEB	PRMCEB was used to gain the requested access.
CHP$V_UPGRADE	UPGRADE was used to gain the requested access.
CHP$V_DOWNGRADE	DOWNGRADE was used to gain the requested access.

CHP$_PROT

A vector describing the object's SOGW protection mask. The following diagram depicts the format for describing the object's protection.



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

The $CHKPRO service compares the low-order four bits of CHP$_ACCESS against one of the 4-bit fields in the low-order word of CHP$_PROT, the next four bits of CHP$_ACCESS against one of the 4-bit fields in the next word of CHP$_PROT, and so on. The $CHKPRO service chooses a field of CHP$_PROT based on the privileges specified for the accessor (CHP$_PRIV), the UICs of the accessor (CHP$_RIGHTS or CHP$_ADDRIGHTS, or both), and the object's owner (CHP$_OWNER).

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

CHP$_RIGHTS

A vector that points to an accessor's rights list. The accessor's UIC is the identifier of the first entry in the rights list. The accessor's rights list consists of the rights list specified by CHP$_RIGHTS and, optionally, the rights list specified by the CHP$_ADDRIGHTS item codes.

CHP$_UIC

A longword specifying the accessor's owner UIC. This item code can be used to avoid having to pass an entire rights list segment via the CHP$_RIGHTS item code. If CHP$_RIGHTS and then CHP$_UIC are specified, in that order, $CHKPRO initializes the local rights list and then replaces just the owner UIC with the value of CHP$_UIC.

Description

The Check Access Protection service determines whether an accessor with the specified rights and privileges can access an object with the specified attributes. The service invokes the system's access protection check, which permits layered products and other subsystems to build protected structures that are consistent with the protection facilities provided by the base operating system. The service also allows a privileged subsystem to perform protection checks on behalf of a requester.

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

The item list arguments accepted by this service permit you to specify the protection of the object being accessed, the rights and privileges of the accessor, and the type of access desired.

At a minimum, the following item codes should be specified to perform a third-party protection check:

CHP$_ACCESS
CHP$_OWNER
CHP$_PRIV
CHP$_PROT
CHP$_UIC

The default for information relating to the subject is to use the current process information (for example, privileges). The default for missing object information is a representation of 0.

The caller can also request that an object access audit be performed if security auditing has been enabled for the object class or if auditing ACEs are contained in the object's ACL. The CHP$V_AUDIT flag requests an access audit. This requires that the caller be in executive or kernel mode or possess the AUDIT privilege.

Usually, $CHKPRO generates an object access audit when an audit is required. The caller can specify the CHP$V_CREATE flag to force an object creation audit instead of an object access audit. Similarly, the CHP$V_DELETE flag forces an object deletion audit. The CHP$_AUDIT_LIST item code can be used to specify additional information to be included in any resulting audit records.

Required Access or Privileges

AUDIT privilege is required when requesting an audit.

Required Quota

None

Related Services

$AUDIT_EVENT, $CHECK_ACCESS, $CREATE_USER_PROFILE, $FORMAT_ACL

Condition Values Returned

SS$_NORMAL

The service completed successfully; the desired access is granted.

SS$_ACCVIO

The item list cannot be read by the caller, or one of the buffers specified in the item list cannot be written by the caller.

SS$_ACLFULL

More than 20 CHP$_ACL items were given.

SS$_BADPARAM

The argument is invalid.

SS$_BUFFEROVF

The output buffer is too small and the protection check succeeded.

SS$_IVACL

You supplied an invalid ACL segment with the CHP$_ACL item.

SS$_IVBUFLEN

The output buffer is too small and the protection check failed.

SS$_NOAUDIT

Caller lacks privilege to request audit.

SS$_NOPRIV

The desired access is not granted.

SS$_RIGHTSFULL

More than 11 CHP$_ADDRIGHTS items were given.

$CLEAR_SYSTEM_EVENT (Alpha and Integrity servers)

Clear System Event — Removes one or more notification requests previously established by a call to $SET_SYSTEM_EVENT. This service does not allow you to specify a handle and an event. You must pass a zero as one of these parameters. You can either clear by handle or request that all events for the user be cleared.

Format

```
SYS$CLEAR_SYSTEM_EVENT [handle] ,[acmode] ,event
```

C Prototype

```
nt sys$clear_system_event
```

```
(struct _generic_64 * handle, unsigned int acmode, unsigned int event);
```

Arguments

handle

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

Identification of the AST request to be cleared. The *handle* argument uniquely identifies the request and is returned when the $SET_SYSTEM_EVENT service is called. The *handle* argument may be omitted by specifying a zero address.

acmode

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

Access mode of the system event to be cleared. The *acmode* argument is a longword containing the access mode. The value of the access mode is maximized with the access mode of the caller.

event

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

The *event* argument is a value indicating the type of system event to be cleared. SYSEVT$C_ALL_EVENTS may be specified to clear all event types.

Description

The Clear System Event service removes one or more event types or notification objects previously established by a call to the $SET_SYSTEM_EVENT service.

A valid request specifies either the *handle* for a specific notification request, or is a wildcard clear of all notification objects whose is access mode is greater than or equal to *acmode*.

If the *handle* argument is specified, caller's access mode must be less than or equal to the access mode of the object to be cleared.

If SYSEVT$C_ALL_EVENTS is specified, or the set of events enabled for the object(s) becomes empty, the notification object is deleted.

Required Access or Privileges

None

Required Quota

None

Related Services

$SET_SYSTEM_EVENT

Condition Values Returned

SS$_NORMAL

The service completed successfully.

SS$_ACCVIO

The service cannot access the location specified by the handle.

SS$_BADPARAM

One or more arguments has an invalid value, such as an invalid handle.

SS$_NOSUCHOBJ

No request was found that matches the description supplied.

$CLEAR_UNWIND_TABLE (Integrity servers Only)

Clear Unwind Table Routine — Clears unwind table (UT) information.

Format

```
SYS$CLEAR_UNWIND_TABLE code_base_va
```

C Prototype

```
int SYS$CLEAR_UNWIND_TABLE (unsigned __int64 code_base_va);
```

Arguments

code_base_va

OpenVMS usage:	address

type:	quadword (unsigned)
access:	read only
mechanism:	by value

Input by value. Must be the process virtual address of the start of a registered code range.

Description

Clears (removes) the indicated registration. Error status returned on bad *code_base_va* or insufficient access mode.

Required Access or Privileges

The unwind table information that corresponds to code_base_va will be removed only if it was registered in a mode equal to or less privileged than the caller of $CLEAR_UNWIND_TABLE.

Required Quota

None

Related Services

SYSSET_UNWIND_TABLE, SYSGET_UNWIND_ENTRY_INFO. Also see LIB$GET_UIB_INFO in *VSI OpenVMS Calling Standard*.

Condition Values Returned

SS$_NORMAL

Routine completed successfully.

SS$_IVAADDR

code_base_va not registered.

SS$_IVACMODE

Insufficient access mode.

$CLOSE

Closes File — The Close service terminates file processing and closes the file. This service performs an implicit Disconnect service for all record streams associated with the file. For additional information about this service, see the *VSI OpenVMS Record Management Services Reference Manual*.

$CLRAST

Clear AST — Clears the "AST active" status. This enables delivery of asynchronous system traps (ASTs) for the access mode from which the service call was issued, while an AST routine is active.

Format

```
SYS$CLRAST
```

Arguments

None.

Description

Note

The explicit use of $CLRAST is strongly discouraged, as it complicates synchronization issues and may lead to the unbounded consumption of stack space.

Normally, AST delivery for a particular access mode is deferred while an AST routine is executing in that access mode. When the AST routine returns, an implicit call is made to $CLRAST to re-enable AST delivery.

Explicitly calling $CLRAST within an AST routine allows the delivery of ASTs for the access mode from which the service call was issued, prior to completion of the active AST routine.

Required Access or Privileges

None

Required Quota

None

Related Services

$SETAST

Condition Values Returned

None

The return value is undefined.

$CLRCLUEVT

Clear Cluster Event — Removes one or more notification requests previously established by a call to SYS$SETCLUEVT.

Format

```
SYS$CLRCLUEVT [handle] ,[acmode] ,[event]
```

C Prototype

```
int sys$clrcluevt
   (struct _cluevthndl *handle, unsigned int acmode,
    unsigned int event);
```

Arguments

handle

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

Identification of the AST request to be canceled. The *handle* argument uniquely identifies the request and is returned when the $SETCLUEVT service is called.

acmode

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

Access mode of the cluster configuration event to be canceled. 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 types.

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 cluster configuration event for which an AST is no longer to be delivered. The *event* argument is a value indicating which type of event is no longer of interest.

Each event type has a symbolic name. The $CLUEVTDEF macro defines the following symbolic names.

Symbolic Name	Description
CLUEVT$C_ADD	One or more OpenVMS nodes have been added to the OpenVMS Cluster system.
CLUEVT$C_REMOVE	One or more OpenVMS nodes have been removed from the OpenVMS Cluster system.

Description

The Clear Cluster Event service removes one or more notification requests previously established by a call to the $SETCLUEVT service. $CLRCLUEVT verifies that the parameters specify a valid request, and dequeues and deallocates the request.

A valid request specifies either the *handle* argument or the *event* argument. If the *handle* argument is specified, the *acmode* argument must match the value recorded when $SETCLUEVT was called. If the *event* argument is specified, all requests matching the access mode are canceled, provided that the access mode is not greater than the caller's mode. If the access mode parameter is more privileged than the mode of the caller, the mode of the caller will be used.

Required Access or Privileges

None

Required Quota

None

Related Services

$SETCLUEVT, $TSTCLUEVT

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.

$CLREF

Clear Event Flag — Clears (sets to 0) an event flag in a local or common event flag cluster.

Format

```
SYS$CLREF efn
```

C Prototype

```
int sys$clref (unsigned int efn);
```

Argument

efn

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

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

Condition Values Returned

SS$_WASCLR

The service completed successfully. The specified event flag was previously 0. Note that this is also the same value as SS$_NORMAL.

SS$_WASSET

The service completed successfully. The specified event flag was previously 1. Note that while the message id is the same as SS$_ACCVIO, the severity bits are different.

SS$_ILLEFC

You specified an illegal event flag number.

SS$_UNASEFC

The process is not associated with the cluster containing the specified event flag.

$CMEXEC

Change to Executive Mode — Changes the access mode of the calling process to executive mode.

Format

```
SYS$CMEXEC routin ,[arglst]
```

C Prototype

```
int sys$cmexec (int (*routin)(__unknown_params), unsigned int *arglst);
```

Arguments

routin

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

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

arglst

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

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

If the *arglst* value is nonzero and is not accessible as an address or if the routine is inaccessible, the service returns SS$_ACCVIO.

Alpha and Integrity server systems require a pointer to a valid argument list or a value of 0 in the *arglst* argument. This means that the *arglst* argument must contain an accessible virtual address for an argument list, the first longword of which must be a valid list size.

Description

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

The $CMEXEC service uses standard procedure calling conventions to pass control to the specified routine.

To conform to the OpenVMS calling standard, you must not omit the *arglst* argument.

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

All of the Change Mode system services are intended to allow for the execution of a routine at an access mode more (not less) privileged than the access mode from which the call is made. If $CMEXEC is

called while a process is executing in kernel mode, the routine specified by the *routin* argument executes in kernel mode, not executive mode.

Required Access or Privileges

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

Required Quota

None

Related Services

None

Condition Values Returned

SS$_ACCVIO

The *arglst* or routine argument is not accessible.

SS$_BADPARAM

The routine specified is in a translated image.

SS$_NOPRIV

The process does not have the privilege to change mode to executive.

All other values

The routine executed returns all other values.

$CMEXEC_64

Change to Executive Mode with Quadword Argument List — On Alpha and Integrity server systems, changes the access mode of the calling process to executive mode. This service accepts 64-bit addresses.

Format

```
SYS$CMEXEC_64 routin_64 ,arglst_64
```

C Prototype

```
int sys$cmexec_64
   (int (*routin_64)(__unknown_params),
    unsigned __int64 *arglst_64);
```

Arguments

routin_64

OpenVMS usage:	procedure
type:	procedure value
access:	call without stack unwinding
mechanism:	by 32- or 64-bit reference

Routine to be executed while the process is in executive mode. The *routin_64* argument is the 32- or 64-bit address of this routine.

arglst_64

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

Argument list to be passed to the routine specified by the *routin_64* argument. The *arglst_64* argument is the 32- or 64-bit address of this argument list.

If the *arglst* value is nonzero and is not accessible as an address or if the routine is inaccessible, the service returns SS$_ACCVIO.

Alpha and Integrity server systems require a pointer to a valid argument list or a value of 0 in the *arglst_64* argument. This means that the *arglst_64* argument, if nonzero, must contain an accessible virtual address for an argument list, the first quadword of which must be a number between 0 and 255 specifying the number of quadwords that follow it on the list.

Description

The Change to Executive Mode with Quadword Argument List service allows a process to change its access mode to executive, execute a specified routine, and then return to the access mode in effect before the call was issued.

The $CMEXEC_64 service uses standard procedure-calling conventions to pass control to the specified routine.

When you use the $CMEXEC_64 service, the system modifies the registers before entry into the target routine. The specified routine must exit with a RET instruction.

All of the Change Mode system services are intended to allow for the execution of a routine at an access mode more (not less) privileged than the access mode from which the call is made. If $CMEXEC_64 is called while a process is executing in kernel mode, the routine specified by the *routin_64* argument executes in kernel mode, not executive mode.

Required Access or Privileges

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

Required Quota

None

Related Services

$CMEXEC, $CMKRNL, $CMKRNL_64

Condition Values Returned

SS$_ACCVIO

The *arglst* argument or routine is not accessible.

SS$_BADPARAM

The routine specified is in a translated image.

SS$_NOCMEXEC

The process does not have the privilege to change mode to executive.

All other values

The routine executed returns all other values.

$CMKRNL

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

Format

```
SYS$CMKRNL routin ,[arglst]
```

C Prototype

```
int sys$cmkrnl (int (*routin)(__unknown_params), unsigned int *arglst);
```

Arguments

routin

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

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

arglst

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

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

If the *arglst* value is nonzero and is not accessible as an address or if the routine is inaccessible, the service returns SS$_ACCVIO.

Alpha systems require a pointer to a valid argument list or a value of 0 in the *arglst* argument. This means that the *arglst* argument must contain an accessible virtual address for an argument list, the first longword of which must be a valid list size.

Description

The Change Mode to Kernel ($CMKRNL) and the Change Mode to Executive ($CMEXEC) system services provide a simple and secure path for applications to execute code in the privileged kernel and executive processor modes. These services first check for the necessary CMKRNL or CMEXEC privileges, and then call the routine specified in the argument list in the specified processor mode.

When code is executing in a privileged processor mode, such as executive or kernel mode, the code executes with full OpenVMS privileges. Furthermore, specific protection checks can also be bypassed. For example, $CMKRNL bypasses the check for CMKRNL privilege that is normally required when $CMKRNL is called from executive mode, and $SETPRV calls are processed without SETPRV privilege when called from executive or kernel mode.

The condition value returned from the procedure specified in the argument list is used as the return status from the $CMKRNL or $CMEXEC system service call. Based on the OpenVMS calling standard, this condition value is returned by register R0, using a language-specific mechanism.

Note

The $CMKRNL and $CMEXEC system services are typically used to access privileged or internal OpenVMS routines or data structures. The code to access these data structures can be OpenVMS version-dependent, particularly if the internal routines or data structures change. Errors that occur in code executing in a privileged processor mode can lead to one or more possible situations: data corruptions, process deletions, or system crashes.

The particular library routines and libraries that can be called from code executing in executive or kernel mode can also be limited, because not all library routines accessible from user mode can be called from kernel mode.

Code Example

The following code example shows how to call a specified routine in kernel mode using this service:

```
/*
```

```
//  cmkrnl.c
//
//  OpenVMS example of calling a specified routine in kernel mode,
//  using the SYS$CMKRNL system service.
//
//  Requires CMKRNL privilege.
//
//  Errors in kernel-mode code can corrupt critical data structures,
//  can cause process deletions, and can potentially crash the OpenVMS
//  operating system.
//
//  To build:
//
//    $ CC/DECC CMKRNL
//    $ LINK CMKRNL
//    $ RUN CMKRNL
*/
#include <ssdef.h>
#include <starlet.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stsdef.h>

/*
//  The KernelRoutine routine executes in kernel mode, but does
//  absolutely nothing useful.
*/
int KernelRoutine( int *UsrArg1, int *UsrArg2)
  {
  return SS$_NORMAL;
  }
main()
  {
  int RetStat;
  int ArgList[3];
  int i = 0;

  printf("OpenVMS Alpha example of calling sys$cmkrnl\n");

  /*
  //  Build the routine argument list in an array -- the KernelRoutine
  //  call expects two arguments, necessitating an array containing the
  //  count and the two arguments.
  */
  ArgList[++i] = 1;
  ArgList[++i] = 2;
  ArgList[0] = i;

  /*
  //  Now invoke the KernelRoutine in kernel mode...
  */
  RetStat = sys$cmkrnl( KernelRoutine, ArgList );
  if (!$VMS_STATUS_SUCCESS( RetStat ))
    return RetStat;

  printf("Now successfully back in user mode.\n");

  return SS$_NORMAL;
  }
```

Required Access or Privileges

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

Required Quota

None

Related Services

$CMEXEC, $CMEXEC_64, $CMKRNL_64, $SETPRV

Condition Values Returned

SS$_ACCVIO

The *arglst* argument or routine is not accessible.

SS$_BADPARAM

The routine specified is in a translated image.

SS$_NOCMKRNL

The process does not have the privilege to change mode to kernel.

All other values

The routine executed returns all other values.

$CMKRNL_64

Change to Kernel Mode with Quadword Argument List — On Alpha and Integrity server systems, changes the access mode of the calling process to kernel mode. This service allows a process to change its access mode to kernel, execute a specified routine, and then return to the access mode in effect before the call was issued. This service accepts 64-bit addresses.

Format

```
SYS$CMKRNL_64 routin_64 ,arglst_64
```

C Prototype

```
int sys$cmkrnl_64
   (int (*routin_64)(__unknown_params), unsigned __int64 *arglst_64);
```

Arguments

routin_64

OpenVMS usage:	procedure
type:	procedure value

access:	call without stack unwinding
mechanism:	by 32- or 64-bit reference

Routine to be executed while the process is in kernel mode. The *routin_64* argument is the 32- or 64-bit address of this routine.

arglst_64

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

Quadword argument list to be passed to the routine specified by the *routin_64* argument. The *routin_64* argument is the 32- or 64-bit address of this routine.

If the *arglst* value is nonzero and is not accessible as an address or if the routine is inaccessible, the service returns SS$_ACCVIO.

Alpha and Integrity server systems require a pointer to a valid argument list or a value of 0 in the *arglst_64* argument. This means that the *arglst_64* argument, if nonzero, must contain an accessible virtual address for an argument list, the first quadword of which must be a number between 0 and 255 specifying the number of quadwords that follow it on the list.

Description

The Change to Kernel Mode with Quadword Argument List service allows a process to change its access mode to kernel, execute a specified routine, and then return to the access mode in effect before the call was issued.

The $CMKRNL_64 service uses standard procedure calling conventions to pass control to the specified routine.

When you use the $CMKRNL_64 service, the system modifies the registers before entry into the target routine. The system loads R4 with the address of the process control block (PCB). The specified routine (if programmed in MACRO-32) must exit with a RET instruction.

Required Access or Privileges

To call the $CMKRNL_64 service, a process must either have CMKRNL privilege or be currently executing in executive or kernel mode.

Required Quota

None

Related Services

$CMEXEC, $CMEXEC_64, $CMKRNL, $SETPRV

Condition Values Returned

SS$_ACCVIO

The *arglst* argument or routine is not accessible.

SS$_BADPARAM

The routine specified is in a translated image.

SS$_NOCMKRNL

The process does not have the privilege to change mode to kernel.

All other values

The routine executed returns all other values.

$CONNECT

Connect RAB with FAB — The Connect service establishes a record stream by associating and connecting a RAB with a FAB. You can invoke the Connect service only for files that are already open. For additional information about this service, see the *VSI OpenVMS Record Management Services Reference Manual*.

$CPU_CAPABILITIES

Modify CPU User Capabilities — On Alpha and Integrity server systems, allows modification of the user capability set for a specified CPU, or for the global user capability CPU default. This service accepts 64-bit addresses.

Format

```
SYS$CPU_CAPABILITIES
   cpu_id [,select_mask] [,modify_mask]  [,prev_mask] [,flags]
```

C Prototype

```
int sys$cpu_capabilities
   (int cpu_id, struct _generic_64 *select_mask,
    struct _generic_64 *modify_mask, struct _generic_64 *prev_mask,
    struct _generic_64 *flags);
```

Arguments

cpu_id

OpenVMS usage:	longword
type:	longword (unsigned)

access:	read only
mechanism:	by value

Identifier of the CPU whose user capability mask is to be modified or returned. The *cpu_id* argument is a longword containing this number, which is in the supported range of individual CPUs from 0 to SYI$_MAX_CPUS -1.

Specifying the constant CAP$K_ALL_ACTIVE_CPUS applies the current modification operation to all CPUs currently in the active set, and to the default CPU initialization context in SCH$GL_DEFAULT_CPU_CAP. If the *prev_mask* argument is also supplied, the previous default CPU initialization context in SCH$GL_DEFAULT_CPU_CAP will be returned rather than any specific CPU state.

To modify only the user capabilities in SCH$GL_DEFAULT_CPU_CAP, the *flags* argument has a bit constant CAP$M_FLAG_DEFAULT_ONLY. When this bit is set, all service operations are performed on the global cell rather than on an individual CPU specified in the *cpu_id* argument. This bit does not supersede the CAP$K_ALL_ACTIVE_CPUS constant, however. If both constants are specified, CAP$K_ALL_ACTIVE_CPUS take precedence; nevertheless, the operations to SCH$GL_DEFAULT_CPU are identical because that function is a direct subset of the other.

select_mask

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

Mask specifying which bits of the specified CPU's user capability mask are to be modified. The *select_mask* argument is the 32- 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 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 connecting the appropriate bits with a logical OR operation.

The constant CAP$K_ALL_USER, when specified in the *select_mask* argument, selects all user capability bits.

modify_mask

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

Mask specifying the settings for those capabilities selected in the *select_mask* argument. The *modify_mask* argument is the 32- 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 CPU; when clear, the corresponding user capability is to be removed from the specified CPU.

The 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 connecting the appropriate bits with OR.

To add a specific user capability to the specified CPU, that bit position must be set in both *select_mask* and *modify_mask*. To remove a specific user capability from the specified CPU, that bit position must be set in *select_mask* and clear in *modify_mask*.

The symbolic constant CAP$K_ALL_USER_ADD, when specified in *modify_mask*, indicates that all capabilities specified in *select_mask* are to be added to the current user capability set. The constant CAP$K_ALL_USER_REMOVE indicates that all capabilities specified are to be cleared from the set.

prev_mask

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

Previous user capability mask for the specified CPU before execution of this call to $CPU_CAPABILITIES. The *prev_mask* argument is the 32- or 64-bit address of a quadword into which $CPU_CAPABILITIES writes a quadword bit mask specifying the previous user capabilities.

If this argument is specified in conjunction with CAP$K_ALL_ACTIVE_CPUS as the *cpu_id* selection constant or with CAP$M_FLAG_DEFAULT_ONLY, the user capability portion of the default boot initialization state context SCH$GL_DEFAULT_CPU_CAP will be returned.

flags

OpenVMS usage:	mask_quadword
type:	quadword (unsigned)
access:	read only
mechanism:	by 32- 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 in the following table are used; the remainder of the quadword bits are reserved and must be 0.

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$M_FLAG_DEFAULT_ONLY	Indicates that the specified operations are to be performed on the global context cell instead of on a specific CPU. This bit supersedes any individual CPU specified in *cpu_id* but does not override the all active set behavior (CAP$K_ALL_ACTIVE_CPUS). Specifying this bit

Symbolic Name	Description
	constant applies this operation to the default startup capabilities for all CPUs booted for the first time.
CAP$M_FLAG_CHECK_CPU	Determines whether the kernel thread can be left in a non-runnable state under some circumstances. No operation of this service allows a transition from a runnable to blocked state; however, if the kernel thread is already at a blocked state, this bit determines whether the result of the operation must leave it runnable. If CAP$M_FLAG_CHECK_CPU is set or *flags* is not specified, the kernel thread is checked to ensure it can safely run on one of the CPUs in the active set. If CAP$M_FLAG_CHECK_CPU is not set, any state operations on kernel threads already in a blocked state are allowed.

Description

The Modify CPU User Capabilities system service, based on the arguments *select_mask* and *modify_mask*, adds or removes user capabilities for the specified *cpu_id*. If specified, the previous capability mask is returned in *prev_mask*. With the *modify_mask* argument, multiple user capabilities for a CPU 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, *select_mask* must be specified as an argument. If *modify_mask* is not specified, no modifications are made to the user capability mask for the specified CPU. In this case, *select_mask* is ignored. If *prev_mask* is not specified, no previous mask is returned.

No service state changes that will place any currently runnable kernel thread into a blocked state are allowed.

If CAP$K_ALL_ACTIVE_CPUS is specified in *cpu_id*, the user capability modifications are performed on all CPUs currently in the active set, as well as the global initialization cell. If the bit constant CAP$M_FLAG_DEFAULT_ONLY is set in the *flags* argument, the user capability modifications are made only to the global initialization cell, regardless of what individual CPU is specified in *cpu_id*.

Required Access or Privileges

The caller must have both ALTPRI and WORLD privileges to call SYS$CPU_CAPABILITIES to modify CPU user capabilities.

No privileges are required if SYS$CPU_CAPABILITIES is called only to retrieve the current user capabilities mask from the specified CPU or global default.

Related Services

$PROCESS_CAPABILITIES

Condition Values Returned

SS$_NORMAL

The service completed successfully.

SS$_BADPARAM

One of more arguments has an invalid value or the specified CPU is not in the configuration.

SS$_ACCVIO

The service cannot access the locations specified by one or more arguments.

SS$_NOPRIV

Insufficient privilege for attempted operation.

SS$_CPUCAP

Attempted operation would place one or more processes in an unrunnable state.

SS$_INSFARG

Fewer than the required number of arguments were specified or no operation was specified.

$CPU_TRANSITION (Alpha and Integrity servers)

— On Alpha and Integrity server systems, changes the current processing state of a CPU in the configure set of the current system or an unassigned CPU in an OpenVMS Galaxy configuration. This service completes asynchronously. For synchronous completion, use the $CPU_TRANSITIONW service.
This service accepts 64-bit addresses. Parameter and bit definitions are resolved in $CSTDEF in the appropriate STARLET library.
For more information, see the *VSI OpenVMS Alpha Partitioning and Galaxy Guide*.

Format

```
SYS$CPU_TRANSITION
   tran_code ,cpu_id ,nodename ,node_id ,flags ,efn ,iosb ,astadr_64
   ,astprm_64
```

C Prototype

```
int sys$cpu_transition
   (int tran_code, int cpu_id, dsc64$descriptor_s_pq nodename, int node_id,
    uint32 flags, int efn, IOSB *iosb, VOID_PQ astadr, uint64 astprm,
    uint32 timout);
```

Arguments

tran_code

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

Identifier specifying the type of state change to be initiated on the target CPU. The *tran_code* argument is a longword containing one of the following values:

Symbolic Name	Description
CST$K_CPU_STOP	The target CPU is to be removed from the active set and halted into console mode. It remains in the configure set of the current partition.
CST$K_CPU_MIGRATE	The target CPU is removed from the configure set of the local partition and the console is requested to add it to the configure set of the partition specified in node_id. If the CPU is currently in the active set, it is automatically brought to console mode through the CST$K_CPU_STOP function first.
CST$K_CPU_START	The target CPU is requested to exit console mode and join the active set of the current partition. The CPU must already be part of the configure set.
CST$K_CPU_FAILOVER	The CPU is assigned a default target partition where it will automatically migrate on system failure. This assignment persists until it is superseded. To remove an assignment or partition name, the current partition ID should be specified.
CST$K_CPU_POWER_OFF	The requested operation is initiated on the target CPU to bring the electrical power to the OFF state. If the CPU is currently in the active set, it is automatically brought to console mode through the CST$K_CPU_STOP function first.
CST$K_CPU_POWER_ON	The requested operation is initiated on the target CPU to bring the electrical power to the ON state.

Each $K code represents an end state operation, each of which has a specific start state that the CPU must be in, in order to initiate the transition.

This service may automatically initiate a successful completion of the requested operation by initiating one or more transparent transitions. This operation takes place if the CPU is not in that specific start state, and there are an obvious and unique set of transitions that can be initiated prior to the specified end state.

Multiple transitions can also be initiated simultaneously through the system service *tran_code* parameter. Each transition code has a $M form as shown in the following list, that can be or'd with a specific end state $K code:

- CST$M_CPU_STOP
- CST$M_CPU_MIGRATE
- CST$M_CPU_START
- CST$M_CPU_FAILOVER
- CST$M_CPU_POWER_OFF
- CST$M_CPU_POWER_ON

Any legal combination of transitions can be specified with the $M form, however no more than one $K code is allowed.

cpu_id

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

Identifier of the CPU whose state is to be modified. The *cpu_id* argument is a longword number in the supported range of individual CPUs from 0 to SYI$_MAX_CPUS - 1.

Generic identifiers can also be used to allow OpenVMS to select the most appropriate resource. The following table lists these codes:

Code	Description
CST$K_ANY_OWNED_CPU	Any CPU in the configure set, regardless of the active set state
CST$K_ANY_ACTIVE_CPU	Any CPU in the active set
CST$K_ANY_STOPPED_CPU	Any CPU in the configure set, but not the active set

node_id

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

Identifier of the target Galaxy partition in CSTK_CPU_ASSIGN, CSTK_CPU_FAILOVER, or CST$K_CPU_MIGRATE transition. The *node_id* argument is a longword containing a number in the supported range of IDs provided by the console for the current hardware platform. If the *nodename* parameter is specified, *node_id* is ignored.

flags

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

Options selected for the CPU state transition. The *flags* argument is a longword bit vector wherein a bit corresponds to an option. Only the bits specified below are used; the remainder of the longword bits are reserved and must be 0.

Each option (bit) has a symbolic name. The *flags* argument is constructed by performing a logical OR operation using the symbolic names of the following options:

Symbolic Name	Description
CST$V_CPU_DEFAULT_ CAPABILITIES	At the completion of the transition, the CPU's user capabilities are set back to the default system value. If

Symbolic Name	Description
	this option is not specified, modified user capabilities are maintained across STOP and START transitions as long as the CPU remains in the local partition configure set.
CST$V_CPU_ALLOW_ORPHANS	The transition is to be allowed even though it will leave at least one thread in the system unable to execute on any CPU in the active set.

efn

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

The event flag to be set when the state transition attempt has completed. The *efn* argument is a longword specifying the number of the event flag; however, this service only uses the low-order byte.

When you invoke $CPU_TRANSITION, the specified event flag is cleared; when the operation is complete, the event flag is set.

iosb

OpenVMS usage:	io_status_area
type:	IOSB 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 transition operation. The *iosb* 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; its definition can be found in $IOSBDEF in STARLET.MLB for macro and in the file IOSBDEF.H in SYS$STARLET_C.TLB for C.

When you call $CPU_TRANSITION, the I/O status area is cleared. After the transition operation is complete, the block is modified as follows:

Symbolic Name	Description
iosb$w_status	The first word contains the condition value return, indicating the final completion status of the operation.
	The first bit in the second word of the IOSB is set only if an error occurred during the operation; the remaining bits are zeroes.

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 requested transition attempt has completed. The *astadr_64* argument is the 32-bit or 64-bit virtual address of this routine. If you specify the *astadr_64* argument, the AST routine executes at the access mode from which the state transition was requested.

astprm_64

OpenVMS usage:	**user_arg**
type:	**quadword**
access:	**read only**
mechanism:	**by value**

The quadword AST parameter to be passed to the AST routine.

Description

The state transition in *tran_id* is requested for the target *cpu_id*.

A CPU currently in the active set can transition:

- To the STOPPED state; removed from the active set and left in the configure set, halted in console mode.

A CPU in the configure set can transition:

- To the UNASSIGNED state by STOPPING it and then DEASSIGNING it back to the console.
- To the ACTIVE state; warm rebooted and a full member of the symmetric multiprocessing (SMP) active set on the requesting partition.
- To another partition through MIGRATION; the target CPU is removed from the configure set and added to the configure set of the partition specified by *node_id*.

A CPU in the Galaxy unassigned state (not in the configure set of any partition in the platform) can transition:

- To the ASSIGNED state; in the configure set of the partition specified by *node_id*. Any partition can make the assignment, but the CPU must be unassigned.

CPU state transition is a two-phase operation in the OpenVMS scheduling model. This service initiates the request in the process context of the caller and returns the setup status in the service condition value. Phase 2 proceeds asynchronously; the *efn* and *iosb* arguments can be used to indicate the completion of the transition through the standard wait system services. Additional notification of the completion can be made through the OpenVMS AST mechanisms using a routine specified in *astadr_64* and a user-supplied parameter in *astprm_64*.

Required Privileges

The caller must have the CMKRNL privilege to call SYS$CPU_TRANSITION to modify CPU states.

Related Services

$CPU_TRANSITIONW

Condition Values Returned

SS$_NORMAL

The service completed successfully.

SS$_BADPARAM

One of more arguments has an invalid value or the specified CPU is not in the configuration.

SS$_ACCVIO

The service cannot access the locations specified by one or more arguments.

SS$_NOPRIV

The service cannot access the locations specified by one or more arguments.

SS$_NOPRIV

Caller does not have CMKRNL privilege needed to complete operation.

SS$_INSFARG

Fewer than the required number of arguments were specified or no operation was specified.

SS$_TOO_MANY_ARGS

More arguments were specified than are allowed by the service.

SS$_INVCOMPID

The target partition ID is not valid in this configuration.

SS$_CPUNOTACT

The specified CPU is not part of the current partition's active set.

SS$_NOSUCHCPU

The specified CPU does not exist in the current hardware configuration, or is not in the configure set of the local partition.

SS$_CPUSTARTD

The specified CPU is already in the local active set, or is in the process of joining it.

SS$_CPUSTOPPING

The specified CPU is already STOPPED or in the processing of leaving the active set.

$CPU_TRANSITIONW (Alpha and Integrity servers)

CPU Transition and Wait — On Alpha and Integrity server systems, changes the current processing state of a CPU in the configure set or an unassigned CPU in a Galaxy configuration. This service completes synchronously; that is, it returns to the caller only after the final completion status of the operation is known. In all other respects, $CPU_TRANSITIONW is identical to $CPU_TRANSITION. For all other information about the $CPU_TRANSITIONW service, see the description of $CPU_TRANSITION in this manual.
This service accepts 64-bit addresses.
For more information, see the *VSI OpenVMS Alpha Partitioning and Galaxy Guide*.

Format

```
SYS$CPU_TRANSITIONW
   tran_id ,cpu_id ,nodename ,node_id ,flags ,efn ,iosb ,astadr_64
   ,astprm_64
```

C Prototype

```
int sys$cpu_transitionw
   (int tran_code, int cpu_id, dsc64$descriptor_s_pq nodename, int node_id,
    uint32 flags, int efn, IOSB *iosb, UINT64_PQ astadr, uint64 astprm,
    uint32 timout);
```

$CREATE

Constructs New File — The Create service constructs a new file according to the attributes you specify in the FAB. If any XABs are chained to the FAB, then the characteristics described in the XABs are applied to the file. This service performs implicit Open and Display services. For additional information about this service, see the *VSI OpenVMS Record Management Services Reference Manual*.

$CREATE_BUFOBJ_64 (Alpha and Integrity servers)

Create Buffer Object — On Alpha and Integrity server systems, creates a buffer object out of a range of pages. This service accepts 64-bit addresses.

Format

```
SYS$CREATE_BUFOBJ_64
   start_va_64 ,length_64 ,acmode ,flags ,return_va_64 ,return_length_64,
   buffer_handle_64
```

C Prototype

```
int sys$create_bufobj_64
   (void *start_va_64, unsigned __int64 length_64, unsigned int acmode,
```

```
unsigned int flags, void *(*(return_va_64)),
unsigned __int64 *return_length_64,
struct _generic_64 *buffer_handle_64);
```

Arguments

start_va_64

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

Starting virtual address of the pages to be included in the buffer object. The specified virtual address will be rounded down to a CPU-specific page boundary.

The virtual address space must already exist.

length_64

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

Length of the virtual address space to be included in the buffer object. The specified length will be rounded up to a CPU-specific page boundary such 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. For the $CREATE_BUFOBJ_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 in the specified input range.

flags

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

Flag mask specifying the request options. The *flags* argument is a longword bit vector in which each bit corresponds to a flag. The $CBODEF macro in STARLET.MLB and CBODEF.H file in SYS $STARLET_C.TLB define a symbolic name for each flag. The following table describes each flag that is valid for the $CREATE_BUFOBJ_64 service:

Flag	Value	Description
CBO$M_RETSVA	1	If set, returns the system virtual address in the *return_va_64* argument instead of the process virtual address range. (Valid for inner mode callers only.)
CBO$M_SVA_32	4	If set, creates the buffer object window in 32-bit S0/S1 space. (By default, this service creates the window in 64-bit S2 space).

return_va_64

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

The lowest process virtual address of the pages in the buffer object. The *return_va_64* argument is the 32- 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- or 64-bit reference

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

buffer_handle_64

OpenVMS usage:	handle

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

The 32- or 64-bit virtual address of a naturally aligned quadword into which a buffer handle is returned to be used when referencing the created buffer object.

Description

The Create Buffer Object service creates a buffer object for use by the I/O subsystem. The pages that constitute the buffer object are permanently locked into physical memory (but not the process's working set) and double mapped into system space. The net effect is:

- I/O can be initiated to or from the buffer without the need to probe or lock the buffer pages.
- The process is still fully swappable.

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 *buffer_handle_64* arguments.

If a condition value other than SS$_ACCVIO is returned, the returned address and returned length indicate the pages that were successfully made part of the buffer object before the error occurred. If no pages were made part of the buffer object, the *return_va_64* argument will contain the value -1, and a value is *not* returned in the memory location pointed to by the *return_length_64* argument.

Required Privileges

No privileges are required if calling $CREATE_BUFOBJ_64 from an inner mode. If calling from user mode, the process must hold the rights identifier VMS$BUFFER_OBJECT_USER at the time of the call. This identifier is normally granted by the system administrator via the Authorize utility.

Required Quota

No process quota is charged, but the number of pages is limited by the system parameter MAXBOBMEM.

Related Services

$CRETVA_64, $DELETE_BUFOBJ, $EXPREG_64

Condition Values Returned

SS$_NORMAL

The service completed successfully.

SS$_ACCVIO

The *return_va_64*, *return_length_64*, or *buffer_handle_64* argument cannot be written by the caller.

SS$_BADPARAM

Invalid flags options specified.

SS$_EXBUFOBJLM

Buffer object cannot be created because it would bring the total number of buffer object pages above the systemwide limit MAXBOBMEM.

SS$_INSFMEM

Insufficient dynamic memory.

SS$_INSFSPTS

Insufficient system page table entries.

SS$_NOBUFOBJID

The process attempted to create a buffer object from user mode but was not holding required rights identifier VMS$BUFFER_OBJECT_USER.

SS$_NOPRIV

Valid flag options were specified but from user mode.

SS$_PAGNOTWRITE

A page within the address range is not writeable.

SS$_PAGOWNVIO

The pages could not put into the buffer object because the access mode associated with the call to $CREATE_BUFOBJ_64 was less privileged than the access mode associated with the pages.

$CREATE_GALAXY_LOCK (Alpha Only)

Create OpenVMS Galaxy Lock — Allocates an OpenVMS Galaxy lock block from a lock table created with the $CREATE_GALAXY_LOCK_TABLE service. Note that this system service is supported only in an OpenVMS Alpha Galaxy environment. For more information about programming with OpenVMS Galaxy system services, see the *VSI OpenVMS Alpha Partitioning and Galaxy Guide*.

Format

```
YS$CREATE_GALAXY_LOCK
   lcktbl_handle ,name ,size ,timeout ,ipl ,rank ,handle
```

C Prototype

```
int sys$create_galaxy_lock
   (unsigned int lcktbl_handle, void *name, unsigned int size,
    unsigned int timeout, unsigned int ipl, unsigned int rank,
    unsigned __int64 *lock_handle);
```

Arguments

lcktbl_handle

OpenVMS usage:	lock table handle
type:	longword (unsigned)
access:	read
mechanism:	input by value

The 32-bit lock table handle that identifies the lock table in which to create the lock. This value is returned by SYS$CREATE_GALAXY_LOCK_TABLE.

name

OpenVMS usage:	address
type:	ASCID string
access:	read
mechanism:	input by reference

The *name* parameter is a pointer to an ASCID string (passed by descriptor). The name can be a maximum of 15 characters. Lock names are not checked for uniqueness; therefore, multiple locks can be created with the same name.

timeout

OpenVMS usage:	wait timeout
type:	longword (unsigned)
access:	read
mechanism:	input by value

The 32-bit wait or spin timeout specified in 10 microsecond units. If not specified, the timeout defaults to 10 microseconds.

size

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

The size of the galaxy lock in bytes. Galaxy locks have two legal sizes. These values are returned by SYS$GET_GALAXY_LOCK_SIZE. The value passed to SYS$CREATE_GALAXY_LOCK must be equal to the value passed to the call to SYS$CREATE_GALAXY_LOCK_TABLE.

ipl

OpenVMS usage:	IPL of lock

type:	longword (unsigned)
access:	read
mechanism:	input by value

For galaxy locks acquired in kernel mode, the IPL to raise to while the lock is held. This parameter is ignored for all other access mode.

rank

OpenVMS usage:	rank of lock
type:	longword (unsigned)
access:	read
mechanism:	input by value

Rank applied to a galaxy lock. Ranking is used to detect potential deadlocks. This parameter is currently ignored.

handle

OpenVMS usage:	address
type:	quadword (unsigned)
access:	write
mechanism:	output by reference

The handle parameter is a pointer to a quadword. The value returned is a 64-bit handle that uniquely identifies the lock galaxy-wide.

Description

This service allocates an OpenVMS Galaxy lock block from a lock table created with the $CREATE_GALAXY_LOCK_TABLE service.

Required Access or Privileges

For System Lock: CMKRNL, SHMEM

For User Lock: SHMEM

Write access to Lock Table

Required Quota

None

Related Services

$ACQUIRE_GALAXY_LOCK, $CREATE_GALAXY_LOCK_TABLE, $DELETE_GALAXY_LOCK, $DELETE_GALAXY_LOCK_TABLE, $GET_GALAXY_LOCK_INFO, $GET_GALAXY_LOCK_SIZE, $RELEASE_GALAXY_LOCK

Condition Values Returned

SS$_NORMAL

Normal completion.

SS$_ACCVIO

Access violation on parameter.

SS$_BADLCKTBL

OpenVMS Galaxy lock table is corrupt.

SS$_BADPARAM

Bad parameter value.

SS$_IVLOCKID

Invalid lock id.

SS$_IVLOCKTBL

Invalid lock table.

SS$_INSFMEM

Insufficient memory in lock table.

SS$_NOCMKRNL

Operation requires CMKRNL privilege.

SS$_NOSHMEM

Operation requires SHMEM privilege.

$CREATE_GALAXY_LOCK_TABLE (Alpha Only)

Create OpenVMS Galaxy Lock Table — Allocates an OpenVMS Galaxy lock table. Note that this system service is supported only in an OpenVMS Alpha Galaxy environment. For more information about programming with *VSI OpenVMS Alpha Partitioning and Galaxy Guide*.

Format

```
SYS$CREATE_GALAXY_LOCK_TABLE
   name ,accmode ,section_size ,section_type ,prot ,lock_size
   ,lcktbl_handle
```

C Prototype

```
int sys$create_galaxy_lock_table
```

```
(void *name, unsigned int accmode, unsigned __int64 section_size,
 unsigned int section_type, unsigned int prot, unsigned int lock_size,
 unsigned int *lcktbl_handle);
```

Arguments

name

OpenVMS usage:	address
type:	ASCID string
access:	read
mechanism:	input by reference

The name parameter is a pointer to an ASCID string (passed by descriptor). The name is given to the global section that is created to contain the galaxy locks.

accmode

OpenVMS usage:	access mode
type:	longword (unsigned)
access:	read
mechanism:	input by value

Access mode that is to be the owner of the pages created during the mapping. The *accmode* argument is a longword containing the access mode.

section_size

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

Length of the global section to be created, in bytes. The size must be specified as a multiple of the CPU-specific page size. A size of zero is illegal.

section_type

OpenVMS usage:	bit mask
type:	longword (unsigned)
access:	read
mechanism:	input by value

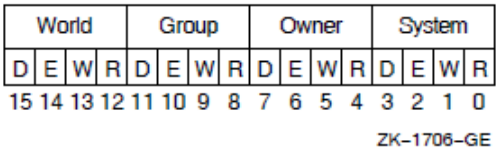
Used to control where in virtual memory the global section is created. If GLCKTBL$C_PROCESS is specified, the section is created in P2 (process) space. If GLCKTBL$C_SYSTEM is specified, the section is created in S0/S1 (system) space. These constants are defined in the GLOCKDEF macro.

prot

OpenVMS usage:	protection
type:	longword (unsigned)
access:	read
mechanism:	input by value

Protection to be applied to the global section.

The mask contains four 4-bit fields. Bits are read from right to left in each field. The following diagram depicts the mask:



Cleared bits indicate that read, write, execute, and delete access, in that order, are granted to the particular category of user.

Only read and write access are meaningful for lock section protection. Delete access bits are ignored.

lock_size

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

The size of the galaxy lock in bytes. Galaxy locks have two sizes. The legal values are returned by SYS$GET_GALAXY_LOCK_SIZE.

lcktbl_handle

OpenVMS usage:	address
type:	longword
access:	write
mechanism:	output by reference

Pointer to a longword. The value returned is a 32-bit handle that uniquely identifies the lock table galaxy-wide.

Description

This service allocates an OpenVMS Galaxy lock table. This structure is used to maintain information about a shared memory section, which this service also creates. The first caller of the service with a unique lock table name creates the section. Additional callers map it. This shared memory section contains a set of Galaxy locks. All locks residing in the section are of the same size. Once the lock table

is created, the $CREATE_GALAXY_LOCK service can be used to create and allocate a lock from the table.

The flags GLCKTBL$C_PROCESS and GLCKTBL$C_SYSTEM specify whether the shared memory region is mapped into system space or process space. Creation of process space sections requires the SHMEM privilege. Creation of system space sections requires the SHMEM and CMKRNL privileges.

Required Access or Privileges

CMKRNL, SHMEM

Required Quota

None

Related Services

$ACQUIRE_GALAXY_LOCK, $CREATE_GALAXY_LOCK, $DELETE_GALAXY_LOCK, $DELETE_GALAXY_LOCK_TABLE, $GET_GALAXY_LOCK_INFO, $GET_GALAXY_LOCK_SIZE, $RELEASE_GALAXY_LOCK

Condition Values Returned

SS$_NORMAL

Normal completion.

SS$_ACCVIO

Access violation on parameter.

SS$_BADPARAM

Bad parameter value.

SS$_CREATED

File or section did not exist; has been created.

SS$_IVLOCKID

Invalid lock id.

SS$_NOPRIV

No privilege for attempted operation.

SS$_NOCMKRNL

Operation requires CMKRNL privilege.

SS$_NOSHMEM

Operation requires SHMEM privilege.

$CREATE_GDZRO

Create Permanent Global Demand-Zero Section — On Alpha and Integrity server systems, creates a permanent, memory-resident, global demand-zero section to which processes can map. Shared page table sections can also be created. This service accepts 64-bit addresses.

Format

```
SYS$CREATE_GDZRO
   gs_name_64 ,ident_64 ,prot ,length_64 ,acmode ,flags
   [,reserved_length_64] [,rad_mask]
```

C Prototype

```
int sys$create_gdzro
   (void *gs_name_64, struct _secid *ident_64, unsigned int prot,
    unsigned __int64 length_64, unsigned int acmode,
    unsigned int flags,...);
```

Arguments

gs_name_64

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

Name of the global section. The *gs_name_64* argument is the 32- or 64-bit virtual address of a naturally aligned 32- 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- or 64-bit reference

Identification value specifying the version number of the global section. The *ident_64* argument is a quadword containing three fields. The *ident_64* argument is the 32- 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 2 bits. 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.

Value	Symbolic Name	Match Criteria
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.

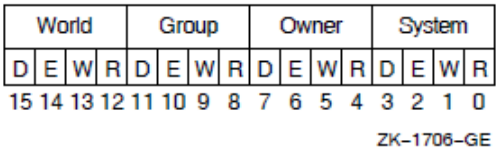
If you specify the *ident_64* argument as 0, the version number and match control fields default to 0.

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.

prot

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

Protection to be applied to the global demand-zero section. The mask contains four 4-bit fields. Bits are read from right to left in each field. The following diagram depicts the mask:



Cleared bits indicate that read, write, execute, and delete access, in that order, are granted to the particular category of user. Only read, write, and execute access are meaningful for section protection. Delete access bits are ignored. Read access also grants execute access for those situations where execute access applies. If 0 is specified, read access and write access are granted to all users.

length_64

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

Length, in bytes, of the global demand-zero section to be created. The *length_64* argument must be specified as a multiple of the CPU-specific page size. A length of 0 cannot be specified.

Note

Creating a memory-resident global section with shared page tables does not imply that the global section must have a length that is an even multiple of CPU-specific page table pages. The global section might not fully use the last shared page table page.

acmode

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

Access mode that is to be the owner of the pages created during the mapping. The *acmode* argument is a longword containing the access mode.

If the memory-resident global section is created with shared page tables, this is the access mode that is stored in the owner, read, and write fields of the corresponding shared page table entries (PTEs).

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

flags

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

Flag mask specifying the type of global section to be created as well as its characteristics. 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 the flags that are valid for the $CREATE_GDZRO service:

Flag	Description
SEC$M_DZRO	Pages are demand-zero pages. By default, this flag is always present in this service and cannot be disabled.
SEC$M_GBL	Pages form a global section. By default, this flag is always present in this service and cannot be disabled.
SEC$M_MRES	Pages form a memory-resident section. By default, this flag is always present in this service and cannot be disabled.
SEC$M_PERM	Global section is permanent. By default, this flag is always present in this service and cannot be disabled.
SEC$M_RAD_HINT	When set, the argument *rad_mask* is used as a mask of RADs from which to allocate memory. See the *rad_mask* argument description for more information.

Flag	Description
SEC$M_READ_ONLY_SHPT	Create shared table pages for the section that allow read access only.
SEC$M_SHMGS	Create a shared-memory global section.
SEC$M_SYSGBL	Pages form a system global section. By default, pages form a group global section.
SEC$M_WRT	Pages form a read/write section. By default, this flag is always present in this service and cannot be disabled.

All other bits in the **flags** argument are reserved to OpenVMS for future use and should be specified as 0. The condition value SS$_IVSECFLG is returned if any undefined bits are set.

reserved_length_64

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

Length, in bytes, of the global section as currently registered in the Reserved Memory Registry. The *reserved_length_64* argument is the 32- or 64-bit virtual address of a naturally aligned quadword into which the service returns the reserved length.

If *reserved_length_64* is not specified or is specified as 0, no reserved length is returned to the caller.

If the memory-resident global section is not registered, *reserved_length_64* is written with the value 0.

rad_mask

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

Use the *rad_mask* argument to specify from which RADs to allocate memory. Currently only one bit may be set. The specified RAD must contain memory. This argument is only a hint. Memory may be obtained from other RADs if no free memory is available at the time of allocation.

The *rad_mask* argument is considered only if the SEC$M_RAD_HINT flag is specified. Otherwise, this argument is ignored.

On a system that does not support resource affinity domains (RADs), specifying 1 for the *rad_mask* argument is allowed.

RAD is supported on AlphaServer GS series systems and starting from OpenVMS Version 8.4, support is extended to NUMA capable Integrity servers.

Description

The Create Permanent Global Demand-Zero Section service allows a process to create a permanent, memory-resident, global demand-zero section. If you set the SEC$M_SHMGS flag, the section is created as a Galaxy-wide global demand-zero section in shared memory.

You must call either the $CREATE_GDZRO service or the $CRMPSC_GDZRO_64 service on each instance where the Galaxy shared memory will be accessed.

Memory-resident or Galaxy-wide global sections contain demand-zero allocation pages that are writable and memory resident. All pages in these types of global section are shared by all processes that map to the global section.

The pages are always resident in memory and are not backed up by any file on any disk. The pages are not placed into the process's working set list when the process maps to the global section and the virtual memory is referenced by the process. The pages are also not charged against the process's working set quota or against any page-file quota.

To create a memory-resident section, the process must have the rights identifier, VMS$MEM_RESIDENT_USER. The error status, SS$_NOMEMRESID, is returned if the caller has not been granted this identifier. To create a Galaxy-wide shared section, the process must have the SHMEM privilege.

Only memory-resident sections can be registered with the Reserved Memory Registry in the SYSMAN facility. Memory for Galaxy-wide shared sections is reserved through appropriate settings of the console environment parameters.

If the section is not registered in the Reserved Memory Registry, or if the /NOALLOCATE qualifier was specified when the global section was registered in the Reserved Memory Registry, invalid global PTEs are written to the global page table. When the global section is mapped, invalid page table entries are placed in the process page table. Physical memory is not allocated until the pages are referenced.

If the global section is registered in the Reserved Memory Registry, the size of the global section need not match the reserved size. If the global section is not registered in the Reserved Memory Registry, or if the reserved size is smaller than the size of the global section, the error status SS$_INSFLPGS is returned if there are not enough fluid pages in the system to satisfy the request.

If the /ALLOCATE qualifier was specified when the global section was registered in the Reserved Memory Registry, contiguous, aligned, physical pages are preallocated during system initialization for this global section. Valid page table entries are placed in the global page table and when the global section is mapped, valid page table entries are placed in the process page table. With the proper virtual alignment, granularity hints (GH) are used to map to the global pages.

If the global section is not registered in the Reserved Memory Registry, or if the /PAGE_TABLES qualifier was specified when the global section was registered, shared page tables are created for the memory-resident global section.

If the /ALLOCATE and /PAGE_TABLE qualifiers were specified when the global section was registered in the Reserved Memory Registry, contiguous, aligned physical pages are preallocated during system initialization for this global section, and granularity hints are used to map to the shared page table sections.

The following table lists the factors affecting the creation of shared page tables for memory-resident global sections. The /ALLOCATE and the /PAGE_TABLES qualifiers pertain to the Reserved Memory

Registry command RESERVED_MEMORY ADD entered for the memory-resident global section being created. For more information about using the SYSMAN utility to create entries to the Reserved Memory Registry, see the *VSI OpenVMS System Management Utilities Reference Manual*.

/ALLOCATE	/PAGE_TABLES	Outcome
Not registered	Not registered	Shared page tables created. Shared page tables cannot use GH. Returns SS$_CREATED_SHPT.
No	No	No shared page tables created. Returns SS$_CREATED.
No	Yes	Shared page tables created. Shared page tables cannot use GH. Returns SS$CREATED_SHPT.
Yes	No	No shared page tables created. Returns SS$_CREATED.
Yes	Yes	Shared page tables created. Shared page tables can use GH. Returns SS$_CREATED_SHPT.

Shared page tables are always created for Galaxy-wide shared sections of at least 128 pages.

Shared page tables consume the same internal OpenVMS data structures as global sections. The system parameters GBLPAGES and GBLSECTIONS must account for the additional global pages and the additional global section.

Note that only one set of shared page tables can be associated with any memory-resident or Galaxy-wide section. By default, shared page tables will allow write access. To create shared page tables that allow only read access, you must set the READ_ONLY_SHPT flag. A process that requires write access to a section where the shared page tables only allow read access must use private page tables to map the section.

To use the shared page tables associated with a memory-resident global section, a process must first create a shared page table region (with $CREATE_REGION_64). Additionally, a subsequent request to map to the memory-resident global section must do the following:

- Specify a shared page table region to the mapping request (see Table 22 for additional information).
- Specify the same access mode as specified by the *acmode* argument to this service.
- Set the flag SEC$M_WRT in the mapping request only if shared page tables allow write access.
- Set the flag SEC$M_EXPREG in the mapping request or provide a CPU-specific page table page aligned virtual address. (See the description of the CREATE_REGION_64 service for information about calculating virtual addresses that are aligned to a CPU-specific page table page boundary.)

If a shared page table region is not specified, process-private page tables are used to map to the global section.

If the service returns an error status value that is neither SS$_INSFLPGS nor SS$_MRES_PFNSMALL, a value is not returned in the *reserved_length_64* argument.

If the service returns a successful condition value or if SS$_INSFLPGS or SS$_MRES_PFNSMALL is returned and the *reserved_length_64* argument is specified as a nonzero address, the length in bytes of the global section as registered in the Reserved Memory Registry is returned in the *reserved_length_64* argument.

To map a Galaxy shared section or a memory resident section, see the $CRMPSC_GDZRO_64 service.

For additional information, see the *VSI OpenVMS Alpha Partitioning and Galaxy Guide*.

Required Privileges

To create a permanent memory-resident global DZRO section, the process must have the following privileges or rights identifiers:

- SYSGBL privilege to create a system global section (if flag SEC$M_SYSGBL is set)
- PRMGBL privilege to create a permanent global section
- VMS$MEM_RESIDENT_USER rights identifier to create a memory-resident section
- SHMEM privilege on OpenVMS Galaxy systems to create an object in Galaxy shared memory.

Required Quota

None

Related Services

$CRMPSC_GDZRO_64, $DGBLSC, $MGBLSC_64

Condition Values Returned

SS$_NORMAL

A Galaxy-wide section already existed and has been made available.

SS$_CREATED

Global section has been created.

SS$_CREATED_SHPT

Global section has been created with shared page tables.

SS$_ACCVIO

The *gs_name_64* descriptor cannot be read by the caller, or the *reserved_length_64* argument was specified as a nonzero value and cannot be written by the caller.

SS$_BADRAD

The specified RAD contains no memory, or if the specified RAD is greater than or equal to the maximum number of RADs on the system.

SS$_DUPLNAM

A global section of the same name already exists; a new global section was not created.

SS$_GPTFULL

There is no more room in the system global page table to set up page table entries for the global section or for the shared page tables.

SS$_GSDFULL

There is no more room in the system space allocated to maintain control information for global sections.

SS$_INSFLPGS

Insufficient fluid pages available.

SS$_INSFRPGS

Insufficient free shared pages or private pages.

SS$_INV_SHMEM

Shared memory is not valid.

SS$_IVLOGNAM

The specified global section name has a length of 0 or has more than 43 characters.

SS$_IVPROTECT

The protection argument format is invalid.

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_NOTPAGMULT

The *length_64* argument is not a multiple of CPU-specific pages.

SS$_LOCK_TIMEOUT

An OpenVMS Galaxy lock timed out.

SS$_MRES_PFNSMALL

Preallocated, contiguous, aligned physical memory specified in the Reserved Memory Registry is smaller than the length specified for the global section by the *length_64* argument.

SS$_NOBREAK

An OpenVMS Galaxy lock is held by another node and was not broken.

SS$_NOMEMRESID

The process attempted to create a memory-resident section but was not holding the correct identifier (VMS$MEM_RESIDENT_USER).

SS$_NOPRMGBL

The process does not have the privileges to create or delete a permanent group global section (PRMGBL).

SS$_NOSYSGBL

The process does not have the privileges to create or delete a system global section (SYSGBL).

SS$_SECTBLFUL

There are no entries available in the system global section table for the global section or for the shared page tables.

SS$_TOOMANYLNAM

The logical name translation of the *gs_name_64* argument exceeded the allowed depth of 10.

$CREATE_GFILE (Alpha and Integrity servers)

Create Permanent Global Disk File Section — On Alpha and Integrity server systems, creates a permanent global disk file section to which processes can map. This service accepts 64-bit addresses.

Format

```
SYS$CREATE_GFILE
   gs_name_64 ,ident_64 ,file_offset_64 ,length_64 ,chan
   ,acmode ,flags ,return_length_64 [,fault_cluster]
```

C Prototype

```
int sys$create_gfile
   (void *gs_nam_64, struct _secid *ident_64, unsigned __int64
    file_offset_64, unsigned __int64 length_64, unsigned short int chan,
    unsigned int acmode, unsigned int flags,
    unsigned __int64 *return_length_64,...);
```

Arguments

gs_name_64

OpenVMS usage: section_name

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

Name of the global section. The *gs_name_64* argument is the 64-bit virtual address of a naturally aligned 32- 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- 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- 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 and 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.

file_offset_64

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

Byte offset into the file that marks the beginning of the section. The *file_offset_64* argument is a quadword containing this number. If you do not specify the *file_offset_64* argument or specify it as 0, the section is created beginning with the first byte in the file.

The *file_offset_64* argument must be a multiple of virtual disk blocks.

length_64

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

Length, in bytes, of the global disk file section to be created. The length specified must be 0 or a multiple of virtual disk blocks. If the length specified is 0 or extends beyond end-of-file (EOF), the global disk file section is created up to and including the virtual block number that contains EOF.

chan

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

Number of the channel on which the file has been accessed. The *chan* argument is a longword containing this number. The access mode at which the channel was opened must be equal to or less privileged than the access mode of the caller.

You can use the OpenVMS Record Management Services (RMS) macro $OPEN to access a file; the file options parameter in the file access block must indicate a user file open (UFO keyword).

acmode

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

Access mode that is to be the owner of the pages created during the mapping. The *acmode* argument is a longword containing the access mode.

The $PSLDEF macro in STARLET.MLB and the file PSLDEF.H in SYS$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

flags

OpenVMS usage:	mask_longword

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

Flag mask specifying the type of global section to be created as well as its characteristics. 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 $CREATE_GFILE service:

Flag	Description
SEC$M_CRF	Pages are copy-on-reference. By default, pages are shared.
SEC$M_DZRO	Pages are demand-zero pages. By default, they are not zeroed when copied. Note that SEC$M_DZRO and SEC$M_CRF cannot both be set and that SEC$M_DZRO set and SEC$M_WRT clear is an invalid combination.
SEC$M_GBL	Pages form a global section. By default, this flag is always present in this service and cannot be disabled.
SEC$M_PERM	Global section is permanent. By default, this flag is always present in this service and cannot be disabled.
SEC$M_SYSGBL	Pages form a system global section. By default, pages form a group global section.
SEC$M_WRT	Pages form a read/write section. By default, pages form a read-only section.

All other bits in the *flags* argument are reserved for future OpenVMS use 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_length_64

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

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

fault_cluster

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

Page fault cluster in byte units indicating how many pages are to be brought into memory when a page fault occurs for a single page. The fault cluster specified is rounded up to a multiple of CPU-specific pages.

If this argument is specified as 0, the system default page fault cluster is used. If this argument is specified as more than the maximum allowed for the system, no error is returned. The systemwide maximum is used.

Description

The Create Permanent Global Disk File Section service allows a process to create a permanent global disk file section.

Creating a global disk file section involves defining all or part of a disk file as a section. The global section is created as entire pages; however, the last page in the section might correspond to less than a full page of virtual disk blocks. Only the number of virtual disk blocks specified by the *length_64* argument, or as many as exist in the disk file, will be associated with the disk file section.

Upon successful completion of this service, the *return_length_64* argument will contain the length of the global section created in even multiples of virtual disk blocks.

The security profile of the file is used to determine access to the global section. For a global disk file section to allow write access to the file during the mapping of the global section, the channel used to open the file must allow write access to the file.

Required Privileges

To create a global section, the process must have the following privileges:

- SYSGBL privilege to create a system global section (if flag SEC$M_SYSGBL is set)
- PRMGBL privilege to create a permanent global section

Required Quota

None

Related Services

$CRMPSC, $CRMPSC_GFILE_64, $DGBLSC, $MGBLSC, $MGBLSC_64

Condition Values Returned

SS$_CREATED

The service completed successfully. The specified global section did not previously exist and has been created.

SS$_ACCVIO

The *gs_name_64* argument or the *return_length_64* argument cannot be read by the caller.

SS$_CHANVIO

The specified channel was assigned from a more privileged access mode.

SS$_DUPLNAM

A global section of the same name already exists; a new global section was not created.

SS$_ENDOFFILE

The *file_offset_64* argument specified is beyond the logical end-of-file.

SS$_EXBYTLM

Process has exceeded the byte count quota; the system was unable to map the requested file.

SS$_GPTFULL

There is no more room in the system global page table to set up page table entries for the section.

SS$_GSDFULL

There is no more room in the system space allocated to maintain control information for global sections.

SS$_IVCHAN

An invalid channel number was specified; the channel number specified was 0 or a channel that is unassigned.

SS$_IVCHNLSEC

The channel number specified is currently active, or there are no files opened on the specified channel.

SS$_IVIDENT

An invalid channel number was specified; the channel number specified is larger than the number of channels available.

SS$_IVLOGNAM

The specified global section name has a length of 0 or has more than 43 characters.

SS$_IVLVEC

The specified section was not installed using the /PROTECT qualifier.

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.

SS$_NOPRMGBL

The process does not have the privileges to create or delete a permanent group global section (PRMGBL).

SS$_NOSYSGBL

The process does not have the privileges to create or delete a system global section (SYSGBL).

SS$_NOTFILEDEV

The device is not a file-oriented, random-access, or directory device.

SS$_NOWRT

The file is read-only, and the flag bit SEC$M_CRF is not set.

SS$_OFF_NOTBLKALGN

The *file_offset_64* argument is not a multiple of virtual disk blocks.

SS$_SECTBLFUL

There are no entries available in the system global section table.

SS$_TOOMANYLNAM

The logical name translation of the *gs_name_64* argument exceeded the allowed depth of 10.

$CREATE_GPFILE (Alpha and Integrity servers)

Create Permanent Global Page File Section — On Alpha and Integrity server systems, creates a permanent global page file section to which processes can map. This service accepts 64-bit addresses.

Format

```
SYS$CREATE_GPFILE gs_name_64 ,ident_64 ,prot ,length_64 ,acmode ,flags
```

C Prototype

```
int sys$create_gpfile
   (void *gs_name_64, struct _secid *ident_64, unsigned int prot,
    unsigned __int64 length_64, unsigned int acmode, unsigned int flags);
```

Arguments

gs_name_64

OpenVMS usage: section_name

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

Name of the global section. The *gs_name_64* argument is the 32- or 64-bit virtual address of a naturally aligned 32- 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- 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- 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 2 bits. 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.

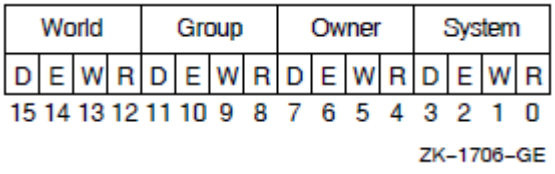
If you specify the *ident_64* argument as 0, the version number and match control fields default to 0.

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.

prot

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

Protection to be applied to the global page file section. The mask contains four 4-bit fields. Bits are read from right to left in each field. The following diagram depicts the mask:



Cleared bits indicate that read, write, execute, and delete access, in that order, are granted to the particular category of user. Only read, write, and execute access are meaningful for section protection. Delete access bits are ignored. Read access also grants execute access for those situations where execute access applies. If 0 is specified, read access and write access are granted to all users.

length_64

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

Length, in bytes, of the global page file section to be created. The *length_64* argument must be specified as a multiple of the CPU-specific page size. A length of 0 cannot be specified.

acmode

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

Access mode that is to be the owner of the pages created during the mapping. The *acmode* argument is a longword containing the access mode.

The $PSLDEF macro in STARLET.MLB and the file PSLDEF.H in SYS$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

flags

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

Flag mask specifying the type of global section to be created as well as its characteristics. 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 the flags that are valid for the $CREATE_GPFILE service:

Flag	Description
SEC$M_DZRO	Pages are demand-zero pages.
SEC$M_GBL	Pages form a global 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. By default, this flag is always present in this service and cannot be disabled.
SEC$M_PERM	Global section is permanent. By default, this flag is always present in this service and cannot be disabled.
SEC$M_SYSGBL	Pages form a system global section. By default, pages form a group global section.
SEC$M_WRT	Pages form a read/write section. By default, this flag is always present in this service and cannot be disabled.

All other bits in the *flags* argument are reserved to OpenVMS for future use and should be specified as 0. The condition value SS$_IVSECFLG is returned if any undefined bits are set.

Description

The Create Permanent Global Page File Section service allows a process to create a permanent global page file section. Global page file sections contain demand-zero allocation pages that are writable and backed up by the system page file. All pages in the global page file section are shared by all processes that map to the global section.

Required Privileges

To create a permanent global page file section, the process must have the following privileges:

- SYSGBL privilege to create a system global section (if flag SEC$M_SYSGBL is set)
- PRMGBL privilege to create a permanent global section

Required Quota

The systemwide number of global page file pages is limited by the system parameter GBLPAGFIL.

Related Services

$CRMPSC, $CRMPSC_GPFILE_64, $DGBLSC, $MGBLSC, $MGBLSC_64

Condition Values Returned

SS$_CREATED

The service completed successfully. The specified global section did not previously exist and has been created.

SS$_ACCVIO

The *gs_name_64* descriptor cannot be read by the caller.

SS$_DUPLNAM

A global section of the same name already exists; a new global section was not created.

SS$_GPTFULL

There is no more room in the system global page table to set up page table entries for the section.

SS$_GSDFULL

There is no more room in the system space allocated to maintain control information for global sections.

SS$_IVLOGNAM

The specified global section name has a length of 0 or has more than 43 characters.

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_NOTPAGMULT

The *length_64* argument is not a multiple of CPU-specific pages or was specified as 0.

SS$_NOPRMGBL

The process does not have the privileges to create or delete a permanent group global section (PRMGBL).

SS$_NOSYSGBL

The process does not have the privileges to create or delete a system global section (SYSGBL).

SS$_SECTBLFUL

There are no entries available in the system global section table.

SS$_TOOMANYLNAM

The logical name translation of the *gs_name_64* argument exceeded the allowed depth of 10.

$CREATE_GPFN (Alpha and Integrity servers)

Create Permanent Global Page Frame Section — On Alpha and Integrity server systems, creates a permanent page frame section to which processes can map. This service accepts 64-bit addresses.

Format

```
SYS$CREATE_GPFN
```

```
   gs_name_64 ,ident_64 ,prot ,start_pfn ,page_count ,acmode ,flags
```

C Prototype

```
int sys$create_gpfn
   (void *gs_name_64, struct _secid *ident_64, unsigned int prot,
    unsigned int start_pfn, unsigned int page_count, unsigned int acmode,
    unsigned int flags);
```

Arguments

gs_name_64

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

Name of the global section. The *gs_name_64* argument is the 32- or 64-bit virtual address of a naturally aligned 32- 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- 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- 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

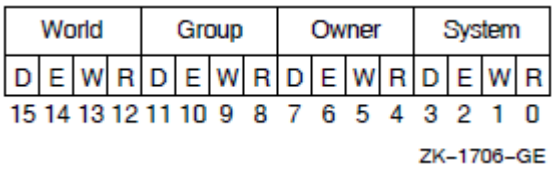
number is specified when a section is created, processes that specify a version number when mapping cannot access the global section.

prot

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

Protection to be applied to the global page frame section.

The mask contains four 4-bit fields. Bits are read from right to left in each field. The following diagram depicts the mask:



Cleared bits indicate that read, write, execute, and delete access, in that order, are granted to the particular category of user. Only read, write, and execute access are meaningful for section protection. Delete access bits are ignored. Read access also grants execute access for those situations where execute access applies. If zero is specified, read access and write access are granted to all users.

start_pfn

OpenVMS usage:	page frame number
type:	longword (unsigned) on Alpha, quadword (unsigned) on Integrity servers
access:	read only
mechanism:	by value

The CPU-specific page frame number where the section begins in memory.

page_count

OpenVMS usage:	CPU-specific page count
type:	longword (unsigned) on Alpha, quadword (unsigned) on Integrity servers
access:	read only
mechanism:	by value

Length of the page frame section in CPU-specific pages.

acmode

OpenVMS usage:	access_mode

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

Access mode that is to be the owner of the pages created during the mapping. The *acmode* argument is a longword containing the access mode.

The $PSLDEF macro in STARLET.MLB and the file PSLDEF.H in SYS$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

flags

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

Flag mask specifying the characteristics of the page frame section to be created. 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 the flags that are valid for the $CREATE_GPFN service:

Flag	Description
SEC$M_ARGS64	Indicates that all parameters, specifically *start_pfn* and *page_count*, are passed as 64-bit numbers. This flag is ignored on OpenVMS Alpha but must be set on Integrity server systems. If the flag is not set on Integrity servers, the error code SS$_IVSECFLG is returned.
SEC$M_GBL	Pages form a global section. By default, this flag is always present in this service and cannot be disabled.
SEC$M_PERM	Global section is 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_SYSGBL	Pages form a system global page frame section. By default, pages form a group global page frame section.
SEC$M_UNCACHED	Flag that must be set when a PFN-mapped section is created if this section must be treated as uncached memory. Flag is ignored on Alpha systems; it applies only to Integrity server systems.

Flag	Description
SEC$M_WRT	Pages form a read/write section. By default, pages form a read-only section.

All other bits in the *flags* argument are reserved to OpenVMS for future use 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.

Description

The Create Permanent Global Page Frame Section service allows a process to create a global page frame section. All global page frame sections are permanent. 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_64; this can result in a machine check if they are in I/O space.

Required Privileges

To create a permanent global page frame section, the process must have the following privileges:

- SYSGBL privilege to create a system global section (if flag SEC$M_SYSGBL is set)
- PRMGBL privilege to create a permanent global section
- PFNMAP privilege to create a page frame section

Required Quota

None

Related Services

$CRMPSC, $CRMPSC_GPFN_64, $DGBLSC, $MGBLSC, $MGBLSC_GPFN_64

Condition Values Returned

SS$_CREATED

The service completed successfully. The specified global section did not previously exist and has been created.

SS$_ACCVIO

The *gs_name_64* argument cannot be read by the caller.

SS$_DUPLNAM

A global section of the same name already exists; a new global section was not created.

SS$_GPTFULL

There is no more room in the system global page table to set up page table entries for the section.

SS$_GSDFULL

There is no more room in the system space allocated to maintain control information for global sections.

SS$_IVLOGNAM

The specified global section name has a length of 0 or has more than 43 characters.

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$_NOPRMGBL

The process does not have the privileges to create or delete a permanent group global section (PRMGBL).

SS$_NOSYSGBL

The process does not have the privileges to create or delete a system global section (SYSGBL).

SS$_TOOMANYLNAM

The logical name translation of the *gs_name_64* argument exceeded the allowed depth of 10.

$CREATE_RDB

Create Rights Database — Initializes a rights database.

Format

```
SYS$CREATE_RDB [sysid]
```

C Prototype

```
int sys$create_rdb (struct _generic_64 *sysid);
```

Argument

sysid

OpenVMS usage:	system_access_id
type:	quadword (unsigned)
access:	read only

mechanism: by reference

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

Description

The Create Rights Database service initializes a rights database. The database name is the file equated to the logical name RIGHTSLIST, which must be defined as a system logical name from executive mode. If the logical name does not exist, the database is created in SYS$COMMON:[SYSEXE] with the file name RIGHTSLIST.DAT. If the database already exists, $CREATE_RDB fails with the error RMS$_FEX.

The rights database is created with an owner of [1,4] and a protection of (RWED, RWED, R).

Required Access or Privileges

Write access to the directory in which the file is being created is required.

Required Quota

None

Related Services

$ADD_HOLDER, $ADD_IDENT, $ASCTOID, $FIND_HELD, $FIND_HOLDER, $FINISH_RDB, $FORMAT_ACL, $GRANTID, $GET_SECURITY, $IDTOASC, $MOD_HOLDER, $MOD_IDENT, $PARSE_ACL, $REM_HOLDER, $REM_IDENT, $REVOKID, $SET_SECURITY

Condition Values Returned

SS$_NORMAL

The service completed successfully.

SS$_ACCVIO

The *sysid* argument cannot be read by the caller.

SS$_INSFMEM

The process dynamic memory is insufficient for opening the rights database.

RMS$_FEX

A rights database already exists. To create a new one, you must explicitly delete or rename the old one.

RMS$_PRV

The user does not have write access to SYS$SYSTEM.

Because the rights database is an indexed file accessed with 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 *VSI OpenVMS Record Management Services Reference Manual*.

$CREATE_REGION_64 (Alpha and Integrity servers)

Create Virtual Region — On Alpha and Integrity server systems, creates a virtual region within the process's private address space. This service accepts 64-bit addresses.

Format

```
SYS$CREATE_REGION_64
   length_64 ,region_prot ,flags ,return_region_id_64 ,return_va_64
   ,return_length_64 [,start_va_64]
```

C Prototype

```
int sys$create_region_64
   (unsigned __int64 length_64, unsigned int region_prot,
    unsigned int flags, struct _generic_64 *return_region_id,
    void *(*(return_va_64)), unsigned __int64 *return_length_64,...);
```

Arguments

length_64

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

Length of the virtual region to be created. The length specified must be a multiple of CPU-specific pages. This length is fixed at the time the region is created.

If you want to map multiple memory-resident sections to this region, specify a length large enough not only to accommodate all of the sections, but also to fill the space necessary to align the next section for a maximum of effective page sizes (granularity hints). You can satisfy this requirement by simply allocating a region that is twice as large as the sum of all sections you want to map.

If the flag VA$M_SHARED_PTS is set, this length is rounded up to include an even multiple of CPU-specific pages mapped by a page table page.

region_prot

OpenVMS usage:	region_protection
type:	longword (unsigned)
access:	read only

mechanism:	by value

Region protection to be associated with the region to be created. The *region_prot* argument is a longword containing the create and owner mode.

The file VADEF.H in SYS$STARLET_C.TLB and the $VADEF macro in STARLET.MLB define the following symbols for valid combinations of create and owner modes:

Symbol	Create and Owner Modes
VA$C_REGION_UCREATE_UOWN	User create mode and user owner mode
VA$C_REGION_UCREATE_SOWN	User create mode and supervisor owner mode
VA$C_REGION_UCREATE_EOWN	User create mode and executive owner mode
VA$C_REGION_UCREATE_KOWN	User create mode and kernel owner mode
VA$C_REGION_SCREATE_SOWN	Supervisor create mode and supervisor owner mode
VA$C_REGION_SCREATE_EOWN	Supervisor create mode and executive owner mode
VA$C_REGION_SCREATE_KOWN	Supervisor create mode and kernel owner mode
VA$C_REGION_ECREATE_EOWN	Executive create mode and executive owner mode
VA$C_REGION_ECREATE_KOWN	Executive create mode and kernel owner mode
VA$C_REGION_KCREATE_KOWN	Kernel create mode and kernel owner mode

For both create and owner mode, the $CREATE_REGION_64 service uses whichever of the following two access modes is least privileged:

- Access mode specified by the *acmode* argument.
- Access mode of the caller.

A subsequent call to any system service that created address space within a region must be made from an access mode that is the same or more privileged than the create mode associated with the region.

A subsequent call to $DELETE_REGION_64 to delete the region must be made from an access mode that is the same or more privileged than the owner mode associated with the region.

All regions created by $CREATE_REGION_64 are automatically deleted when the image is run down on image exit.

flags

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

Flag mask specifying the characteristics of the region to be created. The *flags* argument is a longword bit vector in which each bit corresponds to a flag. The file VADEF.H in SYS$STARLET_C.TLB and the $VADEF macro in STARLET.MLB 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 the flag that is valid for the $CREATE_REGION_64 service:

Flag	Description
VA$M_DESCEND	Created region is a descending region; that is, allocation occurs toward decreasing virtual addresses. If VA$M_DESCEND is not specified, the region allocation occurs toward increasing virtual addresses.
VA$M_SHARED_PTS	Created region requires the virtual address space created within it to be capable of using shared page tables. If this flag is not specified, the virtual address space created within the region is mapped by process-private page tables only. By default, the region does not allow the use of shared page tables.
VA$M_P0_SPACE	Create region in P0 space. This flag cannot be set if VA$M_P1_SPACE is set.
VA$M_P1_SPACE	Create region in P1 space. This flag cannot be set if VA$M_P0_SPACE is set.

All other bits in the *flags* argument are reserved to OpenVMS for future use. The condition value SS$_IVREGFLG is returned if any undefined bits are set.

return_region_id_64

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

The region ID associated with the created region. The *return_region_id_64* argument is the 32- or 64-bit virtual address of a naturally aligned quadword into which the service returns the region ID.

return_va_64

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

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

If the flag VA$M_SHARED_PTS is set, the returned virtual address is aligned to a CPU-specific page table page boundary. If the global section mapped by this shared page table region is large enough that multiple page table pages are required to map the global section, the page tables themselves can be mapped with granularity hints. Therefore, the alignment of the returned virtual address can be even greater than that of a single CPU-specific page table page boundary.

return_length_64

OpenVMS usage:	byte count
type:	quadword (unsigned)

access:	write only
mechanism:	by 32- or 64-bit reference

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

If the flag VA$M_SHARED_PTS is set, the returned length is the input length rounded up to an even multiple of bytes mapped by a single CPU-specific page table page.

start_va_64

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

The starting address for the created virtual region. The specified virtual address must be a CPU-specific page aligned address.

If the *start_va_64* argument is not specified or is specified as 0, the region can be created anywhere within the following address spaces:

- P2 space (if the flags VA$M_P0_SPACE and VA$M_P1_SPACE are clear)
- P0 space (if the flag VA$M_P0_SPACE is set and VA$M_P1_SPACE is clear)
- P1 space (if the flag VA$M_P1_SPACE is set and VA$M_P0_SPACE is clear)

If the flag VA$M_SHARED_PTS is set and this argument is specified, the specified starting address must be aligned to the larger of a natural page table boundary or the largest possible page size used to map the section. If the alignment is less than a page table boundary, the $CREATE_REGION_64 service returns an error. If the alignment is less than the largest page size used in the section, an error might be returned when you attempt to map the section.

If you do not specify a starting address, OpenVMS automatically ensures correct alignment.

Description

The Create Virtual Region service allows a process to create a virtual region within its P0, P1, or P2 address space. Once a virtual region has been created, virtual address space can be created within it using the system services that accept a region identifier argument. Note that the virtual region is simply a reservation of virtual address space. No physical memory is occupied for a virtual region until virtual address space is created within the region.

If the VA$M_SHARED_PTS flag is set in the *flags* argument, only memory-resident global sections can be mapped into the virtual region. The $CRMPSC_GDZRO_64 and $MGBLSC_64 system services are available for mapping to memory-resident global sections. If a memory-resident global section was not created with shared page tables, private page tables are used to map to the global section.

If a memory-resident global section with shared page tables is mapped into a virtual region that does not have the shared page table attribute, the global section is mapped with process private page tables. Other

address-space creation services (see Table 18) are not allowed to create address space into a shared page table region because they have an implicit dependency on process-private page tables.

Table 18. Services That Do Not Accept Shared Page Table Regions

Service	Description
$CRETVA[_64]	Adds a range of demand-zero allocation pages to a process's virtual address space for the execution of the current image. The new pages are added at the virtual address specified by the caller.
$CRMPSC	Allows a process to create a private or global section and to map a section of its address space to the private or global section.
$CRMPSC_FILE_64	Allows a process to map a section of its address space to a specified portion of a file. This services maps a private disk file section.
$CRMPSC_GFILE_64	Allows a process to create a global disk file section and to map a section of its address space to the global section.
$CRMPSC_GPFILE_64	Allows a process to create a global page file section and to map a section of its address space to the global section.
$CRMPSC_GPFN_64	Allows a process to create a permanent global page frame section and to map a section of its address space to the global page frame section.
$CRMPSC_PFN_64	Allows a process to map a section of its address space to a specified physical address range represented by page frame numbers. This service creates and maps a private page frame section.
$DELTVA	Deletes a specified number of pages from a process's virtual address space.
$EXPREG_[64][1]	Adds a specified number of demand-zero allocation pages to a process's virtual address space for the execution of the current image. Expansion occurs at the next free available address within the specified region.
$MGBLSC	Establishes a correspondence between pages in the virtual address space of the process and the pages occupied by a global section.
$MGBLSC_GPFN_64	Establishes a correspondence between pages in the virtual address space of the process and the pages occupied by a global page frame section.

[1]$EXPREG can specify only the P0 or P1 region and thus cannot specify a shared page table region.

$CREATE_REGION_64 creates the virtual region on a CPU-specific page aligned boundary. However, if the VA$M_SHARED_PTS flag is set in the *flags* argument, the virtual region is created on a CPU-specific page table page aligned boundary.

It is recommended not to specify the *start_va_64* argument when creating a shared page table region due to the particular alignment that must prevail for virtual addresses created within the virtual region to exploit page table sharing. If the *start_va_64* argument does not contain the proper alignment, $CREATE_REGION_64 returns SS$_VA_NOTPAGALGN.

If a starting virtual address must be specified for a shared page table region, use the following steps to compute a properly aligned *start_va_64*:

1. Determine the CPU-specific page size by using the $GETSYI system service and specifying the SYI$_PAGE_SIZE item code.

2. Determine the number of CPU-specific pages mapped by a single page table page by using the $GETSYI system service and specifying the SYI$_PTES_PER_PAGE item code.

3. Multiply the CPU-specific page size by the number of pages mapped by a page table page. The product represents the minimum virtual alignment required for a shared page table region. It also represents the number of bytes mapped by a single CPU-specific page table page. Assuming a system with an 8 kilobyte page size, the alignment of the *start_va_64* argument must be an even multiple of 8,388,608 (8 megabytes). The virtual address, therefore, must have 23 low-order zero bits.

4. If the shared page tables are to be mapped with granularity hints (GH), the address computed in the previous step should to be adjusted to account for the granularity hint factor:

   - On Alpha systems, granularity hints mean multiples of pages, regardless of page size. The multiples 8, 64, and 512 pages are architected.

   - On Integrity server systems, OpenVMS initially supports page sizes of 64KB, 256KB, and 4MB instead of granularity hints. Additional pages sizes will be supported in the future.

The virtual address alignment factors required for shared page table regions (and mappings using shared page tables) are more stringent than the simple CPU-specific page alignment. Global pages provide a level of data sharing in which the unit is a single CPU-specific page or, on today's systems, 8 kilobytes (KB). Shared page tables increase the level of sharing by an order of magnitude, such that the unit of sharing is a CPU-specific page table page or, on today's systems, 8 megabytes (MB). Therefore, virtual regions that are to be used for shared page tables and mappings that use shared page tables require an alignment of at least 8 MB.

Table 19 highlights the values $CREATE_REGION_64 returns for various region lengths. When the length_64 argument is not even multiple of 8 MB, the returned length is rounded up to an even multiple of 8 MB. This must occur so that a shared page table region ends on an even CPU-specific page table page boundary.

Note

The requirement for CPU-specific page table page multiples for shared page table regions does not imply that memory-resident global sections must also be sized at even CPU-specific page table page multiples. Memory-resident global section must be specified in single CPU-specific page multiples as is the case for global page file sections.

The virtual alignment of the returned address is further biased by the ability to map the shared page tables with granularity hints. All values listed are based upon an 8 KB page size. All of the virtual addresses in the *return_va_64* column accommodate the maximum GH factor for 8 KB page table pages.

Table 19. Sample Returned Values from $CREATE_REGION_64

length_64	***return_va_64***	**return_ length_64**	**Comments**
1,048,576 (1 MB)	FFFFFFFB00800000 at least 23 zero bits	8,388,608 (8 MB)	GH not possible for shared page table pages. Region occupies 1 page table page.
67,108,864 (64 MB)	FFFFFFFBFC000000 at least 26 zero bits	67,108,864 (64 MB)	Returned VA accommodates GH factor of 8 for shared page table pages.

length_64	*return_va_64*	return_ length_64	Comments
73,400,320 (70 MB)	FFFFFFFBF8000000 at least 26 zero bits	75,497,472 (72 MB)	Returned VA accommodates GH factor of 8 for shared page table pages. Region occupies 9 page table pages. Only the first 8 can be mapped with GH.
1,073,741,824 (1 GB)	FFFFFFFBC0000000 at least 30 zero bits	1,073,741,824 (1 GB)	Returned VA accommodates GH factor of 64 for shared page table pages. Region occupies 128 page table pages. In this case, there would be two GH regions, each containing 64 page table pages.

If the returned value of the service is not a successful condition value, a value is not returned in the memory locations pointed to by the *return_region_id_64*, *return_va_64*, or *return_size_64* arguments.

Required Privileges

None

Required Quota

None

Related Services

$CRETVA_64, $CRMPSC_GDZRO, $CRMPSC_FILE_64, $CRMPSC_GFILE_64, $CRMPSC_GPFILE_64, $CRMPSC_GPFN_64, $CRMPSC_PFN_64, $DELETE_REGION_64, $DELTVA_64, $EXPREG_64, $GET_REGION_INFO, $MGBLSC_64, $MGBLSC_GPFN_64

Condition Values Returned

SS$_NORMAL

The service completed successfully.

SS$_ACCVIO

The *return_region_id_64* argument, the *return_va_64* argument, or the *return_length_64* argument cannot be written by the caller.

SS$_IVREGFLG

One or more of the reserved bits in the *flags* argument is set, or an illegal combination of *flags* bits are set.

SS$_LEN_NOTPAGMULT

The *length_64* argument is not a multiple of CPU-specific pages.

SS$_VASFULL

The process private address space is full, or no space is available in the process private address space for a region of the specified size.

SS$_VA_IN_USE

A page in the specified virtual address range is within another virtual region or is otherwise inaccessible.

SS$_VA_NOTPAGALGN

The *start_va_64* argument is not CPU-specific page aligned; or, if the flag VA$M_SHARED_PTS is set, the *start_va_64* argument is not CPU-specific page table page aligned.

$CREATE_UID

Create UID — Generates a universally unique identifier (UID)

Format

```
SYS$CREATE_UID uid
```

C Prototype

```
int sys$create_uid (unsigned int uid [4]);
```

Arguments

uid

OpenVMS usage:	uid
type:	octaword (unsigned)
access:	write only
mechanism:	by reference

Address of an octaword in which the unique identifier is returned to the calling process.

Description

Generates an identifier that is unique across all computer systems.

Required Privileges

None

Required Quotas

None

Related Services

None

Condition Values Returned

SS$_NORMAL

The request was successful.

SS$_ACCVIO

An argument was not accessible to the caller.

$CREATE_USER_PROFILE

Create User Profile — Returns an encoded security profile for the specified user.

Format

```
SYS$CREATE_USER_PROFILE
   usrnam ,[itmlst] ,[flags] ,usrpro  ,usrprolen ,[contxt]
```

C Prototype

```
int sys$create_user_profile
   (void *usrnam, void *itmlst,  unsigned int flags, void *usrpro,
    unsigned int *usrprolen,  unsigned int *contxt);
```

Arguments

usrnam

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

Name of the user whose security profile is to be returned. The *usrnam* argument is the address of a descriptor pointing to a text string containing the user name. The user name string can contain a maximum of 12 alphanumeric characters.

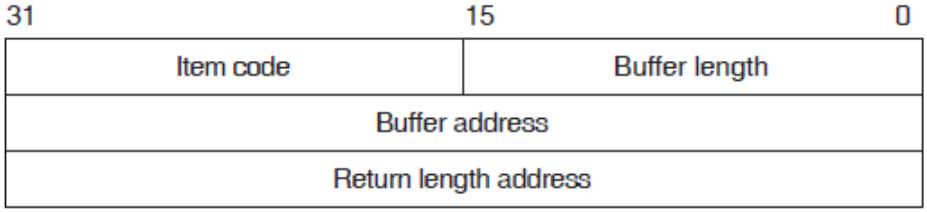
For more information about user names, see the *VSI OpenVMS Guide to System Security*.

itmlst

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

Item list specifying the portions of the user's security profile to be replaced or augmented.

The item list is a standard format item list. The following figure depicts the general format of an item descriptor. See the section called “Item Codes” for a list of valid item codes for $CREATE_USER_PROFILE.



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 from which the service is to read the information. The length of the buffer needed depends on the item code specified in the item code field of the item descriptor.
Item code	A word containing a user-supplied symbolic code specifying the item of information.
Buffer address	A longword containing the user-supplied address of the buffer.
Return length address	A longword that normally contains the user-supplied address of a word in which the service writes the length (in bytes) of the information it returned. This is not used by $CREATE_USER_PROFILE and should contain a 0.

flags

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

The *flags* argument is used for controlling the behavior of the $CREATE_USER_PROFILE service. The following table describes each flag.

Symbol	Description
CHP$M_DEFCLASS	By default, $CREATE_USER_PROFILE initializes the security profile with the user's maximum authorized classification. When this flag is

Symbol	Description
	set, the service initializes the security profile from the user's default classification instead. This flag is reserved to OpenVMS.
CHP$M_DEFPRIV	By default, $CREATE_USER_PROFILE initializes the security profile with the user's authorized privilege mask. When this flag is set, the service initializes the security profile from the user's default privilege mask instead.
CHP$M_NOACCESS	Instructs the service not to access the user authorization file (SYSUAF.DAT) or rights database (RIGHTSLIST.DAT) to build the security profile. This flag can be used as an optimization when all the information necessary to build the security profile is known to the caller.

usrpro

OpenVMS usage:	char_string
type:	opaque byte stream
access:	write only
mechanism:	by descriptor

Buffer to receive the security profile. The *usrpro* argument is the address of a buffer to receive the encoded security profile. If an address of 0 is specified, $CREATE_USER_PROFILE returns the size of the buffer needed in the *usrprolen* argument.

usrprolen

OpenVMS usage:	word
type:	word (unsigned)
access:	read/write
mechanism:	by reference

Word to receive the full size of the security profile. On input, the *usrprolen* argument specifies the length of the buffer pointed to by the *usrpro* argument. The *usrprolen* argument is the address of a word to which $CREATE_USER_PROFILE writes the actual length of the security profile. If the caller specifies a *usrpro* address of 0, $CREATE_USER_PROFILE returns the anticipated size, in bytes, of the buffer needed to hold the user's security profile in the *usrprolen* argument.

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.

Using the *contxt* argument keeps the UAF open across all calls, thereby improving the performance of the system on subsequent calls. To close the UAF, you must run down the image.

The resulting context value from a $CREATE_USER_PROFILE call can also be used as the input *contxt* argument to the $GETUAI system service, and vice versa.

Item Codes

CHP$_ADDRIGHTS

A rights list segment containing additional identifiers to be appended to the set of identifiers held by the user. A rights list segment is a list of quadword identifier/attributes pairs, each containing a longword identifier value, followed by a longword mask identifying the attributes of the holder. The *buflen* argument should be set to the total size, in bytes, of the rights list segment. The *bufadr* argument points to a descriptor that points to the first byte in the rights list segment (that is, the first byte of the first identifier value).

This item code can be repeated to add up to 256 additional rights list segments. If more than 256 identifiers are granted to the user, $CREATE_USER_PROFILE returns SS$_INSFMEM.

CHP$_CLASS

The classification to be associated with the created security profile. This item code is reserved to OpenVMS.

CHP$_PRIV

A quadword privilege mask specifying the user's privileges. The $PRVDEF macro defines the list of available privileges.

CHP$_UIC

A longword describing the user identification code (UIC).

ISS$_ACCOUNT

Variable-length buffer containing the account name. The maximum size of this buffer is 32 bytes.

ISS$_ADD_RIGHTS

A rights list segment containing additional identifiers to be appended to the set of identifiers held by the user. A rights list segment is a list of quadword identifier/attributes pairs, each containing a longword identifier value, followed by a longword mask identifying the attributes of the holder. The *buflen* argument should be set to the total size, in bytes, of the rights list segment. The *bufadr* argument points to a descriptor that points to the first byte in the rights list segment (that is, the first byte of the first identifier value).

This item code can be repeated to add up to 256 additional rights list segments. If more than 256 identifiers are granted to the user, $CREATE_USER_PROFILE returns SS$_INSFMEM.

ISS$_AUTHPRIV

Quadword containing the authorized privileges. See $PRVDEF macro for definitions.

ISS$_FLAGS

Longword containing user flags. The following flag is supported:

ISS$M_FLAG_SECAUDIT - Mandatory audit flag.

ISS$_MAXCLASS

Buffer containing the maximum classification. The maximum size of this buffer is CLS$K_LENGTH. This item code is reserved to OpenVMS. See the $CLSDEF macro for definitions.

ISS$_MINCLASS

Buffer containing the minimum classification. The maximum size of this buffer is CLS$K_LENGTH. This item code is reserved to OpenVMS. See the $CLSDEF macro for definitions.

ISS$_MODE

Longword containing the access mode. See $PSLDEF macro for definitions.

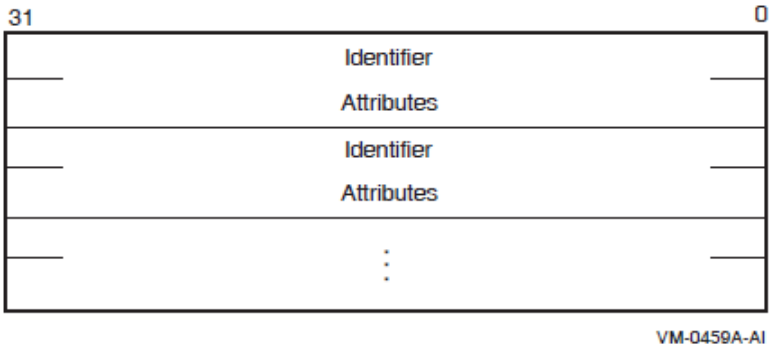
ISS$_PERMPRIV

Quadword containing the permanent privileges. See $PRVDEF macro for definitions.

ISS$_RIGHTS

Descriptor pointing to a vector of quadwords containing identifier/attribute pairs used to initialize the rights identifier list. See the $KGBDEF macro for definitions. Any identifiers specified by the ISS$_ADD_RIGHTS item code will be added to this list.

The format of this vector is as follows:



ISS$_UIC

A longword describing the user identification code (UIC).

ISS$_WORKCLASS The classification to be associated with the created security profile. This item code is reserved to OpenVMS.

ISS$_WORKPRIV

A quadword privilege mask specifying the user's privileges. The $PRVDEF macro defines the list of available privileges.

Description

The Create User Profile service returns a security profile for a user. This profile can be generated in two ways.

- If the caller does not specify the CHP$_NOACCESS flag in the *flags* argument, $CREATE_USER_PROFILE accesses the system authorization database (SYSUAF.DAT) or the rights database (RIGHTSLIST.DAT) for the specified user name and builds a representation of the privileges and rights granted to that user. The security profile is returned as an opaque byte stream.

  $CREATE_USER_PROFILE returns a representation of the security profile that the user would have when logged in at the highest authorized classification with all authorized privileges enabled.

- When the caller specifies the CHP$M_NOACCESS flag in the *flags* argument, $CREATE_USER_PROFILE creates a security profile without accessing the user authorization file (SYSUAF.DAT) or the rights database (RIGHTSLIST.DAT). When CHP$M_NOACCESS is specified, all of the information is obtained from the item list. The caller must supply the CHP$_PRIV and CHP$_UIC items. In addition, an address of 0 can be specified for the *usrnam* argument.

In either case, the newly created security profile can be passed as input to the $CHKPRO and $CHECK_ACCESS system services using the *usrpro* argument.

$CREATE_USER_PROFILE returns the set of identifiers associated with the user's owner identifier. The CHP$_ADDRIGHTS item code can be used to add additional identifiers to this set.

Required Access or Privileges

Access to SYSUAF.DAT and RIGHTSLIST.DAT is required unless you are constructing the security profile for your own user name.

Required Quota

None

Related Services

$CHECK_ACCESS, $CHKPRO, $FIND_HELD, $FINISH_RDB, $GETUAI

Condition Values Returned

SS$_NORMAL

Profile created successfully.

SS$_BADITMCOD

Item list code is invalid.

SS$_BADBUFLEN

Size specified for item is invalid.

SS$_ACCVIO

Buffer address is invalid or inaccessible.

SS$_INSFARG

Insufficient call arguments.

SS$_INSFMEM

Insufficient memory.

SS$_IVSTSFLG

Invalid system service flags specified.

SS$_NOPRIV

Caller lacks privilege to access UAF.

RMS$_RNF

User name is not in UAF.

$CREATE_USER_PROFILE can also return any error returned by the $GETUAI or $FIND_HELD services.

$CRELNM

Create Logical Name — Creates a logical name and specifies its equivalence names. On Alpha and Integrity server systems, this service accepts 64-bit addresses.

Format

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

C Prototype

```
int sys$crelnm
   (unsigned int *attr, void *tabnam, void *lognam, unsigned char *acmode,
    void *itmlst);
```

Arguments

attr

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

Attributes to be associated with the logical name. The *attr* argument is the 32- or 64-bit address of a longword bit mask specifying these attributes.

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

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

The attributes are as follows.

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

tabnam

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

Name of the table in which to create the logical name. The *tabnam* argument is the 32- or 64-bit address of a descriptor that points to the name of this table. This argument is required and must be specified in uppercase.

The name must be entered in uppercase letters. (This requirement differs from the $CRELNT system service, which automatically changes *tabnam* to uppercase).

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

lognam

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

Name of the logical name to be created. The *lognam* argument is the 32- or 64-bit address of a descriptor that points to the logical name string.

Logical name strings of logical names created within either the system or process directory table must consist of uppercase alphanumeric characters, dollar signs ($), hyphens (-), and underscores (_); the maximum length is 31 characters. The maximum length of logical name strings created within other tables is 255 characters with no restrictions on the types of characters that can be used. This argument is required.

acmode

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

Access mode to be associated with the logical name. The *acmode* argument is the 32- or 64-bit address of a byte that specifies the access mode.

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

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

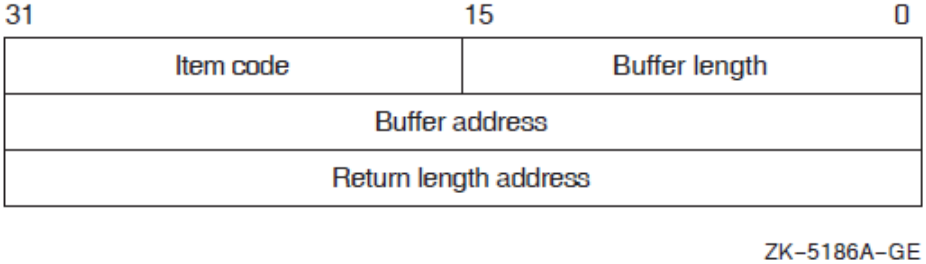
If you omit this argument or specify it as 0, the access mode of the caller is associated with the logical name.

itmlst

OpenVMS usage:	32-bit item_list_3 or 64-bit item_list 64b
type:	longword (unsigned) for 32-bit; quadword (unsigned) for 64-bit
access:	read only
mechanism:	by 32- or 64-bit reference

Item list describing the equivalence names to be defined for the logical name and information to be returned to the caller. The *itmlst* argument is the 32- or 64-bit address of a list of item descriptors, each of which specifies information about an equivalence name. An item list in 32-bit format is terminated by a longword of 0; an item list in 64-bit format is terminated by a quadword of 0. All items in an item list must be of the same format—either 32-bit or 64-bit.

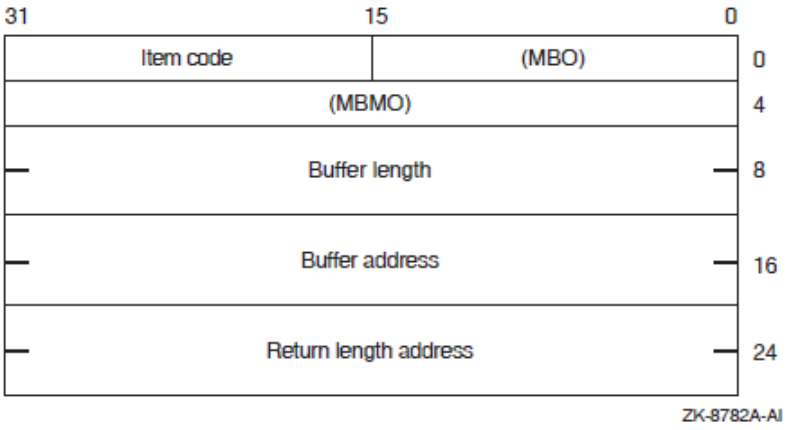
The following diagram depicts the 32-bit format of a single item descriptor.



The following table defines the item descriptor fields for 32-bit item list entries.

Descriptor Field	Definition
Buffer length	A word specifying the number of bytes in the buffer pointed to by the buffer address field. The length of the buffer needed depends on the item code specified in the item code field of the item descriptor. If the value of buffer length is too small, the service truncates the data.
Item code	A word containing a symbolic code that describes the information in the buffer or the information to be returned to the buffer, pointed to by the buffer address field. The item codes are listed in the Item Codes section.
Buffer address	A longword containing the 32-bit address of the buffer that receives or passes information.
Return length address	A longword containing the 32-bit address of a word specifying the actual length in bytes of the information returned by $CRELNM in the buffer pointed to by the buffer address field. The return length address field is used only when the item code specified is LNM$_TABLE. Although this field is ignored for all other item codes, it must nevertheless be present as a placeholder in each item descriptor.

The following diagram depicts the 64-bit format of a single item descriptor.



The following table defines the item descriptor fields for 64-bit item list entries.

Descriptor Field	Definition
MBO	The field must contain a 1. The MBO and MBMO fields are used to distinguish 32-bit and 64-bit item list entries.
Item code	A word containing a symbolic code that describes the information in the buffer or the information to be returned to the buffer, pointed to by the buffer address field. The item codes are listed in the Item Codes section.
MBMO	The field must contain a –1. The MBMO and MBO fields are used to distinguish 32-bit and 64-bit item list entries.
Buffer length	A quadword specifying the number of bytes in the buffer pointed to by the buffer address field. 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, the service truncates the data.
Buffer address	A quadword containing the 64-bit address of the buffer that receives or passes information.

Descriptor Field	Definition
Return length address	A quadword containing the 64-bit address of a word specifying the actual length in bytes of the information returned by $CRELNM in the buffer pointed to by the buffer address field. The return length address field is used only when the item code specified is LNM$_TABLE. Although this field is ignored for all other item codes, it must nevertheless be present as a placeholder in each item descriptor.

Item Codes

LNM$_ATTRIBUTES

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

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

LNM$_CHAIN

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

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

You can chain together 32-bit and 64-bit item lists.

LNM$_STRING

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

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

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

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

Note that it is possible to create a logical that has no equivalence names. This is done by either omitting the *itmlst* argument to $CRELNM, or by not including the LNM$_STRING item code to the *itmlst* data structure that is passed into $CRELNM. It is not possible to create this kind of logical using DCL.

LNM$_TABLE

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

This item code can appear anywhere in the item list.

Description

The Create Logical Name service creates a logical name and specifies its equivalence name. Note that logical names are case sensitive.

Required Access or Privileges

The calling process must have the following:

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

Required Quota

The quota for the specified logical name table must be sufficient for the creation of the logical name.

Related Services

$CRELNT, $DELLNM, $TRNLNM

Condition Values Returned

SS$_NORMAL

The service completed successfully; the logical name has been created. However, if you attempted to create a new clusterwide logical name with the same access mode and identical equivalence names and attributes as an existing clusterwide logical name, this message indicates only that the service completed successfully. Because an identical clusterwide logical name already exists, and because a clusterwide update would adversely affect performance, the name is not created.

SS$_SUPERSEDE

The service completed successfully; the logical name has been created and a previously existing logical name with the same name has been deleted.

SS$_BUFFEROVF

The service completed successfully; the buffer length field in an item descriptor specified an insufficient value, so the buffer was not large enough to hold the requested data.

SS$_ACCVIO

The service cannot access the 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. Or, an item list containing both 32-bit and 64-bit item list entries was found.

SS$_DUPLNAM

An attempt was made to create a logical name with the same name as an already existing logical name, and the existing logical name was created at a more privileged access mode and with the LNM$M_NO_ALIAS attribute.

SS$_EXLNMQUOTA

The quota associated with the specified logical name table for the creation of the logical name is insufficient.

SS$_INSFMEM

The dynamic memory is insufficient for the creation of the logical name, or there is insufficient dynamic memory to build a message describing the creation of a clusterwide name.

SS$_IVLOGNAM

The *tabnam* argument, the *lognam* argument, or the equivalence string specifies a string whose length is not in the required range of 1 through 255 characters. The *lognam* argument specifies a string whose length is not in the required range of 1 to 31 characters for directory table entries.

SS$_IVLOGTAB

The *tabnam* argument does not specify a logical name table.

SS$_NOLOGTAB

Either the specified logical name table does not exist or the logical name translation of the table name exceeded the allowable depth of 10 translations.

SS$_NOPRIV

The caller lacks the necessary privilege to create the logical name.

SS$_TOOMANYLNAM

An attempt was made to create a logical name with more than 128 equivalence names.

$CRELNT

Create Logical Name Table — Creates a process-private or shareable logical name table. On Alpha and Integrity server systems, this service accepts 64-bit addresses.

Format

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

C Prototype

```
int sys$crelnt
   (unsigned int *attr, void *resnam,  unsigned short int *reslen,
    unsigned int *quota, unsigned short int *promsk, void *tabnam,
    void *partab, unsigned char *acmode);
```

Arguments

attr

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

Attributes to affect the creation of the logical name table and to be associated with the newly created logical name table. The *attr* argument is the 32- or 64-bit address of a longword bit mask specifying these attributes.

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

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

The following table describes each attribute.

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

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

resnam

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

Name of the newly created logical name table, returned by $CRELNT. The *resnam* argument is the 32- or 64-bit address of a descriptor pointing to this name. The name is a character string whose maximum length is 31 characters.

reslen

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

Length in bytes of the name of the newly created logical name table, returned by $CRELNT. The *reslen* argument is the 32- or 64-bit address of a word to receive this length.

quota

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

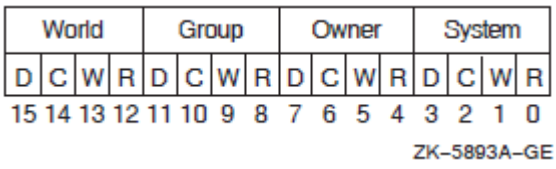
Maximum number of bytes of memory to be allocated for logical names contained in this logical name table. The *quota* argument is the 32- or 64-bit address of a longword specifying this value.

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

promsk

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

Protection mask to be associated with the newly created shareable logical name table. The *promsk* argument is the 32- or 64-bit address of a word that contains a value that represents four 4-bit fields. Each field grants or denies the type of access, either delete, create, write, or read, allowed for system, owner, group, and world users. The following diagram depicts these protection bits.



Create access is required to create a shareable table within another shareable table.

Each field consists of 4 bits specifying protection for the logical name table. The remaining bits in the protection mask are as follows:

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

If a bit is clear, access is granted.

The initial security profile for any shared logical name table is taken from the logical name table template. The owner is then set to the process UIC and, if the *promsk* argument is nonzero, that value replaces the protection mask.

tabnam

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

The name of the new logical name table. The *tabnam* argument is the 32- or 64-bit address of a character-string descriptor pointing to this name string. Table names are contained in either the process or system directory table (LNM$PROCESS_DIRECTORY or LNM$SYSTEM_DIRECTORY); therefore, table names must consist of alphanumeric characters, dollar signs ($), and underscores (_); the maximum length is 31 characters. Names of logical name tables must be in uppercase letters. If you specify a lowercase name, the $CRELNT service automatically changes it to uppercase.

This argument is required for clusterwide logical name tables. For all other logical name tables, if you do not specify this argument, a default name in the format LNM$*xxxx* is used, where *xxxx* is a unique hexadecimal number.

You need SYSPRV privilege or write access to LNM$SYSTEM_DIRECTORY to specify the name of a shareable logical name table.

partab

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

Name string for the parent table name. The *partab* argument is the 32- or 64-bit address of a character string descriptor pointing to this name string.

If the parent table is shareable, then the newly created table is shareable and is entered into the system directory LNM$SYSTEM_DIRECTORY. If the parent table is process-private, then the newly created table is process-private and is entered in the process directory LNM$PROCESS_DIRECTORY.

You need SYSPRV privilege or write access to the system directory to create a named shareable table. This argument is required.

acmode

OpenVMS usage:	access_mode
type:	byte (unsigned)
access:	read only

mechanism: by 32- or 64-bit reference (Alpha and Integrity servers)

Access mode to be associated with the newly created logical name table. The *acmode* argument is the 32- or 64-bit address of a byte containing this access mode. The $PSLDEF macro defines symbolic names for the four access modes.

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

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

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

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

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

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

Description

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

The $CRELNT service uses the following system resources:

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

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

Note that logical names are case sensitive.

Required Access or Privileges

Create access to the parent table and either SYSPRV privilege or write access to the system directory table are required.

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

Required Quota

The parent table must have sufficient quota for the creation of the new table.

Related Services

$CRELNM, $DELLNM, $TRNLNM

Condition Values Returned

SS$_NORMAL

The service completed successfully; the logical name table already exists.

SS$_LNMCREATED

The service completed successfully; the logical name table was created.

SS$_SUPERSEDE

The service completed successfully; the logical name table was created and its logical name superseded the already existing logical names in the directory table.

SS$_ACCVIO

The service cannot access the locations specified by one or more arguments.

SS$_BADPARAM

One or more arguments have an invalid value, or a parent logical name table was not specified.

SS$_DUPLNAM

You attempted to create a logical name table with the same name as an already existing name within the appropriate directory table, and the existing name was created at a more privileged access mode with the LNM$M_NO_ALIAS attribute.

SS$_EXLNMQUOTA

The parent table has insufficient quota for the creation of the new table.

SS$_INSFMEM

The dynamic memory is insufficient for the creation of the table, or there is insufficient dynamic memory to build a message describing the creation of a clusterwide logical name table.

SS$_IVLOGNAM

The *partab* argument specifies a string whose length is not within the required range of 1 to 31 characters.

SS$_IVLOGTAB

The *tabnam* argument is not alphanumeric or specifies a string whose length is not within the required range of 1 to 31 characters, or the TABNAM argument is omitted from a clusterwide $CRELNT call.

SS$_NOLOGTAB

The parent logical name table does not exist.

SS$_NOPRIV

The caller lacks the necessary privilege to create the table.

SS$_PARENT_DEL

The creation of the new table would have resulted in the deletion of the parent table.

SS$_RESULTOVF

The table name buffer is not large enough to contain the name of the new table.

$CREMBX

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

Format

```
SYS$CREMBX
   [prmflg] ,chan ,[maxmsg] ,[bufquo] ,[promsk] ,[acmode] ,[lognam]
   ,[flags] ,[nullarg]
```

C Prototype

```
int sys$crembx
   (char prmflg, unsigned short int *chan, unsigned int maxmsg,
    unsigned int bufquo, unsigned int promsk, unsigned int acmode,
    void *lognam,...);
```

Arguments

prmflg

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

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

chan

OpenVMS usage:	channel

type:	word
access:	write only
mechanism:	by reference

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

maxmsg

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

Maximum size (in bytes) of a message that can be sent to the mailbox. The *maxmsg* argument is a longword value containing this size.

The maximum value you can specify for the *maxmsg* argument is 65535. If you do not specify a value or specify the value as 0, the operating system provides a default value from the DEFMBXBUFQUO system parameter.

bufquo

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

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

For a temporary mailbox, this value must be less than or equal to the process buffer quota.

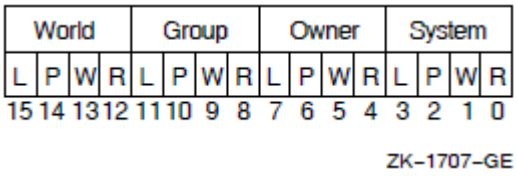
Note that as of Version 7.3-1, the maximum value limit for the *bufquo* argument is the amount of available non-paged pool.

promsk

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

Protection mask to be associated with the created mailbox. The *promsk* argument is a longword value that is the combined value of the bits set in the protection mask. Cleared bits grant access and set bits

deny access to each of the four classes of user: world, group, owner, and system. The following diagram depicts these protection bits.



If you do not specify the *promsk* argument or specify it as 0, the mailbox template is used.

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

The physical access bit is ignored for all categories of user.

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

acmode

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

Access mode to be associated with the channel to which the mailbox is assigned. The *acmode* argument is a longword containing the access mode. The $PSLDEF macro defines the following symbols for the four access modes.

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

The most privileged access mode used is the access mode of the caller. The specified access mode and the access mode of the caller are compared. The less privileged (but the higher numeric valued) of the two access modes becomes the access mode associated with the assigned channel. I/O operations on the channel can be performed only from equal or more privileged access modes.

lognam

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

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

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

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

flags

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

The *flags* argument is used for specifying options for the assign operation that occurs in $CREMBX. The *flags* argument is a longword bit mask that enables the user to specify that the channel assigned to the mailbox is a READ ONLY or WRITE ONLY channel. If the *flags* argument is not specified, then the default channel behavior is READ/WRITE. The $CMBDEF macro defines a symbolic name for each flag bit. The following table describes each flag.

Flag	Description
CMB$M_READONLY	When this flag is specified, $CREMBX assigns a read-only channel to the mailbox device. An attempt to issue a QIO WRITE operation on the mailbox channel results in an illegal I/O operation error.
CMB$M_WRITEONLY	When this flag is specified, $CREMBX assigns a write-only channel to the mailbox device. An attempt to issue a QIO READ operation on the mailbox channel results in an illegal I/O operation error.

For more information about the *flags* argument, see the *VSI OpenVMS I/O User's Reference Manual*.

nullarg

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

Placeholding argument reserved to OpenVMS.

Description

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

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

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

The initial security profile created for a mailbox is taken from the mailbox template for the device class. The owner is then set to the process UIC and the *promsk* argument replaces the protection mask.

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

A channel assigned to the mailbox READ ONLY is considered a READER. A channel assigned to the mailbox WRITE ONLY is considered a WRITER. A channel assigned to the mailbox READ/WRITE is considered both a WRITER and READER.

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

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

The mailbox unit number is determined when the mailbox is created. A process can obtain the unit number of the created mailbox by calling the Get Device/Volume Information ($GETDVI) service using the channel returned by $CREMBX.

Mailboxes are assigned sequentially increasing numbers (from 1 to a maximum of 9999) as they are created. When all unit numbers have been used, the system starts numbering again at unit 1. Logical names or mailbox names should be used to identify a mailbox between cooperating processes.

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

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

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

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

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

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

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

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

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

Required Access or Privileges

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

- TMPMBX privilege whenever the *prmflg* argument is specified as 0. However, a process that has PRMMBX privilege will also meet this requirement.
- PRMMBX privilege whenever the *prmflg* argument is specified as 1.
- SYSNAM privilege to place a logical name for a mailbox in the system logical name table.
- GRPNAM privilege to place a logical name for a mailbox in the group logical name table.

Required Quota

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

Related Services

$ALLOC, $ASSIGN, $BRKTHRU, $BRKTHRUW, $CANCEL, $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 logical name string or string descriptor cannot be read by the caller, or the channel number cannot be written by the caller.

SS$_BADPARAM

One or more of the arguments has an invalid value. One possible problem is the *flags* argument has both the CMB$M_READONLY and CMB$M_WRITEONLY flags set; however, only one of these values is allowed.

SS$_EXBYTLM

The process has insufficient buffer I/O byte count (BYTLM) quota to allocate the mailbox UCB or to satisfy buffer requirements.

SS$_INSFMEM

The system dynamic memory is insufficient for completing the service.

SS$_INTERLOCK

The bit map lock for allocating mailboxes from the specified shared memory is locked by another process.

SS$_IVLOGNAM

The logical name string has a length of 0 or has more than 255 characters.

SS$_IVSTSFLG

The bit set in the *prmflg* argument is undefined; this argument can have a value of 1 or 0.

SS$_NOIOCHAN

No I/O channel is available for assignment.

SS$_NOPRIV

The process does not have the privilege to create a temporary mailbox, a permanent mailbox, a mailbox in memory that is shared by multiple processors, or a logical name.

SS$_NOSHMBLOCK

No shared memory mailbox UCB is available for use to create a new mailbox.

SS$_OPINCOMPL

A duplicate unit number was encountered while linking a shared memory mailbox UCB. If this condition value is returned, contact your VSI support representative.

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.

SS$_TOOMANYLNAM

The logical name translation of the string named in the *lognam* argument exceeded the allowed depth.

$CREPRC

Create Process — Creates, on behalf of the calling process, a subprocess or detached process on the current node, or a detached process on another OpenVMS Cluster node.

Format

```
SYS$CREPRC
   [pidadr] ,[image] ,[input] ,[output] ,[error] ,[prvadr] ,[quota]
   ,[prcnam] ,[baspri] ,[uic] ,[mbxunt] ,[stsflg] ,[itmlst] ,[node]
   ,[home_rad]
```

C Prototype

```
int sys$creprc
   (unsigned int *pidadr, void *image, void *input, void *output,
    void *error, struct _generic_64 *prvadr, unsigned int *quota,
    void *prcnam, unsigned int baspri, unsigned int uic,
    unsigned short int mbxunt, unsigned int stsflg,...);
```

Arguments

pidadr

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

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

image

OpenVMS usage:	logical_name
type:	character-coded text string
access:	read only

mechanism:	by descriptor–fixed-length string descriptor

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

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

To create a process that will run under the control of a command language interpreter (CLI), specify SYS$SYSTEM:LOGINOUT.EXE as the image name.

input

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

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

output

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

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

error

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

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

Note that the *error* argument is ignored if the *image* argument specifies SYS$SYSTEM:LOGINOUT.EXE; in this case, SYS$ERROR has the same equivalence name as SYS$OUPUT.

prvadr

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

Privileges to be given to the created process. The **prvadr** argument is the address of a quadword bit mask wherein each bit corresponds to a privilege; setting a bit gives the privilege. If the **prvadr** argument is not specified, the current privileges are used.

Each bit has a symbolic name; the $PRVDEF macro defines these names. You form the bit mask by specifying the symbolic name of each desired privilege in a logical OR operation.Table 20 gives the symbolic name and description of each privilege.

Table 20. 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$M_ALTPRI	Set (alter) any process priority
AUDIT	PRV$M_AUDIT	Generate audit records
BUGCHK	PRV$M_BUGCHK	Make bugcheck error log entries
BYPASS	PRV$M_BYPASS	Bypass UIC-based protection
CMEXEC	PRV$M_CMEXEC	Change mode to executive
CMKRNL	PRV$M_CMKRNL	Change mode to kernel
DIAGNOSE	PRV$M_DIAGNOSE	Can diagnose devices
DOWNGRADE	PRV$M_DOWNGRADE	Can downgrade classification
EXQUOTA	PRV$M_EXQUOTA	Can exceed quotas
GROUP	PRV$M_GROUP	Group process control
GRPNAM	PRV$M_GRPNAM	Place name in group logical name table
GRPPRV	PRV$M_GRPPRV	Group access via system protection field
IMPERSONATE[1]	PRV$M_IMPERSONATE	Can create detached processes under another UIC
IMPORT	PRV$M_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

Privilege	Symbolic Name	Description
PSWAPM	PRV$M_PSWAPM	Change process swap mode
READALL	PRV$M_READALL	Possess read access to everything
SECURITY	PRV$M_SECURITY	Can perform security functions
SETPRV	PRV$M_SETPRV	Set any process privileges
SHARE	PRV$M_SHARE	Can assign a channel to a non-shared device
SYSGBL	PRV$M_SYSGBL	Create system global sections
SYSLCK	PRV$M_SYSLCK	Queue systemwide locks
SYSNAM	PRV$M_SYSNAM	Place name in system logical name table
SYSPRV	PRV$M_SYSPRV	Access files and other resources as if you have a system UIC
TMPMBX	PRV$M_TMPMBX	Create temporary mailboxes
UPGRADE	PRV$M_UPGRADE	Can upgrade classification
VOLPRO	PRV$M_VOLPRO	Override volume protection
WORLD	PRV$M_WORLD	World process control

[1]This privilege replaces the DETACH privilege; however, the prior mask, PRV$M_DETACH, is still valid for existing programs.

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

quota

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

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

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

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

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

The $PQLDEF macro defines symbolic names for quotas.

In C you can specify a quota list, as follows:

```
#include <pqldef.h>
```

```
...
#pragma member_alignment save
#pragma nomember_alignment
typedef struct
  {
  char Quota;
  int Value;
  } QUOTA_ENTRY_T;
#pragma member_alignment restore
...
  QUOTA_ENTRY_T QuotaArray[] =
    {{PQL$_PRCLM, 2}, {PQL$_ASTLM, 6}, {PQL$_LISTEND, 0}};
```

Individual Quota Descriptions

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

Minimum value	A process cannot be created with a quota less than this minimum. Any quota value you specify is maximized against this minimum. You obtain the minimum value for a quota by running SYSGEN to display the corresponding SYSGEN parameter.
Default value	If the quota list does not specify a value for a particular quota, the system assigns the process this default value. You obtain the default value by running SYSGEN to display the corresponding system parameter.
Deductible quota	When you create a subprocess, the value for a deductible quota is subtracted from the creating process's current quota and is returned to the creating process when the subprocess is deleted. There is currently only one deductible quota, the CPU time limit. Note that quotas are never deducted from the creating process when a detached process is created.
Nondeductible quota	Nondeductible quotas are established and maintained separately for each process and subprocess.
Pooled quota	Pooled quotas are established when a detached process is created, and they are shared by that process and all its descendent subprocesses. Charges against pooled quota values are subtracted from the current available totals as they are used and are added back to the total when they are not being used.

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

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

Minimum values are named PQL_M`xxxxx`, where `xxxxx` are the characters of the quota name that follow “PQL$_” in the quota name.

Default values are named PQL_D`xxxxx`, where `xxxxx` are the characters of the quota name that follow “PQL$_” in the quota name.

Individual Quotas

PQL$_ASTLM

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

Minimum: PQL_MASTLM
Default: PQL_DASTLM
Nondeductible

PQL$_BIOLM

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

Minimum: PQL_MBIOLM
Default: PQL_DBIOLM
Nondeductible

PQL$_BYTLM

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

Minimum: PQL_MBYTLM
Default: PQL_DBYTLM
Pooled

PQL$_CPULM

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

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

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

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

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

Minimum: PQL_MCPULM
Default: PQL_DCPULM
Deductible

PQL$_DIOLM

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

Minimum: PQL_MDIOLM
Default: PQL_DDIOLM
Nondeductible

PQL$_ENQLM

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

Minimum: PQL_MENQLM
Default: PQL_DENQLM
Pooled

PQL$_FILLM

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

Minimum: PQL_MFILLM
Default: PQL_DFILLM
Pooled

PQL$_JTQUOTA

Job table quota. This quota limits the number of bytes of system paged pool used for the job logical name table. If the process being created is a subprocess, this item is ignored. A value of 0 represents an unlimited number of bytes.

Minimum: PQL_MJTQUOTA
Default: PQL_DJTQUOTA
Nondeductible

PQL$_PGFLQUOTA

Paging file quota. This quota limits the number of pagelets (adjusted up or down to represent CPU-specific pages) that can be used to provide secondary storage in the paging file for the execution of a process.

Minimum: PQL_MPGFLQUOTA
Default: PQL_DPGFLQUOTA
Pooled

PQL$_PRCLM

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

Minimum: PQL_MPRCLM
Default: PQL_DPRCLM
Pooled

PQL$_TQELM

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

Minimum: PQL_MTQELM

Default: PQL_DTQELM
Pooled

PQL$_WSDEFAULT

Default working set size. This quota defines the number of pagelets (adjusted up or down to represent CPU-specific pages) in the default working set for any image the process executes. The working set size quota determines the maximum size you can specify for this quota.

Minimum: PQL_MWSDEFAULT
Default: PQL_DWSDEFAULT
Nondeductible

PQL$_WSEXTENT

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

Minimum: PQL_MWSEXTENT
Default: PQL_DWSEXTENT
Nondeductible

PQL$_WSQUOTA

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

Minimum: PQL_MWSQUOTA
Default: PQL_DWSQUOTA
Nondeductible

Use of the Quota List

The values specified in the quota list are not necessarily the quotas that are actually assigned to the created process. The $CREPRC service performs the following steps to determine the quota values that are assigned when you create a process on the same node:

1. It constructs a default quota list for the process being created, assigning it the default values for all quotas. Default values are system parameters and so might vary from system to system.

2. It reads the specified quota list, if any, and updates the corresponding items in the default list. If the quota list contains multiple entries for a quota, only the last specification is used.

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

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

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

Pooled quota values are ignored.

- If a detached process is being created and the creating process has IMPERSONATE or CMKRNL privilege, the resulting value is not compared with the current value of the corresponding quota of the creating process and the resulting value is not deducted from the creating process's quota. A process with IMPERSONATE or CMKRNL privilege is allowed to create a detached process with quota values larger than it has.

When you create a detached process on another OpenVMS Cluster node, the quotas assigned to the process are determined in the following way:

1. The $CREPRC service reads the specified quota list, if any. If it contains multiple entries for a quota, only the last specification is used. If the process does not have IMPERSONATE or CMKRNL privilege, the service compares each value in the list with the current value of the corresponding quota of the creating process and uses the lesser value. It sends the resulting quota list to the node on which the new process is to be created.

2. On that node, the $CREPRC service constructs a default quota list for the process being created, assigning it default values for all quotas based on that node's system parameters.

3. It updates the default list with the corresponding values from the quota list.

4. For each item in the updated quota list, it compares the quota value with the minimum value required based on that node's system parameters and uses the larger value.

prcnam

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

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

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

baspri

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

Base priority to be assigned to the created process. The *baspri* argument is a longword value.

The OpenVMS Alpha and Integrity servers range is 0 to 63, with real-time priorities in the range 32 to 63.

If you want a created process to have a higher priority than its creating process, you must have ALTPRI privilege to raise the priority level. If the caller does not have this privilege, the specified base priority is compared with the caller's priority and the lower of the two values is used.

A process with ALTPRI privilege running on a VAX node can create a process with a priority greater than 31 on an Alpha or Integrity servers node.

If the *baspri* argument is not specified, the priority defaults to 2 for VAX MACRO, VAX BLISS--32, and Pascal; it defaults to 0 for all other languages.

uic

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

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

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

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

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

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

If the *image* argument specifies the SYS$SYSTEM:LOGINOUT.EXE, the UIC of the created process will be the UIC of the caller of $CREPRC, and the UIC parameter is ignored.

mbxunt

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

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

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

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

If you specify the *mbxunt* argument, the mailbox is used when the created process actually terminates. At that time, the $ASSIGN service is issued for the mailbox in the context of the terminating process and

an accounting message is sent to the mailbox. If the mailbox no longer exists, cannot be assigned, or is full, the error is treated as if no mailbox had been specified.

If you specify this argument when you create a process on another node, an accounting message will be written to the mailbox when the process terminates. If the node is removed from the cluster before the created process terminates, an accounting message will be simulated. The simulated message will contain the created process's PID and name and a final status of SS$_NODELEAVE, but will lack execution statistics.

Note that two processes on different nodes cannot use the termination mailbox for general interprocess communication.

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

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

The $ACCDEF macro defines symbolic names for offsets of fields within the accounting message. The offsets, their symbolic names, and the contents of each field are shown in the following table. Unless stated otherwise, the length of the field is 4 bytes.

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

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

stsflg

OpenVMS usage: mask_longword

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

Options selected for the created process. The *stsflg* argument is a longword bit vector wherein a bit corresponds to an option. Only bits 0 to 18 are used; the others are reserved and must be 0.

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

Symbolic Name	**Description**
PRC$M_BATCH	Create a batch process. IMPERSONATE privilege is required.
PRC$M_IMPERSONATE	Create a detached process under another UIC.
PRC$M_DISAWS	Disable system-initiated working set adjustment.
PRC$M_HIBER	Force process to hibernate before it executes the image.
PRC$M_HOME_RAD	Assign process to specified home resource affinity domain (RAD). RAD is supported on AlphaServer GS series systems and starting from OpenVMS Version 8.4, support is extended to NUMA capable Integrity servers.
PRC$M_IMGDMP	Enable image dump facility. If an image terminates due to an unhandled condition, the image dump facility writes the contents of the address space to a file in your current default directory. The file name is the same as the name of the terminated image. The file type is .DMP.
PRC$M_INTER	Create an interactive process. This option is meaningful only if the *image* argument specifies SYS$SYSTEM:LOGINOUT.EXE. The purpose of this option is to provide you with information about the process. When you specify this option, it identifies the process as one that is in communication with another user (an interactive process). For example, if you use the DCL lexical function F$MODE to make an inquiry about a process that has specified the PRC$M_INTER option, F$MODE returns the value INTERACTIVE.
PRC$M_KT_LIMIT	Assign the specified kernel thread limit to the created process.
PRC$M_NETWRK	Create a process that is a network connect object. IMPERSONATE privilege required.
PRC$M_NOACNT	Do not perform accounting. ACNT privilege is required.
PRC$M_NOPASSWORD	Do not display the *Username:* and *Password:* prompts if the process is interactive and detached and the image is SYS$SYSTEM:LOGINOUT.EXE. If you specify this option in your call to $CREPRC, the process created by the call is logged in under the user name associated with the creating process. If you do not specify this option for an interactive process, SYS$SYSTEM:LOGINOUT.EXE prompts you for the user name and password to be associated with the process. The prompts are displayed at the SYS$INPUT device.

Symbolic Name	Description
PRC$M_NOUAF	Do not check authorization file if the process is detached and the image is SYS$SYSTEM:LOGINOUT.EXE. You should not specify this option if a subprocess is being created. In previous versions of the operating system, the symbolic name of this option was PRC$M_LOGIN. The symbolic name has been changed to more accurately denote the effect of setting this bit. For compatibility with existing user programs, you can still specify this bit as PRC$M_LOGIN. This flag prevents the loading of the new process's security profile from the contents of the UAF record associated with the specified user name. Restrictions are still enforced on the UAF record, if it exists, for account disuser, account expiration, and primary/secondary days/hours.
PRC$M_PARSE_EXTENDED	Sets the PARSE_STYLE_PERM and the PARSE_STYLE_IMAGE properties for the new process to EXTENDED.
PRC$M_PSWAPM	Inhibit process swapping. PSWAPM privilege is required.
PRC$M_SSFEXCU	Enable system service failure exception mode.
PRC$M_SSRWAIT	Disable resource wait mode.
PRC$M_SUBSYSTEM	Inherit any protected subsystem identifiers. The default is that the new process does not inherit subsystem identifiers.
PRC$M_TCB	Mark a process as part of the trusted computing base (TCB). As such, it is expected to perform its own auditing. IMPERSONATE privilege is required.

Note that options PRCM_BATCH, PRCM_INTER, PRCM_NOUAF, PRCM_NETWRK, and PRC$M_NOPASSWORD are intended for OpenVMS use.

itmlst

OpenVMS usage:	reserved
type:	longword (unsigned)

The *itmlst* argument is reserved to OpenVMS.

node

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

Name of the OpenVMS Cluster node on which the process is to be created. The *node* argument is the address of a character string descriptor pointing to a 1- to 6-character SCS node name string. If the argument is present but zero or if the string is zero length, the process is created on the current node.

home_rad

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

Sets the home resource affinity domain (RAD) of a process.

The home RAD is determined by the operating system, unless you explicitly request one. If bit PRC$M_HOME_RAD in the *stsflg* is set, *home_rad* is the RAD on which the process is to start. Note that you may set this bit to 0 on non-RAD systems.

RAD is supported on AlphaServer GS series systems and starting from OpenVMS Version 8.4, support is extended to NUMA capable Integrity servers. For more information about using RADs, see the *VSI OpenVMS Alpha Partitioning and Galaxy Guide*.

kt_limit

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

Sets the limit of the number of kernel threads that can be created in the process. If the value is greater than the SYSGEN MULTITHREAD parameter, an error message is returned.

The number of kernel threads that can be created in a process, is by default controlled by the MULTITHREAD SYSGEN parameter. The *kt_limit* argument is used to further limit the number of possible kernel threads for the process.

Description

The Create Process service creates a subprocess or detached process on behalf of the calling process. A subprocess can be created only on the current OpenVMS Cluster node. A detached process can be created on the current OpenVMS Cluster node or on the node specified with the *node* argument.

A detached process is a fully independent process. For example, the process that the system creates when you log in is a detached process. A subprocess, on the other hand, is related to its creating process in a treelike structure; it receives a portion of the creating process's resource quotas and must terminate before the creating process. Any subprocesses that still exist when their creator is being deleted are automatically deleted.

The presence of the *uic* argument, *node* argument, or the PRC$M_IMPERSONATE flag specifies that the created process is detached.

Creating a process is synchronous in that the process has actually been created and its PID determined before control returns to the program that requested the system service. Note, however, that the new process has not necessarily begun to execute at that point. Some error conditions are not detected until the created process executes. These conditions include an invalid or nonexistent image; invalid SYS$INPUT, SYS$OUTPUT, or SYS$ERROR logical name equivalence; inadequate quotas; or insufficient privilege to execute the requested image.

In creating a detached or subprocess, you can specify that the process run the image SYS$SYSTEM:LOGINOUT.EXE. During interactive logins, LOGINOUT performs the following functions:

1. It validates user name and password.
2. It reads the system authorization file record associated with that user and redefines the process environment based on information from the record.
3. It maps a command language interpreter (CLI) into the process and passes control to it.

The CLI reads a command from SYS$INPUT, processes it, and reads another command. The presence of the CLI enables the process to execute multiple images. It also enables an image running in the process to use Run-Time Library procedures, such as LIB$SPAWN, LIB$DO_COMMAND, and LIB$SET_LOGICAL, that require a CLI.

Running in the context of a process you create through $CREPRC, LOGINOUT can perform some or all of the preceding steps, depending on whether the process is a subprocess or a detached process and on the values of PRC$M_NOPASSWORD and PRC$M_NOUAF in the *stsflg* argument.

Certain characteristics of a created process can be specified explicitly through $CREPRC system service arguments, while other characteristics are propagated implicitly from the $CREPRC caller. Implicit characteristics include the following:

- Current default directory
- Creator's equivalence name for SYS$DISK
- User and account names
- Command language interpreter (CLI) name and command table file name

Note, however, that after the process has been created, if it runs LOGINOUT and LOGINOUT redefines the process environment, those characteristics will be overridden by information from the system authorization file.

Several process characteristics are relevant to the creation of a process on another OpenVMS Cluster node, in particular, process quotas, default directory, SYS$DISK equivalence name, CLI name, and CLI command table name.

Quotas for a process created on another OpenVMS Cluster node are calculated as previously described in the section on the use of the quota list; namely, they are based on explicit values passed by the creator and system parameters on the other OpenVMS Cluster node. If the other node has its own authorization file with node-specific quotas, you might want to specify in the $CREPRC request that the process run LOGINOUT so it can redefine the process environment based on that node's quotas for the user.

Unless overridden by LOGINOUT, the new process will use its creator's default disk and directory. If the disk is not mounted clusterwide, the created process might need to redefine SYS$DISK with an equivalence name that specifies a disk accessible from that node.

When you set the PRC$M_NOUAF flag in the *stsflg* argument and create a process running LOGINOUT, LOGINOUT will attempt to map a CLI and command table with the same file names as those running in your process. The CLI and command table images must therefore have already been installed by the system manager on the other node. Problems can arise when you are using something other than the DCL CLI and its standard command tables. For example, if you are running on a VAX

node with MCR as your current CLI, LOGINOUT will be unable to map that CLI on an Alpha node. The new process will be created but then aborted by LOGINOUT.

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

The $CREPRC service requires system dynamic memory.

Required Access or Privileges

The calling process must have the following:

- IMPERSONATE or CMKRNL privilege to create any of the following types of process:
  - A detached process with a UIC that is different from the UIC of the calling process
  - A detached process with a larger value specified for some quota than is authorized for the caller
  - A detached process on another node if the system parameter CWCREPRC_ENABLE has a value of 0
- IMPERSONATE privilege to create any of the following types of process:
  - A batch process
  - A network process
  - A trusted computing base process
- ALTPRI privilege to create a subprocess with a higher base priority than the calling process
- SETPRV privilege to create a process with privileges that the calling process does not have
- PSWAPM privilege to create a process with process swap mode disabled
- ACNT privilege to create a process with accounting functions disabled
- OPER privilege to create a detached process on another OpenVMS Cluster node on which interactive logins have not yet been enabled

Required Quota

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

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

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

Related Services

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

Condition Values Returned

SS$_ACCVIO

The caller cannot read a specified input string or string descriptor, the privilege list, or the quota list; or the caller cannot write the process identification.

SS$_BADRAD

The specified RAD contains no memory or contains no active CPUs, or the specified RAD is greater than or equal to the maximum number of RADs on the system. Use the $GETSYI item code RAD_MAX_RADS to determine the maximum number of RADs on the system.

SS$_DUPLNAM

The specified process name duplicates one already specified within that group.

SS$_EXPRCLM

The creation of a detached process failed because the creating process already reached its limit for the creation of detached processes. This limit is established by the MAXDETACH quota in the user authorization file (UAF) of the creating process.

SS$_EXQUOTA

At least one of the following conditions is true:

- The process has exceeded its quota for the creation of subprocesses.
- A quota value specified for the creation of a subprocess exceeds the creating process's corresponding quota.
- The quota is deductible and the remaining quota for the creating process would be less than the minimum.

SS$_INCOMPAT

The remote node is running an incompatible version of the operating system, namely, one that does not support remote process creation.

SS$_INSFMEM

The system dynamic memory is insufficient for the requested operation.

SS$_INVARG

An invalid argument is specified.

SS$_INVKTLIM

A value lower than 0, or a value higher than the SYSGEN parameter MULTITHREAD is specified.

SS$_IVLOGNAM

At least one of the following two conditions is true:

- The specified process name has a length of 0 or has more than 15 characters.
- The specified image name, input name, output name, or error name has more than 255 characters.

SS$_IVQUOTAL

The quota list is not in the proper format.

SS$_IVSTSFLG

A reserved status flag was specified.

SS$_NODELEAVE

The specified node was removed from the OpenVMS Cluster during the $CREPRC service's execution.

SS$_NOPRIV

The caller violated one of the privilege restrictions.

SS$_NORMAL

The service completed successfully.

SS$_NOSLOT

No process control block is available; in other words, the maximum number of processes that can exist concurrently in the system has been reached.

SS$_NOSUCHNODE

The specified node is not currently a member 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.

$CRETVA

Create Virtual Address Space — Adds a range of demand-zero allocation pagelets to a process's virtual address space for the execution of the current image.

Format

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

C Prototype

```
int sys$cretva
   (struct _va_range *inadr, struct _va_range *retadr,
    unsigned int acmode);
```

Arguments

inadr

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

Address of a 2-longword array containing the starting and ending virtual addresses of the pages to be created. If the starting and ending virtual addresses are the same, a single page is created. The addresses are adjusted up or down to fall on CPU-specific page boundaries. Only the virtual page number portion of the virtual address is used; the low order byte-within-page bits are ignored.

retadr

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

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

On Alpha and Integrity server systems, the *retadr* argument should be checked by programs for actual allocation. Because the Alpha and Integrity servers architectures define more than one page size, more space might be created than was specified in the *inadr* argument.

acmode

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

Access mode and protection for the new pages. The *acmode* argument is a longword containing the access mode. The $PSLDEF macro defines the following symbols for the four access modes.

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

Symbol	Access Mode
PSL$C_USER	User

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

Description

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

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

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

If $CRETVA creates pages that already exist, the service deletes those pages if they are not owned by a more privileged access mode than that of the caller. Any such deleted pages are reinitialized as demand-zero pages. For this reason, it is important to use the *retadr* argument to capture the address range actually created. Because the Alpha and Integrity servers architectures have a larger page size than the VAX architecture, more space is potentially affected on Alpha and Integrity server systems.

Required Access or Privileges

None

Required Quota

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

Related Services

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

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

Note

Do not use the $CRETVA system service in conjunction with other user-written procedures or procedures supplied by VSI (including Run-Time Library procedures). This system service provides no means to communicate a change in virtual address space with other routines. VSI recommends that you use either $EXPREG or the Run-Time Library procedure Allocate Virtual Memory (LIB$GET_VM) to get memory. You can find documentation on LIB$GET_VM in the *VSI OpenVMS RTL Library (LIB $) Manual*. When using $DELTVA, you should take care to delete only pages that you have specifically created.

Condition Values Returned

SS$_NORMAL

The service completed successfully.

SS$_ACCVIO

The *inadr* argument cannot be read by the caller, or the *retadr* argument cannot be written by the caller.

SS$_EXQUOTA

The process has exceeded its paging file quota.

SS$_INSFWSL

The process's working set limit is not large enough to accommodate the increased size of the virtual address space.

SS$_NOPRIV

A page in the specified range is in the system address space.

SS$_NOSHPTS

A virtual address within a shared page table region was specified.

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$_VA_IN_USE

The existing underlying page cannot be deleted because it is associated with a buffer object.

SS$_VASFULL

The process's virtual address space is full; no space is available in the page tables for the requested pages.

$CRETVA_64 (Alpha and Integrity servers)

Create Virtual Address Space — On Alpha and Integrity server systems, adds a range of demand-zero allocation pages to a process's virtual address space for the execution of the current image. The new pages are added at the virtual address specified by the caller. This service accepts 64-bit addresses.

Format

```
YS$CRETVA_64
   region_id_64 ,start_va_64 ,length_64 ,acmode ,flags ,return_va_64
```

```
   ,return_length_64
```

C Prototype

```
int sys$cretva_64
   (struct _generic_64 *region_id_64, void *start_va_64,
    unsigned __int64 length_64, unsigned int acmode, unsigned int flags,
    void *(*(return_va_64)), unsigned __int64 *return_length_64);
```

Arguments

region_id_64

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

The region ID associated with the region to create the virtual address range. 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. Also, given a particular virtual address, the region ID for the region it is in can be obtained by calling the $GET_REGION_INFO system service specifying the VA$_REGSUM_BY_VA function.

start_va_64

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

The starting address for the created virtual address range. The specified virtual address must be a CPU-specific page-aligned address.

length_64

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

mechanism:	by value

Length of the virtual address space to be created. The length specified must be a multiple of CPU-specific pages.

acmode

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

Access mode associated with the call to $CRETVA_64. The access mode determines the owner mode of the pages as well as the read and write protection on the pages. 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 $CRETVA_64 service uses whichever of the following access modes is least privileged:

- Access mode specified by the *acmode* argument.
- Access mode of the caller.

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

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 controlling the characteristics of the demand-zero pages created. The *flags* argument is a longword bit vector in which each bit corresponds to a flag. The $VADEF macro and the VADEF.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 the flag that is valid for the $CRETVA_64 service:

Flag	Description
VA$M_NO_OVERMAP	Pages cannot overmap existing address space. By default, pages can overmap existing address space.

All other bits in the flags argument are reserved for future OpenVMS use and should be specified as 0. The condition value SS$_IVVAFLG is returned if any undefined bits are set.

return_va_64

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

The lowest process virtual address of the created virtual address range. The *return_va_64* argument is the 32- 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- or 64-bit reference

The length of the virtual address range created. The *return_length_64* argument is the 32- 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 Create Virtual Address Space service is a kernel mode service that can be called from any mode. The service adds a range of demand-zero allocation pages, starting at the virtual address specified by the *start_va_64* argument. The pages are added to a process's virtual address space for the execution of the current image. Expansion occurs at the next free available address within the specified region if the range of addresses is beyond the next free available address.

The new pages, which were previously inaccessible to the process, are created as demand-zero pages.

The returned address is always the lowest virtual address in the range of pages created. The returned length is always an unsigned byte count indicating the length of the range of pages created.

Successful return status from $CRETVA means that the specified address space was created of the size specified in the *length_64* argument.

If $CRETVA_64 creates pages that already exist, the service deletes those pages if they are not owned by a more privileged access mode than that of the caller. Any such deleted pages are reinitialized as demand-zero pages.

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 an address within the specified address range is not within the bounds of the specified region, the condition value SS$_PAGNOTINREG is returned.

If a condition value other than SS$_ACCVIO is returned, the returned address and returned length indicate the pages that were successfully added before the error occurred. If no pages were added, 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

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

Related Services

$CREATE_BUFOBJ_64, $CREATE_REGION_64, $DELETE_REGION_64, $DELTVA_64, $EXPREG_64, $LCKPAG_64, $LKWSET_64, $PURGE_WS, $SETPRT_64, $ULKPAG_64, $ULWSET_64

Condition Values Returned

SS$_NORMAL

The service completed successfully.

SS$_ACCVIO

The *return_va_64* or *return_length_64* argument cannot be written by the caller.

SS$_EXPGFLQUOTA

The process has exceeded its paging file quota.

SS$_INSFWSL

The process's working set limit 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$_IVREGID

An invalid region ID was specified.

SS$_IVVAFLG

An invalid flag, a reserved flag, or an invalid combination of flags and arguments was specified.

SS$_LEN_NOTPAGMULT

The *length_64* argument is not a multiple of CPU-specific pages.

SS$_NOSHPTS

The region ID of a shared page table region was specified.

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.

SS$_VA_IN_USE

A page in the specified range is already mapped, and the VA$M_NO_OVERLAP flag was set, or the existing underlying page cannot be deleted because it is associated with a buffer object.

SS$_VA_NOTPAGALGN

The *start_va_64* argument is not CPU-specific page aligned.

$CRMPSC

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

Format

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

C Prototype

```
int sys$crmpsc
   (struct _va_range *inadr, struct _va_range *retadr, unsigned int acmode
    unsigned int flags, void *gsdnam, struct _secid *ident, unsigned int
 relpag,
    unsigned short int chan, unsigned int pagcnt, unsigned int vbn,
    unsigned int prot,unsigned int pfc);
```

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, an Integrity servers or a VAX system. The two system types are discussed separately in this section.

Alpha and Integrity servers System Usage

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

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. Because the user stack is mapped in P1, it is possible that references to the user stack might inadvertently read or write the pages mapped with $CRMPSC.

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 and Integrity server systems, bit 31 (the most significant bit) of the first *inadr* longword *must be* 0. To ensure compatibility between VAX and Alpha or Integrity server systems when you choose a region, VSI recommends that you specify, for the first *inadr* longword, any virtual address in the desired region.

In general, the *inadr* argument should be specified; however, it can be omitted to request a special feature: for permanent global sections, you can omit the *inadr* argument, or specify it as 0, to request that the section be created but not mapped. Such a request will be granted regardless of the setting of the SEC$M_EXPREG flag; however, to ensure compatibility between VAX and Alpha or Integrity

server systems, VSI recommends that the SEC$M_EXPREG flag be clear when the *inadr* argument is omitted.

retadr

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

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

On Alpha and Integrity server systems, the *retadr* argument returns starting and ending addresses of the *usable*range of addresses. This might 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 that is to be the owner of the pages created during the mapping. The *acmode* argument is a longword containing the access mode. The $PSLDEF macro defines the following symbols for the four access modes.

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

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

flags

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

Flag mask specifying the type of section to be created or mapped to, as well as its characteristics. The *flags* argument is a longword bit vector wherein each bit corresponds to a flag. The $SECDEF macro

defines a symbolic name for each flag. You construct the *flags* argument by performing a logical OR operation on the symbol names for all desired flags.

The following table describes each flag and the default value that it supersedes.

Flag	Description
SEC$M_CRF	Pages are copy-on-reference. By default, pages are shared.
SEC$M_DZRO	Pages are demand-zero pages. By default, they are not zeroed when copied.
SEC$M_EXECUTE	Pages are mapped if the caller has execute access. This flag takes effect only (1) when specified from executive or kernel mode, (2) when the SEC$M_GBL flag is also specified, and (3) when SEC$M_WRT is not specified. By default $CRMPSC performs a read access check against the section.
SEC$M_EXPREG	Pages are mapped into the first available space. By default, pages are mapped into the range specified by the *inadr* argument. See the *inadr* argument description for a complete explanation of how to set the SEC$M_EXPREG flag.
SEC$M_GBL	Pages form a global section. The default is private section.
SEC$M_NO_OVERMAP	Pages cannot overmap existing address space. Note that, by default, pages can overmap existing address space.
SEC$M_PAGFIL	Pages form a global page file section. By default, pages form a disk file section. SEC$M_PAGFIL also implies SEC$M_WRT and SEC$M_DZRO.
SEC$M_PERM	Global section is permanent. By default, global sections are temporary.
SEC$M_PFNMAP	Pages form a page frame section. By default, pages form a disk file section. Pages mapped by SEC$M_PFNMAP are not included in or charged against the process's working set; they are always valid. Do not lock these pages in the working set by using $LKWSET; this can result in a machine check if they are in I/O space. [1]On Alpha and Integrity server systems, when the SEC$M_PFNMAP flag is set, the *pagcnt* and *relpag* arguments are interpreted in CPU-specific pages, *not* as pagelets.
SEC$M_SYSGBL	Pages form a system global section. By default, pages form a group global section.
SEC$M_UNCACHED	Flag that must be set when a PFN-mapped section is created if this section is to be treated as uncached memory. Flag is ignored on Alpha systems; it applies only to Integrity server systems.
SEC$M_WRT	Pages form a read/write section. By default, pages form a read-only section.

[1]Alpha and Integrity servers specific

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

You can specify any name from 1 to 43 characters. All processes mapping to the same global section must specify the same name. Note that the name is case sensitive.

Use of characters valid in logical names is strongly encouraged. Valid values include alphanumeric characters, the dollar sign ($), and the underscore (_). If the name string begins with an underscore (_), the underscore is stripped and the resultant string is considered to be the actual name. Use of the colon (:) is not permitted.

Names are first subject to a logical name translation, after the application of the prefix GBL$ to the name. If the result translates, it is used as the name of the section. If the resulting name does not translate, the name specified by the caller is used as the name of the section.

Additional information on logical name translations and on section name processing is available in the *VSI OpenVMS Programming Concepts Manual*.

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

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

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

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

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

relpag

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

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

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

On Alpha or Integrity servers, you use this argument only for global sections. If you do not specify the *relpag* argument or specify it as 0 (the default), the global section is mapped beginning with the first virtual block in the file.

chan

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

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

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

pagcnt

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

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

On Alpha and Integrity server systems, the smallest allocation is an Alpha or Integrity servers page, which is 8192 bytes. When requesting pagelets, the size requested is a multiple of 512 bytes, but the actual allocation is rounded to 8192. For example, when requesting 17 pagelets, the allocation is for two Alpha or Integrity servers pages, 16384 bytes.

On Alpha and Integrity server systems, if the SEC$M_PFNMAP flag bit is set, the *pagcnt* argument is interpreted as CPU-specific pages, *not* as pagelets.

On Alpha or Integrity server systems, the specified page count is compared with the number of blocks in the section file; if they are different, the lower value is used. If you do not specify the page count or specify it as 0 (the default), the size of the section file is used. However, for physical page frame sections, this argument must not be 0.

vbn

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

Virtual block number in the file that marks the beginning of the section. The *vbn* argument is a longword containing this number. If you do not specify the *vbn* argument or specify it as 0 (the default), the section is created beginning with the first virtual block in the file.

If you specified page frame number mapping (by setting the SEC$M_PFNMAP flag), the *vbn* argument specifies the CPU-specific page frame number where the section begins in memory.

Table 21 shows which arguments are required and which are optional for three different uses of the $CRMPSC service.

Table 21. Required and Optional Arguments for the $CRMPSC Service

Argument	Create/Map Global Section	Map Global[1] Section	Create/Map Private Section
inadr	Optional[2]	Required	Required
retadr	Optional	Optional	Optional
acmode	Optional	Optional	Optional
flags			
SEC$M_GBL	Required	Ignored	Not used
SEC$M_CRF[3]	Optional	Not used	Optional
SEC$M_DZRO[3]	Optional	Not used	Optional
SEC$M_EXPREG	Optional	Optional	Optional
SEC$M_PERM	Optional[2]	Not used	Not used
SEC$M_PFNMAP	Optional	Not used	Optional
SEC$M_SYSGBL	Optional	Optional	Not used
SEC$M_WRT	Optional	Optional	Optional
SEC$M_PAGFIL	Optional	Not used	Not used
gsdnam	Required	Required	Not used
ident	Optional	Optional	Not used
relpag[3]	Optional	Optional	Not used
chan[3]	Required		Required
pagcnt	Required		Required
vbn[3]	Optional		Optional

Argument	Create/Map Global Section	Map Global[1] Section	Create/Map Private Section
prot	Optional		Not used
pfc[3]	Optional		Optional

[1]The Map Global Section ($MGBLSC) service maps an existing global section.

[2]See the description of *inadr* for the rules governing the omission of the argument.

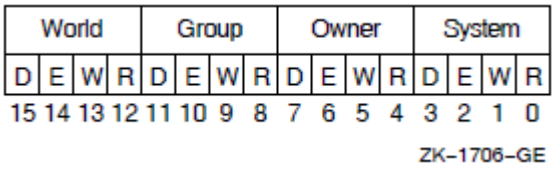
[3]For physical page frame sections: *vbn* specifies the starting page frame number; *chan* must be 0; *pfc* is not used; and the SEC$M_CRF and SEC$M_DZRO flag bit settings are invalid. For page-file sections, *chan* must be 0 and *pfc* not used.

prot

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

Protection to be applied to the global page file and PFN sections. For file-backed sections, the protection is taken from the backing file and the *prot* argument is ignored.

The mask contains four 4-bit fields. Bits are read from right to left in each field. The following diagram depicts the mask.



Cleared bits indicate that read, write, execute, and delete access, in that order, are granted to the particular category of user.

Only read, write, and execute access are meaningful for section protection. Delete access bits are ignored. Read access also grants execute access for those situations where execute access applies.

Protection is taken from the system or group global section template for page file or PFN global sections if the *prot* argument is not specified.

pfc

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

Page fault cluster size indicating how many pagelets are to be brought into memory when a page fault occurs for a single page.

On Alpha and Integrity server systems, this argument is not used for page file sections or physical page frame sections. The *pfc* argument is rounded up to CPU-specific pages. That is, at least 16 pagelets (on an Alpha or Integrity servers system with an 8KB page size) will be mapped for each physical page. The system cannot map less than one physical page.

Description

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

Creating a disk file section involves defining all or part of a disk file as a section. Mapping a disk file section involves making a correspondence between virtual blocks in the file and pagelets in the caller's virtual address space. If the $CRMPSC service specifies a global section that already exists, the service maps it.

Any section created is created as entire pages. See the memory management section in the *VSI OpenVMS Programming Concepts Manual*.

Depending on the actual operation requested, certain arguments are required or optional. Table 21 summarizes how the $CRMPSC service interprets the arguments passed to it and under what circumstances it requires or ignores arguments.

The $CRMPSC service returns the virtual addresses of the virtual address space created in the `retadr` argument, if specified. The section is mapped from a low address to a high address, whether the section is mapped in the program or control region.

If an error occurs during the mapping of a global section, the `retadr` argument, if specified, indicates the pages that were successfully mapped when the error occurred. If no pages were mapped, the value of the longwords is indeterminate. In this case, either both longwords of the `retadr` argument will contain the value –1, or the value of the longwords will be unaltered.

The SEC$M_PFNMAP flag setting identifies the memory for the section as starting at the page frame number specified in the `vbn` argument and extending for the number of CPU-specific pages specified in the `pagcnt` argument. Setting the SEC$M_PFNMAP flag places restrictions on the following arguments.

Argument	**Restriction**
chan	Must be 0
pagcnt	Must be specified; cannot be 0
vbn	Specifies first page frame to be mapped
pfc	Does not apply
SEC$M_CRF	Must be 0
SEC$M_DZRO	Must be 0
SEC$M_PERM	Must be 1 if the flags SEC$M_GBL or SEC$M_SYSGBL are set

Setting the SEC$M_PAGFIL flag places the following restrictions on the following flags.

Flag	**Restriction**
SEC$M_CRF	Must be 0
SEC$M_DZRO	Assumed to be 0
SEC$M_GBL	Must be 1

Flag	Restriction
SEC$M_PFNMAP	Must be 0
SEC$M_WRT	Assumed to be 0

The *flags* argument bits 4 through 13 and 18 through 31 must be 0.

If the global section is mapped to a file (neither SEC$M_PAGFIL nor SEC$M_PFNMAP is set), the security profile of the file is used to determine access to the global section.

On VAX systems, by default, the initial security profile created for a page file or PFN global section is taken from the group global section template. If the SEC$M_SYSGBL flag is set, the profile is taken from the system global section template. The owner is then set to the process UIC. If the *prot* argument is nonzero, it replaces the protection mask from the template.

On Alpha or Integrity server systems , the flag bit SEC$M_WRT applies only to the way in which the newly created section is mapped. For a file to be made writable, the channel used to open the file must allow write access to the file.

If the flag bit SEC$M_SYSGBL is set, the flag bit SEC$M_GBL must be set also.

Required Access or Privileges

If $CRMPSC specifies a global section and the SS$_NOPRIV condition value is returned, the process does not have the required privilege to create that section. To create global sections, the process must have the following privileges:

- SYSGBL privilege to create a system global section
- PRMGBL privilege to create a permanent global section
- PFNMAP privilege to create a page frame section

Note that you do not need PFNMAP privilege to map an existing page frame section.

Required Quota

If the section pages are copy-on-reference, the process must have sufficient paging file quota (PGFLQUOTA). The systemwide number of global page file pages is limited by the system parameter GBLPAGFIL.

Related Services

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

Condition Values Returned

SS$_NORMAL

The service completed successfully. The specified global section already exists and has been mapped.

SS$_CREATED

The service completed successfully. The specified global section did not previously exist and has been created.

SS$_ACCVIO

The *inadr* argument, *gsdnam* argument, or name descriptor cannot be read by the caller; the *inadr* argument was omitted; or the *retadr* argument cannot be written by the caller.

SS$_ENDOFFILE

The starting virtual block number specified is beyond the logical end-of-file, or the value in the *relpag* argument is greater than or equal to the actual size of the global section.

SS$_EXBYTLM

The process has exceeded the byte count quota; the system was unable to map the requested file.

SS$_EXGBLPAGFIL

The process has exceeded the systemwide limit on global page file pages; no part of the section was mapped.

SS$_EXQUOTA

The process exceeded its paging file quota while creating copy-on-reference or page file backing store pages.

SS$_GPTFULL

There is no more room in the system global page table to set up page table entries for the section.

SS$_GSDFULL

There is no more room in the system space allocated to maintain control information for global sections.

SS$_ILLPAGCNT

The page count value is negative or is 0 for a physical page frame section.

SS$_INSFMEM

Not enough pages are available in the specified shared memory to create the section.

SS$_INSFWSL

The process's working set limit is not large enough to accommodate the increased size of the address space.

SS$_IVCHAN

An invalid channel number was specified, that is, a channel number of 0 or a number larger than the number of channels available.

SS$_IVCHNLSEC

The channel number specified is currently active.

SS$_IVLOGNAM

The specified global section name has a length of 0 or has more than 43 characters.

SS$_IVLVEC

The specified section was not installed using the /PROTECT qualifier.

SS$_IVSECFLG

An invalid flag, a reserved flag, a flag requiring a privilege you lack, or an invalid combination of flags was specified.

SS$_IVSECIDCTL

The match control field of the global section identification is invalid.

SS$_NOPRIV

The process does not have the privileges to create a system global section (SYSGBL) or a permanent group global section (PRMGBL).

The process does not have the privilege to create a section starting at a specific physical page frame number (PFNMAP).

The process does not have the privilege to create a global section in memory shared by multiple processors (SHMEM).

A page in the input address range is in the system address space.

The specified channel is not assigned or was assigned from a more privileged access mode.

SS$_NOSHPTS

A virtual address within a shared page table region was specified.

SS$_NOTFILEDEV

The device is not a file-oriented, random-access, or directory device.

SS$_NOWRT

The section cannot be written to because the flag bit SEC$M_WRT is set, the file is read only, and the flag bit SEC$M_CRF is not set.

SS$_PAGOWNVIO

A page in the specified input address range is owned by a more privileged access mode.

SS$_SECREFOVF

The maximum number of references for a global section has been reached (2,147,483,647).

SS$_SECTBLFUL

There are no entries available in the system global section table or in the process section table.

SS$_TOOMANYLNAM

The logical name translation of the *gsdnam* argument exceeded the allowed depth.

SS$_VA_IN_USE

A page in the specified input address range is already mapped, and the flag SEC$M_NO_OVERMAP is set, or the existing underlying page cannot be deleted because it is associated with a buffer object.

SS$_VASFULL

The process's virtual address space is full; no space is available in the page tables for the pages created to contain the mapped global section.

$CRMPSC_FILE_64 (Alpha and Integrity servers)

Create and Map Private Disk File Section — On Alpha and Integrity server systems, allows a process to map a section of its address space to a specified portion of a file. This service creates and maps a private disk file section. This service accepts 64-bit addresses.

Format

```
SYS$CRMPSC_FILE_64

 region_id_64 ,file_offset_64 ,length_64 ,chan ,acmode ,flags ,return_va_64
   ,return_length_64 [,fault_cluster [,start_va_64]]
```

C Prototype

```
int sys$crmpsc_file_64
   (struct _generic_64 *region_id_64, unsigned __int64 file_offset_64,
    unsigned __int64 length_64, unsigned short int chan,
    unsigned int acmode, unsigned int flags, void *(*(return_va_64)),
    unsigned __int64 *return_length_64,...)
```

Arguments

region_id_64

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

The region ID associated with the region to map the private disk file 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.

file_offset_64

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

Byte offset into the file that marks the beginning of the section. The *file_offset_64* argument is a quadword containing this number. If you specify the *file_offset_64* argument as 0, the section is created beginning with the first byte in the file.

The *file_offset_64* argument must be a multiple of virtual disk blocks.

length_64

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

Length, in bytes, of the private disk file section to be created and mapped to. The length specified must be 0 or a multiple of virtual disk blocks. If the length specified is 0 or extends beyond end-of-file (EOF), the disk file is mapped up to and including the virtual block number that contains EOF.

chan

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

Number of the channel on which the file has been accessed. The *chan* argument is a longword containing this number. The access mode at which the channel was opened must be equal to or less privileged than the access mode of the caller.

Use the OpenVMS Record Management Services (RMS) macro $OPEN to access a file; the file options parameter in the file access block must indicate a user file open (UFO) keyword.

acmode

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

Access mode that is to be the owner of the pages created during the mapping. The *acmode* argument is a longword containing the access mode.

The $PSLDEF macro in STARLET.MLB and the file PSLDEF.H in SYS$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. The calling process can delete pages only if those pages are owned by an access mode equal to or less privileged than the access mode of the calling process.

flags

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

Flag mask specifying the characteristics of the private section to be created. 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 $CRMPSC_FILE_64 service:

Flag	Description
SEC$M_CRF	Pages are copy-on-reference.
SEC$M_DZRO	Pages are demand-zero pages. By default, they are not zeroed when copied. Note that SEC$M_DZRO and SEC$M_CRF cannot both be set and that SEC$M_DZRO set and SEC$M_WRT clear is an invalid combination.
SEC$M_EXPREG	Pages are mapped into the first available space at the current end of the specified region.

Flag	Description
SEC$M_NO_OVERMAP	Pages cannot overmap existing address space. By default, pages can overmap existing address space.
SEC$M_WRT	Pages form a read/write section. By default, pages form a read-only section.

All other bits in the *flags* argument are reserved to OpenVMS for future use 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- or 64-bit reference

The lowest process virtual address into which the private disk file section was mapped. The *return_va_64* argument is the 32- 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- or 64-bit reference.

The 32- or 64-bit virtual address of a naturally aligned quadword into which the service returns the length of the usable virtual address range mapped in bytes. This length might differ from the total amount mapped. If the section being mapped does not completely fill the last page used to map the section, the *return_va_64* and *return_length_64* arguments indicate the highest address that actually maps the section.

fault_cluster

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

Page fault cluster in byte units indicating how many pages are to be brought into memory when a page fault occurs for a single page. The fault cluster specified will be rounded up to a multiple of CPU-specific pages.

If this argument is specified as 0, the process default page fault cluster will be used. If this argument is specified as more than the maximum allowed for the system, no condition value will be returned. The systemwide maximum will be used.

start_va_64

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

The starting virtual address to map the private disk file 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.

Description

The Create and Map Private Disk File Section service allows a process to create a map to a private disk file section. Creating a private disk file section involves mapping all or part of a disk file as a section. The section is mapped from a low address to a high address whether the section is mapped in a region that grows from low to high addresses or from high to low addresses.

The flag SEC$M_WRT applies only to the way in which the newly created section is mapped. For a file to be made writable, the channel used to open the file must allow write access to the file.

If the 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

None

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 process must have sufficient byte count quota to satisfy the request.

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

Related Services

$CREATE_REGION_64, $CRMPSC, $CRMPSC_GFILE_64, $CRMPSC_GPFILE_64, $CRMPSC_GPFN_64, $CRMPSC_PFN_64, $DELETE_REGION_64, $DELTVA_64, $LCKPAG_64, $LKWSET_64, $PURGE_WS, $SETPRT_64, $ULKPAG_64, $ULWSET_64, $UPDSEC_64, $UPDSEC_64W

Condition Values Returned

SS$_NORMAL

The service completed successfully.

SS$_ACCVIO

The *return_va_64* argument or the *return_length_64* argument cannot be written by the caller.

SS$_CHANVIO

The specified channel was assigned from a more privileged access mode.

SS$_ENDOFFILE

The *file_offset_64* argument specified is beyond the logical end-of-file.

SS$_EXBYTLM

The process has exceeded the byte count quota; the system was unable to map the requested file.

SS$_EXPGFLQUOTA

The process exceeded its paging file quota.

SS$_INSFWSL

The process's working set limit is not large enough to accommodate the increased virtual address space.

SS$_IVCHAN

An invalid channel number was specified; the channel number specified was 0 or a channel that is unassigned.

SS$_IVCHNLSEC

The channel number specified is currently active, or there are no files opened on the specified channel.

SS$_IVIDENT

An invalid channel number was specified; the channel number specified is larger than the number of channels available.

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 and arguments was specified.

SS$_LEN_NOTBLKMULT

The *length_64* argument is not a multiple of virtual disk blocks.

SS$_NOSHPTS

A virtual address within a shared page table region was specified.

SS$_NOTFILEDEV

The device is not a file-oriented, random-access, or directory device.

SS$_OFF_NOTBLKALGN

The *file_offset_64* argument is not a multiple of virtual disk blocks.

SS$_NOWRT

The file is read-only, the flag bit SEC$M_WRT was set, and the flag bit SEC$M_CRF is not set.

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 section.

SS$_VA_IN_USE

A page in the specified input address range is already mapped and the flag SEC$M_NO_OVERMAP is set, or the existing underlying page cannot be deleted because it is associated with a buffer object.

SS$_VA_NOTPAGALGN

The *start_va_64* argument is not CPU-specific page aligned.

$CRMPSC_GDZRO_64 (Alpha and Integrity servers)

Create and Map to Global Demand-Zero Section — On Alpha and Integrity server systems, allows a process to create a memory-resident global demand-zero section and to map a section of its address

space to the global section. Shared page table sections can also be created. This service accepts 64-bit addresses.

Format

```
SYS$CRMPSC_GDZRO_64
   gs_name_64 ,ident_64 ,prot ,length_64 ,region_id_64 ,section_offset_64
   ,acmode ,flags ,return_va_64 ,return_length_64
   [[[[,start_va_64] ,map_length_64] ,reserved_length_64] ,rad_mask]
```

C Prototype

```
int sys$crmpsc_gdzro_64
   (void *gs_nam_64, struct _secid *ident_64, unsigned int prot,
    unsigned __int64 length_64, struct _generic_64 *region_id_64,
    unsigned __int64 section_offset_64, unsigned int acmode,
    unsigned int flags, void *(*(return_va_64)),
    unsigned __int64 *return_length_64,...);
```

Arguments

gs_name_64

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

Name of the global section. The *gs_name_64* argument is the 32- or 64-bit virtual address of a naturally aligned 32- 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- 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- 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 2 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.

Value	Symbolic Name	Match Criteria
2	SEC$K_MATLEQ	Match if the major identifications are equal and the minor identification of the mapper is less than or equal to the minor identification of the global section.

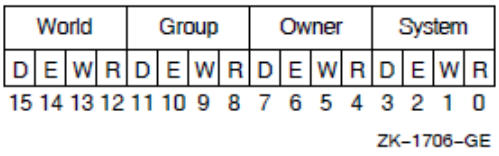
When a section is mapped at creation time, the match control field is ignored. If you specify the *ident_64* argument as 0, the version number and match control fields default to 0.

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.

prot

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

Protection to be applied to the global demand-zero section. The mask contains four 4-bit fields. Bits are read from right to left in each field. The following diagram depicts the mask:



Cleared bits indicate that read, write, execute, and delete access, in that order, are granted to the particular category of user. Only read, write, and execute access are meaningful for section protection. Delete access bits are ignored. Read access also grants execute access for those situations where execute access applies. If zero is specified, read access and write access are granted to all users.

length_64

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

Length, in bytes, of the global demand-zero section to be created. The *length_64* must be specified as a multiple of the CPU-specific page size. A length of 0 cannot be specified.

Note

Creating a memory-resident global section with shared page table does not imply that the global section must have an even multiple of CPU-specific page table pages. The global section might not fully use the last page table page.

region_id_64

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

The region ID associated with the region to map the global page file 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 to start mapping into the process's virtual address space. The offset specified must be a multiple of a CPU-specific page size.

If a shared page table region is specified by the *region_id_64* argument, *section_offset_64* must be an even multiple of the larger of the number of bytes that can be mapped by a CPU-specific page. For Integrity server systems, the alignment of section offsets must also be an integer multiple of the page size used to map VA space at this offset.

VSI recommends that you avoid any partial mapping of memory-resident sections when you use shared page tables on Integrity server systems. If you cannot avoid this, set bit 4 in the system parameter MMG_CTLFLAGS to limit the effective page size to the number of bytes that can be mapped by a page.

acmode

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

Access mode that is to be the owner of the pages created during the mapping. This access mode is also the read access mode and the write access mode. The *acmode* argument is a longword containing the access mode.

If the memory-resident global section is created with shared page tables, this is the access mode that is stored in the owner, read, and write fields of the corresponding shared page table entries (PTEs).

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. The calling process can delete pages only if those pages are owned by an access mode equal to or less privileged than the access mode of the calling process.

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 the type of the global section to be created as well as its characteristics. 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 $CRMPSC_GDZRO_64 service:

Flag	Description
SEC$M_DZRO	Pages are demand-zero pages. By default, this flag is always present in this service and cannot be disabled.
SEC$M_EXPREG	Pages are mapped into the first available space at the current end of the specified region. If the /ALLOCATE qualifier was specified when the global section was registered in the Reserved Memory Registry, virtually aligned addresses after the first available space are chosen for the mapping.
SEC$M_GBL	Pages form a global section. By default, this flag is always present in this service and cannot be disabled.

Flag	Description
SEC$M_NO_OVERMAP	Pages cannot overmap existing address space.
SEC$M_PERM	Global section is permanent.
SEC$M_RAD_HINT	When set, the argument *rad_mask* is used as a mask of RADs from which to allocate memory. See the *rad_mask* argument description for more information.
SEC$M_READ_ONLY_SHPT	Create shared table pages for the section that allow read access only.
SEC$M_SHMGS	Create a shared-memory global section.
SEC$M_SYSGBL	Pages form a system global section. By default, pages form a group global section.
SEC$M_MRES	Pages form a memory-resident section. By default, this page is always present in this service and cannot be disabled.
SEC$M_WRT	Pages form a read/write section. By default, this flag is always present in this service and cannot be disabled.

All other bits in the *flags* argument are reserved to OpenVMS for future use and should be specified as 0. The condition value SS$_IVSECFLG is returned if any undefined bits are set or if an invalid combination of flags is set.

return_va_64

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

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

If a shared page table region is specified by the *region_id_64* argument and the SEC$M_EXPREG flag is set, the returned virtual address is aligned to a CPU-specific page table page boundary.

return_length_64

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

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

start_va_64

OpenVMS usage:	address

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

The starting virtual address to map the memory-resident 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 nonzero, the condition value SS$_IVSECFLG is returned.

If SEC$M_EXPREG is clear, *start_va_64* is nonzero, and a shared page table region is specified, the specified starting address must be aligned to a natural page table page boundary; otherwise, the condition value SS$_VA_NOTPAGALGN is returned.

If the /ALLOCATE qualifier was specified when the memory-resident global section was registered in the Reserved Memory Registry and *start_va_64* is aligned to a multiple of CPU-specific pages appropriate for taking advantage of granularity hints then granularity hints are used to map to the global section:

- On Alpha systems, granularity hints mean multiples of pages, regardless of page size. The multiples 8, 64, and 512 pages are architected.
- On Integrity server systems, OpenVMS initially supports page sizes of 64KB, 256KB, and 4MB instead of granularity hints. Additional pages sizes will be supported in the future.

If the flag VA$M_SHARED_PTS is set and this argument is specified, the specified starting address must be aligned to the larger of a natural page table boundary or to the largest possible page size used to map the section. If the alignment is less than a page table boundary, the $CREATE_REGION_64 service returns an error. If the alignment is less than the largest page size used in the section, an error might be returned when you attempt to map the section.

If you do not specify a starting address, OpenVMS automatically ensures correct alignment.

map_length_64

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

Length of the memory-resident global section to be mapped. The length specified must be a multiple of CPU-specific pages. If this argument is not specified or is specified as zero, the global file section is mapped up to and including the last page in that section.

If a shared page table region is specified by the *region_id_64* argument, *map_length_64* must be an even multiple of the number of bytes that can be mapped by a CPU-specific page table page or must include the last page within the global section.

reserved_length_64

OpenVMS usage:	byte count
type:	quadword (unsigned)

access:	write only
mechanism:	32- or 64-bit reference

Length, in bytes, of the global section as currently registered in the Reserved Memory Registry. The *reserved_length_64* argument is the 32- or 64-bit virtual address of a naturally aligned quadword into which the service returns the reserved length.

If *reserved_length_64* is not specified or is specified as 0, no reserved length is returned to the caller.

If the memory-resident global section is not registered, *reserved_length_64* is written with the value 0.

rad_mask

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

Use the *rad_mask* argument to specify from which RADs to allocate memory. Currently only one bit may be set. The specified RAD must contain memory. This argument is only a hint. Memory may be obtained from other RADs if no free memory is available at the time of allocation.

The *rad_mask* argument is considered only if the SEC$M_RAD_HINT flag is specified. Otherwise, this argument is ignored.

On a system that does not support resource affinity domains (RADs), specifying 1 for the *rad_mask* argument is allowed.

RAD is supported on AlphaServer GS series systems and starting from OpenVMS Version 8.4, support is extended to NUMA capable Integrity servers.

Description

The Create and Map to Global Demand-Zero Section service allows a process to create and map to a memory-resident global demand-zero section. If you set the SEC$M_SHMGS flag, the section is created as a Galaxy-wide global demand-zero section in shared memory.

You must call either the $CREATE_GDZRO service or the $CRMPSC_GDZRO_64 service on each instance where the Galaxy shared memory will be accessed.

Memory-resident or Galaxy-wide global sections contain demand-zero allocation pages that are writable and memory resident. All pages in these types of global section are shared by all processes that map to the global section.

If the $CRMPSC_GDZRO_64 service specifies a global section that already exists, the service maps to it only if it is a memory-resident global section. All pages in the memory-resident global section are shared by all processes that map to the global section.

The global demand-zero pages are always resident in memory and are not backed up by any file on any disk. The global pages are not charged against any page file quota. The process must have the rights

identifier VMS$MEM_RESIDENT_USER to create a memory-resident global section; otherwise, the error status SS$_NOMEMRESID is returned.

The pages are always resident in memory and are not backed up by any file on any disk. The pages are not placed into the process's working set list when the process maps to the global section and the virtual memory is referenced by the process. The pages are also not charged against the process's working set quota or against any page-file quota.

Only memory-resident sections can be registered with the Reserved Memory Registry in the SYSMAN facility. Memory for Galaxy-wide shared sections is reserved through appropriate settings of the console environment parameters.

If the memory-resident global section is either not registered in the Reserved Memory Registry or if the /NOALLOCATE qualifier was specified when the global section was registered in the Reserved Memory Registry, invalid global PTEs are written to the global page table and invalid PTEs are placed in the process page table. Physical memory is not allocated until the virtual memory is referenced.

If the global section is registered in the Reserved Memory Registry, the size of the global section need not match the reserved size. If the global section is not registered in the Reserved Memory Registry or if the reserved size is smaller than the size of the global section, the error status SS$_INSFLPGS is returned if there are not enough fluid pages in the system to satisfy the request.

If the /ALLOCATE qualifier was specified when the global section was registered in the Reserved Memory Registry, contiguous, aligned physical pages are preallocated during system initialization for this global section. Valid page table entries are placed in the global page table and in the process page table. If the reserved preallocated memory is smaller than the size of the global section, the error SS$_MRES_PFNSMALL is returned and the global section is not created.

If the memory-resident global section is not registered in the Reserved Memory Registry or if the /PAGE_TABLES qualifier was specified when the global section was registered in the Reserved Memory Registry, shared page tables are created for the global section.

For more information about using the SYSMAN utility to create entries to the Reserved Memory Registry, refer to the *VSI OpenVMS System Management Utilities Reference Manual*.

Shared page tables consume the same internal OpenVMS data structures as a global section. The system parameters GBLPAGES and GBLSECTIONS must account for the additional global pages and the additional global section.

To use the shared page tables associated with a memory-resident global section, you must first create a shared page-table region (with $CREATE_REGION_64). To map to the memory-resident global section using the shared page tables, you must do the following:

- Specify a shared page-table region in the *region_id_64* argument.
- Set the flag SEC$M_EXPREG or provide a CPU-specific page table page aligned virtual address in the *start_va_64* argument.
- Specify a value for the *section_offset_64* argument that is an even multiple of bytes mapped by a CPU-specific page table page or zero.
- Specify a value for the *map_length_64* argument that is an even multiple of bytes mapped by a CPU-specific page table page or zero, or include the last page of the section.

See the description of the $CREATE_REGION_64 service for information about calculating virtual addresses that are aligned to a CPU-specific page table page boundary.

The memory-resident global section can be mapped with shared page tables or private page tables. The following table lists the factors associated with determining whether the mapping occurs with shared page tables or with private page tables:

Global Section Created with Shared Page Tables	Shared Page Table Region Specified by *region_id_64*	Type of Page Tables Used in Mapping
No	No	Private
No	Yes	Private
Yes	No	Private
Yes	Yes	Shared

In general, if the flag SEC$M_EXPREG is set, the first free virtual address within the specified region is used to map to the global section.

If the flag SEC$M_EXPREG is set and the *region_id_64* argument indicates a shared page table region, the first free virtual address within the specified region is rounded up to a CPU-specific page table page boundary and used to map to the global section.

If the flag SEC$M_EXPREG is set and if the /ALLOCATE qualifier was specified with the SYSMAN command RESERVED_MEMORY ADD for the memory-resident global section, the first free virtual address within the specified region is rounded up to the same virtual alignment as the physical alignment of the preallocated pages and used to map to the global section. Granularity hints are set appropriately for each process private PTE.

In general, if the flag SEC$M_EXPREG is clear, the virtual address in the *start_va_64* argument is used to map to the global section.

If the flag SEC$M_EXPREG is clear, the value specified in the *start_va_64* argument can determine if the mapping is possible and if granularity hints are used in the private page tables. If a shared page table region is specified by the *region_id_64* argument, the virtual address specified by the *start_va_64* argument must be on an even CPU-specific page table page boundary or an error is returned by this service. If the *region_id_64* argument does not specify a shared page table region and /ALLOCATE was specified with the SYSMAN command RESERVED_MEMORY ADD for this global section, granularity hints are used only if the virtual alignment of *start_va_64* is appropriate for the use of granularity hints (either 8-page, 64-page, or 512-page alignment).

Whenever granularity hints are being used within the mapping of a memory-resident global section, if the *length_64* argument is not an exact multiple of the alignment factor, lower granularity hints factors are used as appropriate at the higher addressed portion of the global section. If the *section_offset_64* argument is specified, a lower granularity hint factor can be used throughout the mapping of the global section to match the physical alignment of the first page mapped.

When you map a Galaxy shared section or a memory resident section that has an associated shared page table section, you have the following options for accessing data:

Table 22. Shared Page Tables

Shared Page Tables	Read Only	Read and Write
None created	Do not set the SEC$M_WRT flag in the map request.	Set the SEC$M_WRT flag in the map request.

Shared Page Tables	Read Only	Read and Write
	Private page tables will always be used, even if you are specifying a shared page table region into which to map the section.	Private page tables will always be used, even if you are specifying a shared page table region into which to map the section.
Write access	Do not set the SEC$M_WRT flag in the map request. Ensure that private page tables will be used. Do not specify a shared page table region into which to map the section. If you do, the error status SS$_IVSECFLG is returned.	Set the SEC$M_WRT flag in the map request. The shared page table section will be used for mapping if you specify a shared page table region into which to map the section.
Read access	Do not set the SEC$M_WRT flag in the map request. The shared page table section will be used for mapping if you specify a shared page table region into which to map the section.	Set the SEC$M_WRT flag in the map request. Ensure that private page tables will be used. Do not specify a shared page table region into which to map the section. If you do, the error status SS$_IVSECFLG is returned.

Note

Shared page tables for Galaxy shared sections are also implemented as Galaxy shared sections. This implies that they allow either read access only on all OpenVMS instances connected to this section or read and write access on all instances. The setting of the SEC$M_READ_ONLY_SHPT flag as requested by the first instance to create the section is used on all instances.

Using the $CRMPSC_GDZRO_64 service always implies that the SEC$M_WRT flag is set and that you want to map the section for writing. If you want to use this service to create a section with shared page tables for read only access, you must use private page tables and you cannot specify a shared page table region into which to map the section.

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 and, if specified as a nonzero value, the *reserved_length_64* argument.

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 contains the value --1.

If the service returns an error status value other than SS$_INSFLPGS or SS$_MRES_PFNSMALL, a value is not returned in the *reserved_length_64* argument.

If the service returns a successful condition value or if SS$_INSFLPGS or SS$_MRES_PFNSMALL is returned and the *reserved_length_64* argument is specified as a nonzero address, the length in bytes of the global section as registered in the Reserved Memory Registry is returned in the *reserved_length_64* argument.

Required Privileges

To create a global section, the process must have the following privileges:

- SYSGBL privilege to create a system global section (if flag SEC$M_SYSGBL is set)
- PRMGBL privilege to create a permanent global section
- VMS$MEM_RESIDENT_USER rights identifier to create a memory-resident section
- SHMEM privilege on OpenVMS Galaxy systems to create an object in Galaxy shared memory

Required Quota

If private page tables are used to map to the memory-resident global section, the working set limit quota (WSQUOTA) of the process must be sufficient to accommodate the increased size of the process page tables required by the increase in virtual address space.

If private page tables are used to map to the memory-resident global section, the page file quota (PGFLQUOTA) of the process must be sufficient to accommodate the increased size of the process page tables required by the increase in virtual address space.

Related Services

$CREATE_GDZRO, $CREATE_GPFILE, $CREATE_REGION_64, $CRMPSC, $CRMPSC_FILE_64, $CRMPSC_GFILE_64, $CRMPSC_GPFN_64, $CRMPSC_PFN_64, $DELETE_REGION_64, $DELTVA_64, $DGBLSC, $LCKPAG_64, $LKWSET_64, $MGBLSC_64, $PURGE_WS, $SETPRT_64, $ULKPAG_64, $ULWSET_64, $UPDSEC_64, $UPDSEC_64W

Condition Values Returned

SS$_NORMAL

The service completed successfully. The specified global section already exists and has been mapped.

SS$_CREATED

The service completed successfully. The global section has been created.

SS$_CREATED_SHPT

Global section has been created with shared page tables.

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$_BADRAD

The specified RAD contains no memory, or if the specified RAD is greater than or equal to the maximum number of RADs on the system.

SS$_EXPGFLQUOTA

The process's page file quota is not large enough to accommodate the increased virtual address space.

SS$_GBLSEC_MISMATCH

Global section type mismatch. The specified global section was found; however, it is not a memory-resident section.

SS$_GPTFULL

There is no more room in the system global page table to set up page table entries for the global section or for the shared page tables.

SS$_GSDFULL

There is no more room in the system space allocated to maintain control information for global sections.

SS$_INSFLPGS

Insufficient fluid pages available.

SS$_INSFRPGS

Insufficient free shared pages or private pages.

SS$_INSFWSL

The process's working set limit is not large enough to accommodate the increased virtual address space.

SS$_INSF_SHM_REG

The Galaxy shared memory code has run out of internal SHM_REG data structures. You need to increase the system parameter GLX_SHM_REG and reboot all Galaxy instances with this larger parameter value.

SS$_INV_SHMEM

Shared memory is not valid.

SS$_IVACMODE

The specified access mode is greater than PSL$_USER, or 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$_IVPROTECT

The protection argument format is invalid.

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_NOTPAGMULT

The *length_64* argument is not a multiple of CPU-specified pages. Or, if a shared page table region is specified by the *region_id_64* argument, the *map_length_64* argument is not a multiple of CPU-specific page table pages.

SS$_LOCK_TIMEOUT

An OpenVMS Galaxy lock timed out.

SS$_MRES_PFNSMALL

Preallocated, contiguous, aligned physical memory specified in the Reserved Memory Registry is smaller than the length specified for the memory-resident global section by the *length_64* argument.

SS$_NOBREAK

An OpenVMS Galaxy lock is held by another node and was not broken.

SS$_NOMEMRESID

The process attempted to create a memory-resident section, but was not holding the right identifier VMS$MEM_RESIDENT_USER.

SS$_NOPRMGBL

The process does not have the privileges to create or delete a permanent group global section (PRMGBL).

SS$_NOSYSGBL

The process does not have the privileges to create or delete a system global section (SYSGBL).

SS$_OFF_NOTPAGALGN

The *section_offset_64* argument is not CPU-specific page aligned. Or, if a shared page table region is specified by the *region_id_64* argument, the *section_offset_64* argument is not CPU-specific page table page aligned.

SS$_OFFSET_TOO_BIG

The *section_offset_64* argument specified is beyond the logical end-of-file.

SS$_PAGNOTINREG

A page in the specified input address range is not within the specified region.

SS$_PAGOWNVIO

A page in the specified input address range is owned by a more privileged access mode.

SS$_REGISFULL

The specified virtual region is full; no space is available in the region for the pages created to contain the mapped section.

SS$_SECREFOVF

The maximum number of references for a global section has been reached (2,147,483,647).

SS$_SECTBLFUL

There are no entries available in the system global section table for the global section or for the shared page tables.

SS$_TOOMANYLNAM

The logical name translation of the *gs_name_64* argument exceeded the allowed depth of 10.

SS$_VA_IN_USE

A page in the specified input address range is already mapped and the flag SEC$M_NO_OVERMAP is set, or a page in the specified input address range is in another region, in system space, or inaccessible; or, the existing underlying page cannot be deleted because it is associated with a buffer object.

SS$_VA_NOTPAGALGN

The *start_va_64* argument is not CPU-specific page aligned. Or, if a shared page table region is specified by the *region_id_64* argument, the *start_va_64* argument is not CPU-specific page table page aligned.

$CRMPSC_GFILE_64 (Alpha and Integrity servers)

Create and Map Global Disk File Section — On Alpha and Integrity server systems, allows a process to create a global disk file section and to map a section of its address space to the global section. This service accepts 64-bit addresses.

Format

```
SYS$CRMPSC_GFILE_64
   gs_name_64 ,ident_64 ,file_offset_64 ,length_64 ,chan ,region_id_64
   ,section_offset_64 ,acmode ,flags ,return_va_64 ,return_length_64
   [,fault_cluster [,start_va_64 [,map_length_64]]]
```

C Prototype

```
int sys$crmpsc_gfile_64
   (void *gs_nam_64, struct _secid *ident_64, unsigned __int64
    file_offset_64, unsigned __int64 length_64, unsigned short int chan,
    struct _generic_64 *region_id_64, unsigned __int64 section_offset_64,
    unsigned int acmode, unsigned int flags, void *(*(return_va_64)),
```

```
unsigned __int64 *return_length_64,...);
```

Arguments

gs_name_64

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

Name of the global section. The *gs_name_64* argument is the 32- or 64-bit virtual address of a naturally aligned 32- 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- 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- 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 2 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.

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

file_offset_64

OpenVMS usage:	byte offset
type:	quadword (unsigned)

access:	read only
mechanism:	by value

Byte offset into the file that marks the beginning of the section. The *file_offset_64* argument is a quadword containing this number. If you specify the *file_offset_64* argument as 0, the section is created beginning with the first byte in the file.

The file offset specified must be a multiple of virtual disk blocks.

length_64

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

Length, in bytes, of the global disk file section to be created. The length specified must be 0 or a multiple of virtual disk blocks. If the length specified is 0 or extends beyond the end-of-file (EOF), the global disk file section is created up to and including the virtual block number that contains EOF.

chan

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

Number of the channel on which the file has been accessed. The *chan* argument is a longword containing this number. The access mode at which the channel was opened must be equal to or less privileged than the access mode of the caller.

You can use the OpenVMS Record Management Services (RMS) macro $OPEN to access a file; the file options parameter in the file access block must indicate a user file open (UFO) keyword.

region_id_64

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

The region ID associated with the region in which to map the global disk file 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

Symbol	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 to start mapping into the process's virtual address space. The offset specified must be a multiple of virtual disk blocks.

acmode

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

Access mode that is to be the owner of the pages created during the mapping. The *acmode* argument is a longword containing the access mode.

The $PSLDEF macro in STARLET.MLB and the file PSLDEF.H in SYS$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 the characteristics of the global section to be created. 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 $CRMPSC_GFILE_64 service:

Flag	Description
SEC$M_CRF	Pages are copy-on-reference.
SEC$M_GBL	Pages form a global section. By default, this flag is always present in this service and cannot be disabled.
SEC$M_WRT	Pages form a read/write section. By default, pages form a read-only section.
SEC$M_DZRO	Pages are demand-zero pages. By default, they are not zeroed when copied. Note that SEC$M_DZRO and SEC$M_CRF cannot both be set and that SEC$M_DZRO set and SEC$M_WRT clear is an invalid combination.
SEC$M_EXPREG	Pages are mapped into the first available space at the current end of the specified region.
SEC$M_NO_OVERMAP	Pages cannot overmap existing address space. By default, pages can overmap existing address space.
SEC$M_PERM	Global section is permanent. By default, global sections are temporary.
SEC$M_SYSGBL	Pages form a system global section. By default, pages form a group global section.

All other bits in the *flags* argument are reserved to OPenVMS for future use 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- or 64-bit reference

The lowest process virtual address into which the global disk file section was mapped. The *return_va_64* argument is the 32- 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 *return_va_64* 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- or 64-bit reference

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

Upon successful completion of this service, the value in the *return_length_64* argument indicates the amount of created address space backed by the section file.

If the number of disk blocks mapped does not represent an exact multiple of CPU-specific pages, the last page in the mapped address space will not be completely mapped by the section file. In this case, modifying memory beyond the amount indicated by *return_length_64* can result in the loss of this data.

Unlike the *return_length_64* argument for the $CREATE_GFILE service, upon successful completion of this service, the *return_length_64* argument does not represent the total length of the global section created if the *section_offset_64* argument was specified as non-zero. The value in the *section_offset_64* argument plus the value in the *return_length_64* argument is the total length of the global disk file section created.

fault_cluster

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

Page fault cluster in byte units indicating how many pages are to be brought into memory when a page fault occurs for a single page. The fault cluster specified will be rounded up to a multiple of CPU-specific pages.

If this argument is specified as 0, the system default page fault cluster will be used. If this argument is specified as more than the maximum allowed for the system, no error will be returned. The systemwide maximum will be used.

start_va_64

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

The starting virtual address to map the global disk file section. The specified virtual address must be a CPU-specific page aligned address. If the flag SEC$M_EXPREG is specified, this argument will not be used. If SEC$M_EXPREG is clear and the *start_va_64* argument is not specified or is specified as 0, the condition value SS$_IVSECFLG will be returned.

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

map_length_64

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

Length of the global disk file section to be mapped. The length specified must be a multiple of virtual disk blocks. If this argument is not specified as zero, the global disk section is mapped up to and including the last disk block in the section.

Description

The Create and Map Global Disk File Section service allows a process to create and map to a global disk file section.

Creating a global disk file section involves defining all or part of a disk file as a section. The section is mapped from a low address to a high address whether the section is mapped in a region that grows from low to high addresses or from high to low addresses. If the $CRMPSC_GFILE_64 service specifies a global disk file section that already exists, the service maps it.

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.

The flag SEC$M_WRT applies only to the way in which the newly created section is mapped. For a file to be made writable, the channel used to open the file must allow write access to the file.

Required Privileges

To create a global section, the process must have the following privileges:

- SYSGBL privilege to create a system global section (if flag SEC$M_SYSGBL is set)
- PRMGBL privilege to create a permanent global section

Required Quota

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

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.

Related Services

$CREATE_REGION_64, $CRMPSC, $CRMPSC_FILE_64, $CRMPSC_GPFILE_64, $CRMPSC_GPFN_64, $CRMPSC_PFN_64, $DELETE_REGION_64, $DELTVA_64, $DGBLSC, $LCKPAG_64, $LKWSET_64, $MGBLSC_64, $PURGE_WS, $SETPRT_64, $ULKPAG_64, $ULWSET_64, $UPDSEC_64, $UPDSEC_64W

Condition Values Returned

SS$_NORMAL

The service completed successfully. The specified global section already exists and has been mapped.

SS$_CREATED

The service completed successfully. The specified global section did not previously exist and has been created.

SS$_ACCVIO

The *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$_CHANVIO

The specified channel was assigned from a more privileged access mode.

SS$_ENDOFFILE

The *file_offset_64* argument specified is beyond the logical end-of-file.

SS$_EXBYTLM

The process has exceeded the byte count quota; the system was unable to map the requested file.

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 was not a global disk file section.

SS$_GPTFULL

There is no more room in the system global page table to set up page table entries for the section.

SS$_GSDFULL

There is no more room in the system space allocated to maintain control information for global sections.

SS$_INSFWSL

The process's working set limit 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$_IVCHAN

An invalid channel number was specified; the channel number specified was 0 or a channel that is unassigned.

SS$_IVCHNLSEC

The channel number specified is currently active or there are no files opened on the specified channel.

SS$_IVIDENT

An invalid channel number was specified; the channel number specified is larger than the number of channels available.

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 and arguments was specified.

SS$_IVSECIDCTL

The match control field of the global section identification is invalid.

SS$_LEN_NOTBLKMULT

The *length_64* or the *map_length_64* argument is not a multiple of virtual disk blocks.

SS$_NOPRMGBL

The process does not have the privileges to create or delete a permanent group global section (PRMGBL).

SS$_NOSHPTS

The region ID of a shared page table region was specified.

SS$_NOSYSGBL

The process does not have the privileges to create or delete a system global section (SYSGBL).

SS$_NOTFILEDEV

The device is not a file-oriented, random-access, or directory device.

SS$_NOWRT

The file is read-only, and the flag bit SEC$M_CRF is not set.

SS$_OFF_NOTBLKALGN

The *file_offset_64* or *section_offset_64* argument is not virtual disk block aligned.

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$_PAGOWNVIO

A page in the specified input address range is owned by a more privileged access mode.

SS$_REGISFULL

The specified virtual region is full; no space is available in the region for the pages created to contain the mapped section.

SS$_SECREFOVF

The maximum number of references for a global section has been reached (2,147,483,647).

SS$_SECTBLFUL

There are no entries available in the system global section table.

SS$_TOOMANYLNAM

The logical name translation of the *gs_name_64* argument exceeded the allowed depth of 10.

SS$_VA_IN_USE

A page in the specified input address range is already mapped, and the flag SEC$M_NO_OVERMAP is set.

SS$_VA_NOTPAGALGN

The *start_va_64* argument is not CPU-specific page aligned.

$CRMPSC_GPFILE_64 (Alpha and Integrity servers)

Create and Map Global Page File Section — On Alpha and Integrity server systems, allows a process to create a global page file section and to map a section of its address space to the global section. This service accepts 64-bit addresses.

Format

```
SYS$CRMPSC_GPFILE_64
   gs_name_64 ,ident_64 ,prot ,length_64 ,region_id_64 ,section_offset_64
```

```
   ,acmode ,flags ,return_va_64 ,return_length_64
   [,start_va_64 [,map_length_64]]
```

C Prototype

```
int sys$crmpsc_gpfile_64
   (void *gs_nam_64, struct _secid *ident_64, unsigned int prot,
    unsigned __int64 length_64, struct _generic_64 *region_id_64,
    unsigned __int64 section_offset_64, unsigned int acmode,
    unsigned int flags, void *(*(return_va_64)),
    unsigned __int64 *return_length_64,...);
```

Arguments

gs_name_64

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

Name of the global section. The *gs_name_64* argument is the 32- or 64-bit virtual address of a naturally aligned 32- 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- 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- 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 2 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.

When a section is mapped at creation time, the match control field is ignored. If you specify the *ident_64* argument as 0, the version number and match control fields default to 0.

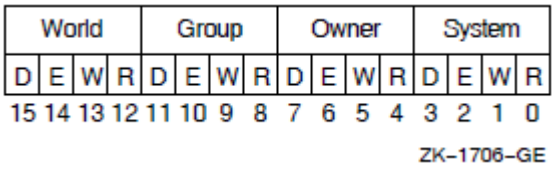
The version number is in the second longword. The version number contains two fields: a minor identification in the low-order 24 bits and a major identification in the high-order 8 bits. You can assign

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.

prot

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

Protection to be applied to the global page file section. The mask contains four 4-bit fields. Bits are read from right to left in each field. The following diagram depicts the mask:



Cleared bits indicate that read, write, execute, and delete access, in that order, are granted to the particular category of user. Only read, write, and execute access are meaningful for section protection. Delete access bits are ignored. Read access also grants execute access for those situations where execute access applies. If zero is specified, read access and write access are granted to all users.

length_64

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

Length, in bytes, of the global page file section to be created. The length specified must be a multiple of CPU-specific pages. A length of 0 cannot be specified.

region_id_64

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

The region ID associated with the region to map the global page file 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

Symbol	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 to start mapping into the process's virtual address space. The offset specified must be a multiple of virtual disk blocks.

acmode

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

Access mode that is to be the owner of the pages created during the mapping. The *acmode* argument is a longword containing the access mode.

The $PSLDEF macro in STARLET.MLB and the file PSLDEF.H in SYS$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. The calling process can delete pages only if those pages are owned by an access mode equal to or less privileged than the access mode of the calling process.

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 the characteristics of the global section to be created. 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 $CRMPSC_GPFILE_64 service:

Flag	Description
SEC$M_DZRO	Pages are demand-zero pages. By default, this flag is always present in this service and cannot be disabled.
SEC$M_EXPREG	Pages are mapped into the first available space at the current end of the specified region. SEC$M_EXPREG cannot be specified with the SEC$M_NO_OVERMAP flag.
SEC$M_GBL	Pages form a global section. By default, this flag is always present in this service and cannot be disabled.
SEC$M_NO_OVERMAP	Pages cannot overmap existing address space. By default, pages can overmap existing address space. SEC$M_NO_OVERMAP cannot be specified with the SEC$M_EXPREG flag.
SEC$M_PAGFIL	Pages form a global page file section. By default, this flag is always present in this service and cannot be disabled.
SEC$M_PERM	Global section is permanent. By default, global sections are temporary.
SEC$M_SYSGBL	Pages form a system global section. By default, pages form a group global section.
SEC$M_WRT	Pages form a read/write section. By default, this flag is always present in this service and cannot be disabled.

All other bits in the *flags* argument are reserved to OpenVMS for future use and should be specified as 0. The condition value SS$_IVSECFLG is returned if any undefined bits are set or if an invalid combination of flags is set.

return_va_64

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

The lowest process virtual address into which the global page file section was mapped. The *return_va_64* argument is the 32- 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- or 64-bit reference

The 32- or 64-bit virtual address of a naturally aligned quadword into which the service returns the length of the virtual address range mapped 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 page file 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 range of virtual addresses mapped.

map_length_64

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

Length of the global page file section to be mapped. The length specified must be a multiple of CPU-specific pages. If this argument is not specified or is specified as zero, the global file section is mapped up to and including the last page in that section.

Description

The Create and Map Global Page File Section service allows a process to create a map to global page file section. Creating a global page file section involves defining a global section backed up by the system page file. The section is mapped from a low address to a high address whether the section is mapped in a region that grows from low to high addresses or from high to low addresses. If the $CRMPSC_GPFILE_64 service specifies a global section that already exists, the service maps it.

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

In order to create a global section, the process must have the following privileges:

- SYSGBL privilege to create a system global section (if flag SEC$M_SYSGBL is set)
- PRMGBL privilege to create a permanent global section

Required Quota

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

The working set limit quota (WSQUOTA) of the process must be sufficient to accommodate the increased size of the process page table required by the increase in virtual address space when the section is mapped.

Related Services

$CREATE_GPFILE, $CREATE_REGION_64, $CRMPSC, $CRMPSC_FILE_64, $CRMPSC_GFILE_64, $CRMPSC_GPFN_64, $CRMPSC_PFN_64, $DELETE_REGION_64, $DELTVA_64, $DGBLSC, $LCKPAG_64, $LKWSET_64, $MGBLSC_64, $PURGE_WS, $SETPRT_64, $ULKPAG_64, $ULWSET_64, $UPDSEC_64, $UPDSEC_64W

Condition Values Returned

SS$_NORMAL

The service completed successfully. The specified global section already exists and has been mapped.

SS$_CREATED

The service completed successfully. The specified global section did not previously exist and has been created.

SS$_ACCVIO

The *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$_EXBYTLM

The process has exceeded the byte count quota.

SS$_EXGBLPAGFIL

The process has exceeded the systemwide limit on global page file pages; no part of the section was mapped.

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$_GPTFULL

There is no more room in the system global page table to set up page table entries for the section.

SS$_GSDFULL

There is no more room in the system space allocated to maintain control information for global sections.

SS$_INSFWSL

The process's working set limit 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 and arguments was specified.

SS$_IVSECIDCTL

The match control field of the global section identification is invalid.

SS$_LEN_NOTPAGMULT

The *length_64* argument is not a multiple of CPU-specified pages or was specified as 0.

SS$_NOPRMGBL

The process does not have the privileges to create or delete a permanent group global section (PRMGBL).

SS$_NOSHPTS

The region ID of a shared page table region was specified.

SS$_NOSYSGBL

The process does not have the privileges to create or delete a system global section (SYSGBL).

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$_PAGOWNVIO

A page in the specified input address range is owned by a more privileged access mode.

SS$_REGISFULL

The specified virtual region is full; no space is available in the region for the pages created to contain the mapped section.

SS$_SECREFOVF

The maximum number of references for a global section has been reached (2,147,483,647).

SS$_SECTBLFUL

There are no entries available in the system global section table.

SS$_TOOMANYLNAM

The logical name translation of the *gs_name_64* argument exceeded the allowed depth of 10.

SS$_VA_IN_USE

A page in the specified input address range is already mapped and the flag SEC$M_NO_OVERMAP is set, or the existing underlying page cannot be deleted because it is associated with a buffer object.

SS$_VA_NOTPAGALGN

The *start_va_64* argument is not CPU-specific page aligned.

$CRMPSC_GPFN_64 (Alpha and Integrity servers)

Create and Map Global Page Frame Section — On Alpha and Integrity server systems, allows a process to create a permanent global page frame section and to map a section of its address space to the global page frame section. This service accepts 64-bit addresses.

Format

```
SYS$CRMPSC_GPFN_64
   gs_name_64 ,ident_64 ,prot ,start_pfn ,page_count ,region_id_64
   ,relative_page ,acmode ,flags ,return_va_64 ,return_length_64
   [,start_va_64 [,map_page_count]]
```

C Prototype

```
int sys$crmpsc_gpfn_64
   (void *gs_nam_64, struct _secid *ident_64, unsigned int prot,
    unsigned int start_pfn, unsigned int page_count,
    struct _generic_64 *region_id_64, unsigned int relative_page,
    unsigned int acmode, unsigned int flags, void *(*(return_va_64)),
    unsigned __int64 *return_length_64,...);
```

Arguments

gs_name_64

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

Name of the global section. The *gs_name_64* argument is the 32- or 64-bit virtual address of a naturally aligned 32- 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- 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- 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 2 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.

When a section is mapped at creation time, the match control field is ignored. If you specify the *ident_64* argument as 0, the version number and match control fields default to 0.

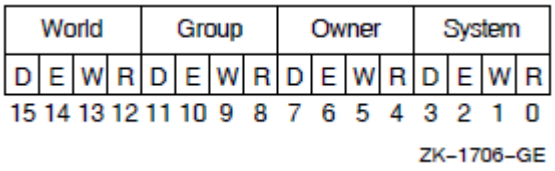
The version number is in the second longword. The version number contains two fields: a minor identification in the low-order 24 bits and a major identification in the high-order 8 bits. You can assign values for these fields by installation convention to differentiate versions of global sections. If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

prot

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

Protection to be applied to the global page file section.

The mask contains four 4-bit fields. Bits are read from right to left in each field. The following diagram depicts the mask:



Cleared bits indicate that read, write, execute, and delete access, in that order, are granted to the particular category of user. Only read, write, and execute access are meaningful for section protection. Delete access bits are ignored. Read access also grants execute access for those situations where execute access applies. If zero is specified, read access and write access are granted to all users.

start_pfn

OpenVMS usage:	page frame number
type:	longword (unsigned) on Alpha, quadword (unsigned) on Integrity servers
access:	read only
mechanism:	by value

The CPU-specific page frame number where the section begins.

page_count

OpenVMS usage:	CPU-specific page count
type:	longword (unsigned) on Alpha, quadword (unsigned) on Integrity servers
access:	read only
mechanism:	by value

Length of the page frame section in CPU-specific pages.

region_id_64

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

The region ID associated with the region to map the global 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 number
type:	longword (unsigned)
access:	read only
mechanism:	by value

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

acmode

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

Access mode that is to be the owner of the pages created during the mapping. The *acmode* argument is a longword containing the access mode.

The $PSLDEF macro in STARLET.MLB and the file PSLDEF.H in SYS$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.

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 the characteristics of the global section to be created. 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 $CRMPSC_GPFN_64 service:

Flag	Description
SEC$M_ARGS64	Indicates that all parameters, specifically *start_pfn* and *page_count*, are passed as 64-bit numbers. This flag is ignored on OpenVMS Alpha but must be set on Integrity server systems. If the flag is not set on Integrity servers, the error code SS$_IVSECFLG is returned.
SEC$M_EXPREG	Pages are mapped into the first available space at the current end of the specified region.
SEC$M_GBL	Pages form a global section. By default, this flag is always present in this service and cannot be disabled.
SEC$M_PERM	Global section is permanent. By default, this flag is always present in this service and cannot be disabled.
SEC$M_PFNMAP	Pages form a page frame section. By default, this flag is always present in this service and cannot be disabled.
SEC$M_NO_OVERMAP	Pages cannot overmap existing address space. By default, pages can overmap existing address space.
SEC$M_SYSGBL	Pages form a system global section. By default, pages form a group global section.
SEC$M_WRT	Pages form a read/write section. By default, pages form a read-only section.

All other bits in the *flags* argument are reserved to OpenVMS for future use 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- 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- or 64-bit 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- or 64-bit reference

The 32- or 64-bit virtual address of a naturally aligned quadword into which the service returns the length of the virtual address range mapped 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 page frame 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 range of virtual addresses mapped.

map_page_count

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

Length of the global page frame section to be mapped in CPU-specific pages.

Description

The Create and Map Global Page Frame Section service allows a process to create and map to a global page frame section. Creating a global page frame section involves defining certain physical page frame numbers (PFNs) as a section.

All global page frame sections are permanent. 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

To create a global page frame section, the process must have the following privileges:

- SYSGBL privilege to create a system global section (if flag SEC$M_SYSGBL is set)
- PRMGBL privilege to create a permanent global section (if flag SEC$M_PERM is set)
- PFNMAP privilege to create a page frame section

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.

Related Services

$CREATE_GPFN, $CREATE_REGION_64, $CRMPSC, $CRMPSC_FILE_64, $CRMPSC_GFILE_64, $CRMPSC_GPFILE_64, $CRMPSC_PFN_64, $DELETE_REGION_64, $DELTVA_64, $DGBLSC, $MGBLSC_GPFN_64

Condition Values Returned

SS$_NORMAL

The service completed successfully. The specified global section already exists and has been mapped.

SS$_CREATED

The service completed successfully. The specified global section did not previously exist and has been created.

SS$_ACCVIO

The *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$_EXBYTLM

The process has exceeded the byte count quota.

SS$_GBLSEC_MISMATCH

Global section type mismatch. The specified global section was found; however, it was not a global disk file section.

SS$_GPTFULL

There is no more room in the system global page table to set up page table entries for the section.

SS$_GSDFULL

There is no more room in the system space allocated to maintain control information for global sections.

SS$_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$_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 and arguments was specified.

SS$_IVSECIDCTL

The match control field of the global section identification is invalid.

SS$_NOPFNMAP

The process does not have the privilege to create or delete a section starting at a specific physical page frame number (PFNMAP).

SS$_NOPRMGBL

The process does not have the privileges to create or delete a permanent group global section (PRMGBL).

SS$_NOSHPTS

A virtual address within a shared page-table region was specified, or the region ID of a shared page-table region was specified.

SS$_NOSYSGBL

The process does not have the privileges to create or delete a system global section (SYSGBL).

SS$_NOWRTACC

The specified global section is not copy-on-reference and does not allow write access.

SS$_PAGNOTINREG

A page in the specified range is not within the specified region.

SS$_REGISFULL

The specified virtual region is full; no space is available in the region for the pages created to contain the mapped section.

SS$_TOOMANYLNAM

The logical name translation of the *gs_name_64* argument exceeded the allowed depth of 10.

SS$_VA_IN_USE

A page in the specified input address range is already mapped and the flag SEC$M_NO_OVERMAP is set, or the existing underlying page cannot be deleted because it is associated with a buffer object.

SS$_VA_NOTPAGALGN

The *start_va_64* argument is not CPU-specific page-aligned.

$CRMPSC_PFN_64 (Alpha and Integrity servers)

Create and Map Private Page Frame Section — On Alpha and Integrity server systems, allows a process to map a section of its address space to a specified physical address range represented by page frame numbers. This service creates and maps a private page frame section. This service accepts 64-bit addresses.

Format

```
SYS$CRMPSC_PFN_64
   region_id_64 ,start_pfn ,page_count ,acmode
   ,flags ,return_va_64 ,return_length_64 [,start_va_64]
```

C Prototype

```
int sys$crmpsc_pfn_64
   (struct _generic_64 *region_id_64, unsigned int start_pfn,
    unsigned int page_count, unsigned int acmode, unsigned int flags,
```

```
void *(*(return_va_64)), unsigned __int64 *return_length_64,
__optional_params); *return_length_64,...);)
```

Arguments

region_id_64

OpenVMS usage:	region identifier
type:	quadword (unsigned)
access:	read only
mechanism:	by 32- 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.

start_pfn

OpenVMS usage:	page frame number
type:	longword (unsigned)
access:	read only
mechanism:	by value

The CPU-specific page frame number where the section begins in memory.

page_count

OpenVMS usage:	CPU-specific page count
type:	longword (unsigned) on Alpha, quadword (unsigned) on Integrity servers
access:	read only
mechanism:	by value

Length of the page frame section in CPU-specific pages.

acmode

OpenVMS usage:	access_mode
type:	longword (unsigned)

access:	read only
mechanism:	by value

Access mode that is to be the owner of the pages created during the mapping. The *acmode* argument is a longword containing the access mode.

The $PSLDEF macro in STARLET.MLB and the file PSLDEF.H in SYS$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. The calling process can delete pages only if those pages are owned by an access mode equal to or less privileged than the access mode of the calling process.

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 the characteristics of the private section to be created. 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 $CRMPSC_PFN_64 service:

Flag	Description
SEC$M_ARGS64	Indicates that all parameters, specifically *start_pfn* and *page_count*, are passed as 64-bit numbers. This flag is ignored on OpenVMS Alpha but must be set on Integrity server systems. If the flag is not set on Integrity servers, the error code SS$_IVSECFLG is returned.
SEC$M_EXPREG	Pages are mapped into the first available space at the current end of the specified region.
SEC$M_NO_OVERMAP	Pages cannot overmap existing address space. By default, pages can overmap existing address space.

Flag	Description
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_UNCACHED	Flag that must be set when a PFN-mapped section is created if this section must be treated as uncached memory. Flag is ignored on Alpha systems; it applies only to Integrity server systems.
SEC$M_WRT	Pages form a read/write section. By default, pages form a read-only section.

All other bits in the *flags* argument are reserved for future use to OpenVMS and should be specified as 0. The condition value SS$_IVSECFLG is returned if any undefined bits are set or if an invalid combination of flags is set.

return_va_64

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

The lowest process virtual address into which the private page frame section was mapped. The *return_va_64* argument is the 32- 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- or 64-bit reference

The length of the virtual address range mapped. The *return_length_64* argument is the 32- or 64-bit virtual address of a naturally aligned quadword into which the 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 private page frame 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$_IVESCFLG is returned.

Description

The Create and Map Private Page Frame Section service allows a process to create a map to a private page frame section. Creating a private page frame section involves defining certain physical page numbers (PFNs) as a section. The section is mapped from a low address to a high address whether the section is mapped in a region that grows from low to high addresses or from high to low addresses.

All global page frame sections are permanent. Pages mapped by SEC$M_PFNMAP are not included in or charged against the process's working set; they are always valid. Do not lock these pages in the working set by using $LKWSET_64; this can result in a machine check if they are in I/O space.

If the condition value SS$_ACCVIO be 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

PFNMAP privilege is required to create a page frame section.

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.

Related Services

$CREATE_REGION_64, $CRMPSC, $CRMPSC_FILE_64, $CRMPSC_GFILE_64, $CRMPSC_GPFILE_64, $CRMPSC_GPFN_64, $DELETE_REGION_64, $DELTVA_64

Condition Values Returned

SS$_NORMAL

The service completed successfully.

SS$_ACCVIO

The *return_va_64* argument or the *return_length_64* argument cannot be written by the caller.

SS$_INSFWSL

The process's working set limit 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$_IVREGID

Invalid region ID specified.

SS$_IVSECFLG

An invalid flag, a reserved flag, or an invalid combination of flags and arguments was specified.

SS$_NOPFNMAP

The process does not have the privilege to create a section starting at a specific physical page frame number (PFNMAP).

SS$_NOSHPTS

The region ID of a shared page table region was specified.

SS$_PAGNOTINREG

A page in the specified range is not within the specified region.

SS$_PAGOWNVIO

A page in the specified input address range is owned by a more privileged access mode.

SS$_REGISFULL

The specified virtual region is full; no space is available in the region for the pages created to contain the mapped section.

SS$_VA_IN_USE

A page in the specified input address range is already mapped and the flag SEC$M_NO_OVERMAP is set, or the existing underlying page cannot be deleted because it is associated with a buffer object.

SS$_VA_NOTPAGALGN

The *start_va_64* argument is not CPU-specific page aligned.

$CVT_FILENAME (Alpha and Integrity servers)

Converts String — Converts a string from RMS format to file-system (ACP-QIO) format or from file-system (ACP-QIO) format to RMS format.

Format

```
SYS$CVT_FILENAME cvttyp ,srcstr ,inflags ,outbuf ,outlen ,outflags
```

C Prototype

```
int sys$cvt_filename
```

```
(unsigned int cvttyp, void *srcstr, unsigned int inflags, void *outbuf,
 unsigned short int *outlen, unsigned int *outflags);
```

Arguments

cvttyp

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

Longword value that indicates whether the conversion is from RMS format to ACP-QIO format or vice versa.

There are two legal values for this parameter, represented by the symbols CVTFNM$C_ACPQIO_TO_RMS and CVTFNM$C_RMS_TO_ACPQIO, that are defined by the $CVTFNMDEF macro.

srcstr

OpenVMS usage:	string of bytes or words
type:	string of bytes or words
access:	read only
mechanism:	by 32-bit descriptor--fixed-length string descriptor

String to be converted by the service.

If the conversion is from RMS format to ACP-QIO format, *srcstr* is an ISO-Latin-1 or VTF-7-encoded character string. If the conversion is from ACP-QIO format to RMS format, *srcstr* is a string of byte-width or word-width characters.

The descriptor length field indicates the length of the input string in bytes, whether the characters are byte-width or word-width.

The *srcstr* argument is the 32-bit address of a descriptor that points to this string.

inflags

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

Longword flag mask indicating the characteristics of the input string.

For conversion from RMS format to ACP-QIO format, only the CVTFNM$V_NO_DELIMITERS flag is valid.

For conversion from ACP-QIO format to RMS format, legal flags are CVTFNM$V_WORD_CHARS and CVTFNM$V_NO_DELIMITERS (defined by the $CVTFNMDEF macro).

Flag	Description
CVTFNM$V_WORD_CHARS	Input source string contains word-width UCS-2 characters (ACPQIO_TO_RMS conversion only).
CVTFNM$V_NO_DELIMITERS	Input source string should be treated as an arbitrary string (such as a subdirectory name) rather than as a file name that contains (or should contain) dots or semicolons as type and version delimiters.
CVTFNM$V_FORCE_UPCASE	Causes this system service to convert each character to uppercase. (ACPQIO_TO_RMS conversion only).

outbuf

OpenVMS usage:	string of bytes or words
type:	string of bytes or words
access:	write only
mechanism:	by 32-bit descriptor--fixed-length string descriptor

The buffer into which the converted string is to be written.

If the conversion is from RMS format to ACP-QIO format, the string may consist of byte-width ISO Latin-1 characters or word-width UCS-2 characters, depending on the characters in the source string. (If any character in the source string must be converted to UCS-2, then all characters in the output buffer will be converted to UCS-2.)

If the conversion is from ACP-QIO format to RMS format, then the output string will consist of ISO Latin-1 and VTF-7 characters in RMS canonical form. (See the *VSI OpenVMS Guide to OpenVMS File Applications*).

For ACPQIO_TO_RMS conversion, if the output string contains word-width characters, the CVTFNM$V_WORD_CHARS flag in the `outflags` flag mask will be set.

The `outbuf` argument is the 32-bit address of a descriptor pointing to a buffer writable in the access mode of the caller.

outlen

OpenVMS usage:	word_unsigned
type:	word (unsigned)
access:	write only
mechanism:	by 32-bit reference

The `outlen` argument is the 32-bit address of a (16-bit) word writable in the access mode of the caller.

outflags

OpenVMS usage:	mask_longword
type:	longword (unsigned)

access:	write only
mechanism:	by 32-bit reference

Longword flag mask in which the service sets or clears flags to indicate characteristics of the output string.

For an RMS_TO_ACPQIO conversion, SYS$CVT_FILENAME sets the bit corresponding to CVTFNM$V_WORD_CHARS (defined by the $CVTFNMDEF macro) if the characters of the converted string are one-word wide rather than one-byte wide. If the characters of the converted string are one-byte wide, the service clears the CVTFNM$V_WORD_CHARS bit. All other bits are cleared by an RMS_TO_ACPQIO conversion.

The *outflags* argument is the 32-bit address of a 32-bit flag mask writable in the access mode of the caller.

Description

This service is intended to provide the conversion of a file name or a subdirectory name between the RMS format (as seen at the RMS interface) and the ACP-QIO format (as stored on disk). A file name consists of a file name, a file type, and a file version; and a subdirectory name is a string to which ".DIR;1" can be appended to form a directory file name, as stored on disk.

Prior to Version 7.2, these representations were the same. This is not necessarily the case for extended (ODS-5) file names. (For details on ODS-5 file specifications, see the *VSI OpenVMS Guide to OpenVMS File Applications*.

Depending on the value of *cvttyp*, the service will perform a conversion of a string from RMS format to ACP-QIO format or from ACP-QIO format to RMS format.

The source string is described by the argument *srcstr*, the output buffer is described by the argument *outbuf*, and the resultant string length is written to the argument *outlen*.

If any of the source string falls within the address range of the output buffer, the output string is unpredictable.

RMS-to-ACPQIO Conversion:

A string described by the *srcstr* descriptor argument is converted to an ISO Latin-1 or UCS-2 string with each character represented in a form that can be passed to the ACP-QIO by the $QIO service.

If the CVTFNM$V_NO_DELIMITERS input flag is not set, the source string will be scanned and, if necessary, a dot and a semicolon will be inserted or appended as though a $PARSE were done with no default name, type, or version fields supplied. If the scan detects any delimiters indicating the presence of fields other than name (without FID), type, or version, a syntax error will be returned.

If the CVTFNM$V_NO_DELIMITERS input flag is set, individual characters will be validated and converted to their on-disk form; however, no scan will be done to determine if the type and version delimiters are present, and no delimiters will be added.

A percent sign (%) that is not preceded by the escape character (^) is converted to a question mark. An ISO Latin-1 character that is preceded by the escape character (^) is converted to the corresponding ISO Latin-1 character. A VTF-7 character (for example, ^U1234) that is preceded by the escape character (^) is converted to a UCS-2 character (for example, 0x1234).

If any character requires UCS-2 (word-width character) representation, all characters are represented in the output string with UCS-2. If no character requires word-width character representation, all characters are represented in the output string with ISO Latin-1 (byte-width) characters.

Valid characters are those that are legal in an RMS file name (file name, file type, and file version) or in an RMS subdirectory name. For example, directory delimiters "[" and "]" are not legal unless preceded by the escape character (^).

ACPQIO-to-RMS Conversion:

The string described by the *srcstr* descriptor argument is converted to the RMS canonical form for that string.

If the CVTFNM$V_NO_DELIMITERS input flag is clear, the source string must contain at least one semicolon and, to the left of the semicolon, at least one dot. If it does not, RMS$_SYN (syntax error) is returned. In the output string, all other dots and semicolons are preceded by the RMS escape character (^).

If the CVTFNM$V_NO_DELIMITERS input flag is set, any dot or semicolon encountered is preceded in the output string by the RMS escape character (^).

The CVTFNM$V_WORD_CHARS flag of the *inflags* argument indicates whether the input string is to be interpreted as having byte-width (ISO Latin-1) or word-width (UCS-2) characters. If the argument indicates word-width characters, but the input length value is an odd number, a syntax error is returned.

Questions marks are converted to percent signs; percent signs are preceded by the escape character (^). UCS-2 characters are converted to VTF-7 characters. All characters are represented in RMS canonical form.

$CVT_FILENAME Usage:

You can use this service to compare two file names using the same conventions as RMS.

For an example program, see: [SYSHLP.EXAMPLES]FILENAME_COMPARE.C.

Required Access or Privileges

None.

Required Quota

None.

Related Services

None.

Condition Values Returned

SS$_NORMAL

The service completed successfully.

SS$_BADPARAM

Unrecognized conversion type, extraneous input flags set, or zero-length input string.

SS$_INSFARG

Not enough arguments provided.

SS$_TOO_MANY_ARGS

Too many arguments provided.

RMS$_SYN

The service could not translate one or more characters in the strings described by the *srcstr* argument. Either the input string has word-width characters but odd byte-length (ACPQIO_TO_RMS only), or the CVTFNM$V_NO_DELIMITERS input flag was clear, and the input string did not contain both type and version delimiters.

SS$_BUFFEROVF

The output buffer was not large enough to accommodate the converted string.

$DACEFC

Disassociate Common Event Flag Cluster — Releases the calling process's association with a common event flag cluster.

Format

```
SYS$DACEFC efn
```

C Prototype

```
int sys$dacefc (unsigned int efn);
```

Argument

efn

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

Number of any event flag in the common cluster to be disassociated. The *efn* argument is a longword containing this number; however, $DACEFC uses only the low-order byte. The number must be in the range of 64 through 95 for cluster 2, and 96 through 127 for cluster 3.

Description

The Disassociate Common Event Flag Cluster service disassociates the calling process from a common event flag cluster and decreases the count of processes associated with the cluster accordingly. When the image associated with a cluster exits, the system disassociates the cluster. When the count of processes associated with a temporary cluster or with a permanent cluster that is marked for deletion reaches 0, the cluster is automatically deleted.

If a process issues this service specifying an event flag cluster with which it is not associated, the service completes successfully.

Required Access or Privileges

A calling process must have PRMCEB privilege to delete a permanent common event flag cluster.

Required Quota

None

Related Services

$ASCEFC, $CLREF, $DLCEFC, $READEF, $SETEF, $WAITFR, $WFLAND, $WFLOR

Condition Values Returned

SS$_NORMAL

The service completed successfully.

SS$_ILLEFC

You specified an illegal event flag number. The number must be in the range of event flags 64 through 127.

SS$_INTERLOCK

The bit map lock for allocating common event flag clusters from the specified shared memory is locked by another process.

$DALLOC

Deallocate Device — Deallocates a previously allocated device.

Format

```
SYS$DALLOC [devnam] ,[acmode]
```

C Prototype

```
int sys$dalloc (void *devnam, unsigned int acmode);
```

Arguments

devnam

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

Name of the device to be deallocated. The *devnam* argument is the address of a character string descriptor pointing to the device name string. The string might be either a physical device name or a logical name. If it is a logical name, it must translate to a physical device name.

If you do not specify a device name, all devices allocated by the process from access modes equal to or less privileged than that specified are deallocated.

acmode

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

Access mode from which the deallocation is to be performed. The *acmode* argument is a longword containing the access mode. The $PSLDEF macro defines the following symbols for the four access modes.

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

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

Description

The Deallocate Device service deallocates a previously allocated device. The issuing process relinquishes exclusive use of the device, thus allowing other processes to assign or allocate that device. You can deallocate an allocated device only from access modes equal to or more privileged than the access mode from which the original allocation was made.

This service does not deallocate a device if, at the time of deallocation, the issuing process has one or more I/O channels assigned to the device; in such a case, the device remains allocated.

At image exit, the system automatically deallocates all devices that are allocated at user mode.

If you attempt to deallocate a mailbox, success is returned but no operation is performed.

Required Access or Privileges

None

Required Quota

None

Related Services

$ALLOC, $ASSIGN, $BRKTHRU, $BRKTHRUW, $CANCEL, $CREMBX, $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 device name string or string descriptor cannot be read by the caller.

SS$_DEVASSIGN

The device cannot be deallocated because the process still has channels assigned to it.

SS$_DEVNOTALLOC

The device is not allocated to the requesting process.

SS$_IVDEVNAM

You did not specify a device name string, or the device name string contains invalid characters.

SS$_IVLOGNAM

The device name string has a length of 0 or has more than 63 characters.

SS$_NONLOCAL

The device is on a remote node.

SS$_NOPRIV

The device was allocated from a more privileged access mode.

SS$_NOSUCHDEV

The specified device does not exist in the host system.

$DASSGN

Deassign I/O Channel — Deassigns (releases) an I/O channel previously acquired using the Assign I/O Channel ($ASSIGN) service.

Format

```
SYS$DASSGN chan
```

C Prototype

```
int sys$dassgn (unsigned short int chan);
```

Argument

chan

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

Number of the I/O channel to be deassigned. The *chan* argument is a word containing this number.

Description

The Deassign I/O Channel service deassigns (releases) an I/O channel that it acquired using the Assign I/O Channel ($ASSIGN) service. You can deassign an I/O channel only from an access mode equal to or more privileged than the access mode from which the original channel assignment was made.

When you deassign a channel, any outstanding I/O requests on the channel are canceled. If a file is open on the specified channel, the file is closed.

If a mailbox was associated with the device when the channel was assigned, the link to the mailbox is cleared.

If the I/O channel was assigned for a network operation, the network link is disconnected.

If the specified channel is the last channel assigned to a device that has been marked for dismounting, the device is dismounted.

I/O channels assigned from user mode are automatically deassigned at image exit.

Required Access or Privileges

None.

Note that you should use the SHARE privilege with caution. Applications, application protocols, and device drivers coded to expect only exclusive access can encounter unexpected and errant behavior

when access to the device is unexpectedly shared. Unless the SHARE privilege is explicitly supported by the application, the application protocol, and the device driver, its use is generally discouraged. For additional information, see the *VSI OpenVMS Programming Concepts Manual*.

Required Quota

None

Related Services

$ALLOC, $ASSIGN, $BRKTHRU, $BRKTHRUW, $CANCEL, $CREMBX, $DALLOC, $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$_IVCHAN

You specified an invalid channel number, that is, a channel number of 0 or a number larger than the number of channels available.

SS$_IVCHNLSEC

A process section file is currently accessed using the specified channel number.

SS$_IVIDENT

On Alpha or Integrity server systems, you specified an invalid channel number that your process never assigned.

SS$_NOICHAN

On Alpha or Integrity server systems, you specified a channel number outside of the set of channel numbers available for your process.

SS$_NOPRIV

The specified channel is not assigned or was assigned from a more privileged access mode.

$DCLAST

Declare AST — Queues an asynchronous system trap (AST) for the calling access mode or for a less privileged access mode. On Alpha and Integrity server systems, this service accepts 64-bit addresses.

Format

```
SYS$DCLAST astadr ,[astprm] ,[acmode]
```

C Prototype

```
int sys$dclast
   (void (*astadr)(__unknown_params),  unsigned __int64 astprm,
    unsigned int acmode);
```

Arguments

astadr

OpenVMS usage:	ast_procedure
type:	procedure value
access:	call without stack unwinding
mechanism:	by 32- or 64-bit reference

AST service routine to be executed. On Alpha and Integrity server systems, the *astadr* argument is the 32- or 64-bit address of this routine.

astprm

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

AST parameter to be passed to the AST routine specified by the *astadr* argument. On Alpha and Integrity server systems, the *astprm* argument is a quadword value containing this parameter.

acmode

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

Access mode for which the AST is to be declared. The most privileged access mode used is the access mode of the caller. The resultant mode is the access mode for which the AST is declared.

Description

The Declare AST service queues an AST for the calling access mode or for a less privileged access mode. For example, a routine executing in supervisor mode can declare an AST for either supervisor or user mode.

The service does not validate the address of the AST service routine. If you specify an illegal address (such as 0), an access violation occurs when the AST service routine is given control.

Required Access or Privileges

None

Required Quota

The $DCLAST service requires system dynamic memory and uses the AST limit (ASTLM) quota of the process.

Related Services

$SETAST, $SETPRA

Condition Values Returned

SS$_NORMAL

The service completed successfully.

SS$_EXQUOTA

The process has exceeded its AST limit (ASTLM) quota.

SS$_INSFMEM

The system dynamic memory is insufficient for completing the service.

$DCLCMH

Declare Change Mode or Compatibility Mode Handler — Specifies the address of a routine to receive control when a Change Mode to User or Change Mode to Supervisor instruction trap occurs.

Format

```
SYS$DCLCMH addres ,[prvhnd] ,[type]
```

C Prototype

```
int sys$dclcmh
   (int (*addres)(__unknown_params), void *(*(prvhnd)), char type);
```

Arguments

addres

OpenVMS usage:	address
type:	longword (unsigned)
access:	read only

mechanism:	by reference

Routine to receive control when a change mode trap or a compatibility mode fault occurs. The *addres* argument is the exception handling code in the address space of the calling process.

If you specify the *addres* argument as 0, $DCLCMH clears the previously declared handler.

prvhnd

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

Address of a previously declared handler. The *prvhnd* argument is the address of a longword containing the address of the previously declared handler.

type

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

Handler type indicator. The *type* argument is a longword value. The value 0 (the default) indicates that a change mode handler is to be declared for the access mode at which the request is issued; the value 1 specifies that a compatibility mode handler is to be declared.

Description

On Alpha and Integrity server systems, the Declare Change Mode or Compatibility Mode Handler service calls the change mode handler as a normal procedure (that is, with a standard procedure call). The change mode handler must exit by performing a standard procedure return to the change mode dispatcher.

Arguments (for example, the change mode code) passed between the routine that issued the change mode instruction and the change mode handler are strictly by agreement between the two procedures.

The following MACRO code example shows a subroutine calling Change Mode to User. The example is written for Alpha and Integrity servers users porting from VAX systems.

```
CHG_MD:  .CALL_ENTRY
         CHMU
         RET
```

Call this subroutine from any program that requires a Change Mode to User instruction to be invoked.

A change mode handler provides users with a dispatching mechanism similar to that used for system service calls. It allows a routine that executes in supervisor mode to be called from user mode. You declare the change mode handler from supervisor mode; then when the process executing in user mode

issues a Change Mode to Supervisor instruction, the change mode handler receives control and executes in supervisor mode.

The top longword of the stack contains the zero-extended change mode code. The change mode handler must exit by removing the change mode code from the stack and issuing an REI instruction.

The operating system uses compatibility mode handlers to bypass normal condition handling procedures when an image executing in compatibility mode causes a compatibility mode exception. Before transferring control to the compatibility mode handler, the system saves the compatibility exception code, the registers R0 through R6, and the PC and PSL in a 10-longword array starting at the location CTL$AL_CMCNTX. Before the compatibility mode handler exits, it must restore the saved registers R0 through R6, push the saved PC and PSL onto the stack, and exit by issuing an REI instruction.

Required Access or Privileges

You can declare a change mode or compatibility mode handler only from user or supervisor mode.

Required Quota

None

Related Services

$SETEXV, $UNWIND

Condition Values Returned

SS$_NORMAL

The service completed successfully.

SS$_ACCVIO

The longword to receive the address of the previous change mode handler cannot be written by the caller.

[1]**SS$_IVSSRQ**

The call to the service is invalid because it attempted to declare a compatibility mode handler on Alpha and Integrity server systems.

$DCLEXH

Declare Exit Handler — Declares an exit handling routine that receives control when an image exits.

Format

```
SYS$DCLEXH desblk
```

[1]Alpha and Integrity servers specific

C Prototype

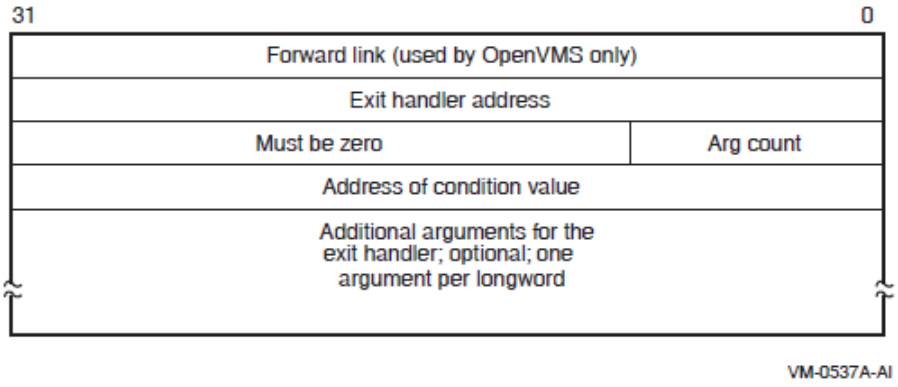
```
int sys$dclexh (void *desblk);
```

Argument

desblk

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

Exit handler control block. The *desblk* argument is the address of this control block. This control block, which describes the exit handler, is depicted in the following diagram.



Description

The Declare Exit Handler service declares an exit handler routine that receives control when an image exits. Image exit normally occurs when the image currently executing in a process returns control to the operating system. Image exit might also occur when you call the Exit ($EXIT) or Force Exit ($FORCEX) service. Process exit handlers are not invoked when a process is deleted (such as using a $DELPRC call, for example).

Exit handlers are described by exit control blocks. The operating system maintains a separate list of these control blocks for user, supervisor, and executive modes, and the $DCLEXH service adds the description of an exit handler to the front of one of these lists. The actual list to which the exit control block is added is determined by the access mode of the $DCLEXH caller.

At image exit, the image context, the image stack pointers, and the validity of any variables allocated on the stack are all indeterminate. Accordingly, the exit handler control block and any variables accessed by each exit handler must all be declared using non-volatile semantics. Examples of such non-volatile declarations include the BASIC and Fortran COMMON construct, and the C static storage class.

The declared exit handlers are called from the least-privileged processor mode to the most-privileged mode, and the exit handler(s) for each processor mode are called in the reverse order from which they were originally declared. Each exit handler is executed only once; it must be explicitly redeclared by the application if it is to be executed again.

The exit handler routine is called as a normal procedure with the argument list specified in the third through *n*th longwords of the exit control block. The first argument is always the address of a user-

allocated longword to receive the system status code indicating the reason for the exit; the system always fills in the referenced longword before calling the exit handler. Accordingly, the exit handler routine receives a pointer to the status code as its first argument. Application programmers can append zero or more additional application-specific longword arguments for use within the exit handler routine, with the total number of arguments controlled by the value specified in the argument count field.

You can call this service only from user, supervisor, and executive modes.

Following is a BASIC programming example for this service. To view a C example, see the *VSI OpenVMS Programming Concepts Manual*.

```
program         DCLEXH_EXAMPLE
        option type = explicit

        external long EXIT_HANDLER
        external long function SYS$DCLEXH
        external long function SYS$EXIT
        external long function LIB$STOP

        declare long RETURN_STATUS

        record EXIT_DESCRIPTOR
          long FORWARD_LINK
          long HANDLER_ADDR
          long ARG_COUNT
          long CONDITION_VALUE_PTR
          long RANDOM_EXAMPLE_VALUE
          ! borrow part of the record structure for data storage...
          long CONDITION_VALUE
        end record EXIT_DESCRIPTOR

        ! declare the exit handler block in non-volatile storage
        common (SaveBlock) EXIT_DESCRIPTOR EXHBLK

        PRINT
        PRINT "DCLEXH_EXAMPLE initializing..."
        PRINT

        EXHBLK::FORWARD_LINK = 0%
        EXHBLK::HANDLER_ADDR = loc(EXIT_HANDLER)
        EXHBLK::ARG_COUNT = 2%
        EXHBLK::CONDITION_VALUE_PTR = loc(EXHBLK::CONDITION_VALUE )
        EXHBLK::RANDOM_EXAMPLE_VALUE = 303147%

        PRINT "Calling SYS$DCLEXH..."

        RETURN_STATUS = SYS$DCLEXH (EXHBLK by ref)
        call LIB$STOP (RETURN_STATUS by value) if (RETURN_STATUS and 1%) = 0%

        PRINT "SYS$DCLEXH called..."

        PRINT "Calling SYS$EXIT..."

        call SYS$EXIT(RETURN_STATUS by value)

end

function LONG EXIT_HANDLER(long CONDITION_VALUE, long RANDOM_VALUE by value)

        ! the exit handler gains control effectively after the main
        ! program module has exited.  Direct access to (or otherwise
        ! sharing) variables used by the main routine requires explicit
        ! storage allocation control.
```

```
        option type = explicit

        print "EXIT_HANDLER invoked..."
        print "CONDITION_VALUE:  ", CONDITION_VALUE
        print "RANDOM_VALUE:     ", RANDOM_VALUE

        PRINT
        PRINT "DCLEXH_EXAMPLE done..."
        PRINT

end function
```

Required Access or Privileges

None

Required Quota

None

Related Services

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

The Cancel Exit Handler ($CANEXH) service removes an exit control block from the list.

Condition Values Returned

SS$_NORMAL

The service completed successfully.

SS$_ACCVIO

The first longword of the exit control block cannot be written by the caller.

SS$_IVSSRQ

The call to the service is invalid because it was made from kernel mode.

SS$_NOHANDLER

The exit handler control block address was not specified or was specified as 0.

$DECLARE_RM

Declare Resource Manager — Creates a new Resource Manager instance (RMI) in the calling process.

Format

```
SYS$DECLARE_RM
   [efn] ,[flags] ,iosb ,[astadr] ,[astprm] ,rm_id ,event_handler
```

```
   ,[part_name] [,[rm_context] ,[acmode] ,[tm_log_id] ,[event_mask]]
```

C Prototype

```
int sys$declare_rm
   unsigned int efn, unsigned int flags, struct _iosb *iosb,
   void (*astadr)(__unknown_params), int astprm, unsigned int *rm_id,
   void (*event_handler)(__unknown_params),… ;
```

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 used.

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, described in Table 23. All undefined bits must be 0. If this argument is omitted, no flags are used.

Table 23. $DECLARE_RM Option Flags

Flag Name	Description
DDTM$M_SYNC	Specifies successful synchronous completion by returning SS$_SYNCH. When SS$_SYNCH is returned, the AST routine is not called, the event flag is not set, and the I/O status block is not filled in.
DDTM$M_VOLATILE	Set this flag for the new RMI to be volatile. With this flag set, the DECdtm transaction manager will not log information about any RM participants associated with the new RMI. Resource managers that never perform recovery should set this flag. If this flag is clear, the new RMI is not volatile. The DECdtm transaction manager will log the following information about each RM participant associated with the new RMI: • The name of the RM participant. • The identifier (TID) of the transaction in which it is participating.

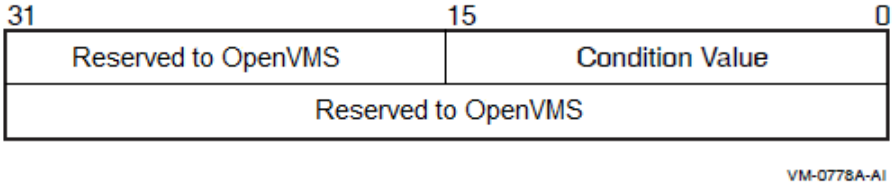
Flag Name	Description
	If this flag is clear and a recoverable failure occurs, such as a system crash, the resource manager can use the $GETDTI system service to query the transaction log to determine the outcome of the transactions in which it was participating before the failure occurred.

iosb

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

The 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:



astadr

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

The AST routine that is executed when the service completes, if SS$_NORMAL is returned in R0. The *astadr* argument is the address of the entry mask of this routine. The routine is executed in the same access mode as that of the caller of the $DECLARE_RM service.

astprm

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

The AST parameter that is passed to the AST routine specified by the *astadr* argument.

rm_id

OpenVMS usage:	identifier
type:	longword (unsigned)

access:	write only
mechanism:	by reference

Longword in which the identifier (RM_ID) of the new RMI is returned. This identifier is unique within the calling process at any time.

event_handler

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

The new RMI's event handler. This routine is called to report an event to the new RMI or one of its RM participants. The *event_handler* argument is the address of the entry mask of this routine. An event handler must be specified.

This routine is called as an AST delivered by the DECdtm transaction manager.

The AST is executed in the access mode specified by the *acmode* argument. The AST parameter is the address of a DECdtm event report block that contains an event report.

The DECdtm transaction manager reports events to an RMI and the RM participants associated with it using ASTs executed in the access mode specified in the call to $DECLARE_RM that created that RMI.

The DECdtm transaction manager creates an event report block, and passes its address to the AST routine in the parameter of the AST. Each event report block contains:

- The identifier of the event report.
- A code that describes the event.
- The identifier (TID) of the transaction.
- The name of the RM participant or RMI.
- The context of the RM participant or RMI.
- Other data that depend on the type of the event.

Table 24 describes the fields in an event report block, in alphabetical order:

Table 24. Fields in an Event Report Block

Symbol	Description	
DDTM$A_TID_PTR	Address of the identifier (TID) of the transaction.	
DDTM$L_ABORT_REASON	Abort reason code (longword).See the $ACK_EVENT service for a list of possible values. Present only in abort event reports.	
DDTM$L_EVENT_TYPE	A code that identifies the event (longword). The following table lists the possible values:	
	Symbol	**Event**

Symbol	Description	
	DDTM$K_ABORT	Abort
	DDTM$K_COMMIT	Commit
	DDTM$K_PREPARE	Prepare
	DDTM$K_ONE_PHASE_ COMMIT	One-phase commit
	DDTM$K_STARTED_DEFAULT	Default transaction started
	DDTM$K_STARTED_ NONDEFAULT	Nondefault transaction started
DDTM$L_REPORT_ID	Event report identifier (unsigned longword).	
DDTM$L_RM_CONTEXT	The context of the RM participant or RMI to which the event report is being delivered (unsigned longword).	
DDTM$Q_PART_NAME	The name of the RM participant or RMI to which the event report is being delivered (descriptor).	
DDTM$Q_TX_CLASS	The transaction class of the transaction (descriptor).	

Each event report must be acknowledged by calling $ACK_EVENT, specifying the identifier of the report. This acknowledgment need not come from AST context.

The DECdtm transaction manager delivers only one event report at a time to each RM participant. For example, if a prepare event report has been delivered to an RM participant, and the transaction is aborted while the RM participant is doing its prepare processing, then the DECdtm transaction manager does not deliver an abort event report to that RM participant until it has acknowledged the prepare event report by a call to $ACK_EVENT. Note that the DECdtm transaction manager may deliver multiple reports to an RMI.

After acknowledging the event report, the RMI or RM participant should no longer access the event report block.

part_name

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

The name of the new RMI. This is:

- The default name of its RM participants, used when a call to $JOIN_RM or $ACK_EVENT that adds one of these RM participants to a transaction does not specify the name of the new RM participant.

  When an RM participant associated with the new RMI is added to a transaction by a call to $JOIN_RM or $ACK_EVENT that has a zero *part_name* argument, then that RM participant inherits its name from the RMI. The name of that RM participant is the same as the name of the RMI.

- The string passed in the participant name field of Transaction Started event reports delivered to the new RMI.

This string must be no longer than 32 characters.

If this argument is omitted, the name of the new RMI is the null string.

To ensure smooth operation in a mixed-network environment, refer to the chapter entitled Managing DECdtm Services in the *VSI OpenVMS System Manager's Manual*, for information on defining node names.

rm_context

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

The context of the new RMI. This is:

- The default context of its RM participants, used when a call to $JOIN_RM or $ACK_EVENT that adds one of these RM participants to a transaction does not specify the context of the new RM participant.

  When an RM participant associated with the new RMI is added to a transaction by a call to $JOIN_RM or $ACK_EVENT that has a zero *rm_context* argument, then that RM participant inherits its context from the RMI. The context of that RM participant is the same as the context of the RMI.

- The string passed in the context field of Transaction Started event reports delivered to the new RMI.

If this argument is omitted, the context of the new RMI is 0.

acmode

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

The access mode of the new RMI. This is:

- The access mode at which the ASTs delivered to its event handler are to be executed.
- The least privileged access mode that the caller must be in to call $ACK_EVENT to acknowledge an event report delivered to the new RMI or to its RM participants.
- The least privileged access mode that the caller must be in to delete the new RMI by calling $FORGET_RM.
- The least privileged access mode that the caller must be in to call $JOIN_RM to add a new RM participant associated with the new RMI.
- The most privileged access mode of new branches that this RMI is interested in, if the *event_mask* argument requests events of type Transaction Started.

The call to $START_TRANS or $START_BRANCH that adds a new branch to a transaction specifies the access mode of that transaction within this process. The DECdtm transaction manager reports a Transaction Started event to the new RMI only if the access mode of the transaction is the same as or less privileged than the access mode of the new RMI.

For example, if the access mode of the new RMI is supervisor, it will receive a Transaction Started event when a branch of the calling process is added to a transaction only if the access mode of that transaction is user or supervisor.

The access mode of the new RMI is the least privileged of:

- The access mode of the caller.
- The access mode specified by the *acmode* argument.

If this argument is omitted, the access mode of the new RMI is the same as the access mode of the caller.

tm_log_id

OpenVMS usage:	DECnet_uid
type:	octaword (unsigned)
access:	write only
mechanism:	by reference

The globally unique identifier of the transaction log for the local node. This identifier is used during resource manager recovery to check that the correct DECdtm transaction manager log is used. See $GETDTI for more information.

To ensure smooth operation in a mixed-network environment, refer to the chapter entitled Managing DECdtm Services in the *VSI OpenVMS System Manager's Manual*, for information on defining node names.

event_mask

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

Requests the types of event to be reported to the new RMI and to its RM participants. The only type of event that can be reported to the new RMI is a Transaction Started event (a default or non-default transaction started event). The following types of event can be reported to its RM participants:

- Abort events
- Commit events
- One-phase commit events
- Prepare events

The *event_mask* argument is a longword bit mask that is the logical OR of each bit set, where each bit corresponds to an event. The $DDTMDEF module defines a symbolic name for each flag bit. Table 25 describes the flags. All undefined bits must be 0.

If this argument is omitted, the following events are requested:

- Abort events
- Commit events
- One-phase commit events
- Prepare events

Table 25. $DECLARE_RM Event Selection Flags

Flag Name	Description
DDTM$M_EV_ABORT	Specifies that abort events are to be reported to the RM participants associated with the new RMI. If this flag is set, when an abort event occurs for a transaction, the DECdtm transaction manager delivers an abort event report to each RM participant in the transaction that is associated with the new RMI.
DDTM$M_EV_COMMIT	Specifies that commit events are to be reported to the RM participants associated with the new RMI. If this flag is set, when the DECdtm transaction manager decides that the outcome of a transaction is commit, it delivers a commit event report to each RM participant in the transaction that is associated with the new RMI.
DDTM$M_EV_PREPARE	Specifies that prepare events are to be reported to the RM participants associated with the new RMI. If this flag is set, when the DECdtm transaction manager initiates the commit protocol (in response to a call to $END_TRANS) to determine the outcome of a transaction, it reports a prepare event to each RM participant in the transaction that is associated with the new RMI. The acknowledgment of a prepare event is a vote on the outcome of the transaction. See $ACK_EVENT for more information.
DDTM$M_EV_TRANS_START	Specifies that events of type Transaction Started are to be reported to the new RMI. Events of type Transaction Started are: • Default transaction started events. • Non-default transaction-started events. If this flag is set, the DECdtm transaction manager will report one of these events to the new RMI whenever a new branch in the calling process is added to a transaction, provided that the access mode of the new branch is not more privileged than the access mode of the new RMI. The acknowledgment of that event report may add a new RM participant associated with the new RMI to that transaction. See the description of the *acmode* argument for a discussion of access modes.

Description

The $DECLARE_RM system service creates a new Resource Manager instance (RMI) in the calling process and returns its identifier.

Preconditions for successful completion of $DECLARE_RM include:

- The local node must have a DECdtm transaction log.
- The TP_SERVER process must be running on the local node.

When $DECLARE_RM completes successfully, a new RMI is created in the calling process, and its identifier is returned. The new RMI has no RM participants. They are added to transactions by subsequent calls to $JOIN_RM or $ACK_EVENT.

DECdtm events for the RMI and its RM participants, as specified by *event_mask*, are reported to the specified event handler.

If an RM does not specify DDTM$M_EV_PREPARE, its RM participants do not have a vote on the outcome of their transactions. The DECdtm transaction manager assumes that their votes are "yes".

If an RM does not specify DDTM$M_EV_ABORT or DDTM$M_EV_COMMIT, DECdtm forgets the involvement of the RM participant associated with the RMI in the transaction when the corresponding event occurs.

A one-phase commit event is reported to an RM participant if:

- Both the DDTM$M_EV_PREPARE and DDTM$M_EV_COMMIT flags are set.
- It is the only RM participant in the transaction.
- It is running in the process that started the transaction (the process that called $START_TRANS).

The new RMI is deleted from the calling process:

- On termination of the calling process.
- On termination of the current image, if the access mode of the RMI was user mode.
- On successful completion of a call to $FORGET_RM in the calling process that passes its identifier.

There is also a wait form of the service, $DECLARE_RMW.

Required Privileges

None

Required Quotas

BYTLM, ASTLM

Related Services

$ABORT_TRANS, $ABORT_TRANSW, $ACK_EVENT, $ADD_BRANCH, $ADD_BRANCHW, $CREATE_UID, $DECLARE_RMW, $END_BRANCH, $END_BRANCHW,

$END_TRANS, $END_TRANSW, $FORGET_RM, $FORGET_RMW, $GETDTI, $GETDTIW, $GET_DEFAULT_TRANS, $JOIN_RM, $JOIN_RMW, $SETDTI, $SETDTIW, $SET_DEFAULT_TRANS, $SET_DEFAULT_TRANSW, $START_BRANCH, $START_BRANCHW, $START_TRANS, $START_TRANSW, $TRANS_EVENT, $TRANS_EVENTW

Condition Values Returned

SS$_NORMAL

If returned in R0, the request was successfully queued. If 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 to the caller.

SS$_BADPARAM

Either the options flags were invalid or the event mask flags were invalid.

SS$_EXASTLM

The process AST limit (ASTLM) was exceeded.

SS$_ILLEFC

The event flag number was invalid.

SS$_INSFARGS

A required argument was missing.

SS$_INSFMEM

There was insufficient system dynamic memory for the operation.

SS$_INVBUFLEN

The string passed in the *part_name* argument was too long.

SS$_NOLOG

The local node did not have a transaction log.

SS$_TPDISABLED

The TP_SERVER process was not running on the local node.

$DECLARE_RMW

Declare Resource Manager and Wait — Creates a new Resource Manager instance (RMI) in the calling process. $DECLARE_RMW always waits for the request to complete before returning to the caller. Other than this, it is identical to $DECLARE_RM.

Format

```
SYS$DECLARE_RMW
   [efn] ,[flags] ,iosb ,[astadr] ,[astprm] ,rm_id ,event_handler
   ,[part_name] [,[rm_context] ,[acmode] ,[tm_log_id] ,[event_mask]]
```

C Prototype

```
int sys$declare_rmw
   (unsigned int efn, unsigned int flags, struct _iosb *iosb,
    void (*astadr)(__unknown_params), int astprm, unsigned int *rm_id,
    void (*event_handler)(__unknown_params),...);
```

$DELETE

Removes Existing Record — The Delete service removes an existing record from a relative or indexed file. You cannot use this service when processing sequential files. For additional information about this service, see the *VSI OpenVMS Record Management Services Reference Manual*.

$DELETE_BUFOBJ (Alpha and Integrity servers)

Delete Buffer Object — On Alpha and Integrity server systems, deletes a buffer object previously created by the $CREATE_BUFOBJ_64 system service. This service accepts 64-bit addresses.

Format

```
SYS$DELETE_BUFOBJ buffer_handle_64
```

C Prototype

```
int sys$delete_bufobj (struct _generic_64 *buffer_handle_64);
```

Arguments

buffer_handle_64

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

The buffer object to be deleted. The *buffer_handle_64* argument is the 32- or 64-bit address of a 2-longword array previously returned by a $CREATE_BUFOBJ_64 call.

Description

The Delete Buffer Object system service deletes the buffer object identified by the *buffer_handle_64* argument. The associated memory is made free to be paged, swapped, or deleted.

Buffer objects are also automatically deleted at image rundown.

Required Privileges

None

Required Quota

None

Related Services

$CREATE_BUFOBJ_64

Condition Values Returned

SS$_NORMAL

The service completed successfully.

SS$_ACCVIO

The *buffer_handle_64* argument cannot be read by the caller.

SS$_BADPARAM

The *buffer_handle_64* argument is not a valid buffer handle.

SS$_NOPRIV

The buffer object was created by a more privileged access mode than the caller's access mode.

$DELETE_GALAXY_LOCK (Alpha Only)

Delete an OpenVMS Galaxy Lock — Invalidates an OpenVMS Galaxy lock and deletes it. Note that this system service is supported only in an OpenVMS Alpha Galaxy environment. For more information about programming with OpenVMS Galaxy system services, see the *VSI OpenVMS Alpha Partitioning and Galaxy Guide*.

Format

```
SYS$DELETE_GALAXY_LOCK handle
```

C Prototype

```
int sys$delete_galaxy_lock (unsigned __int64 lock_handle);
```

Arguments

handle

OpenVMS usage:	galaxy lock handle
type:	quadword (unsigned)
access:	read
mechanism:	input by value

The 64-bit lock handle that identifies the lock to be deleted. This value is returned by SYS$CREATE_GALAXY_LOCK.

Description

This service invalidates the OpenVMS Galaxy lock and deletes it. The memory for the lock is not truly deleted; however, it is put on a free list for later use.

Required Access or Privileges

CMKRNL, SHMEM

Required Quota

None

Related Services

$ACQUIRE_GALAXY_LOCK, $CREATE_GALAXY_LOCK, $CREATE_GALAXY_LOCK_TABLE, $DELETE_GALAXY_LOCK_TABLE, $GET_GALAXY_LOCK_INFO, $GET_GALAXY_LOCK_SIZE, $RELEASE_GALAXY_LOCK

Condition Values Returned

SS$_NORMAL

Normal completion.

SS$_BADLCKTBL

Galaxy lock table is corrupt.

SS$_IVLOCKID

Invalid lock id.

SS$_IVLOCKTBL

Invalid lock table.

SS$_NOCMKRNL

Operation requires CMKRNL privilege.

SS$_NOSHMEM

Operation requires SHMEM privilege.

$DELETE_GALAXY_LOCK_TABLE (Alpha Only)

Delete OpenVMS Galaxy Lock Table — Deletes an OpenVMS Galaxy locktable. Note that this system service is supported only in an OpenVMS Alpha Galaxy environment. For more information about programming with OpenVMS Galaxy system services, see the *VSI OpenVMS Alpha Partitioning and Galaxy Guide*.

Format

```
SYS$DELETE_GALAXY_LOCK_TABLE handle
```

C Prototype

```
int sys$delete_galaxy_lock_table (unsigned int *lcktbl_handle);
```

Arguments

lcktbl_handle

OpenVMS usage:	lock table handle
type:	longword (unsigned)
access:	read
mechanism:	input by value

The 32-bit lock table handle that identifies the table to be deleted. This value is returned by SYS$CREATE_GALAXY_LOCK_TABLE.

Description

This service deletes an OpenVMS Galaxy lock table. If there are no longer any mappers of the locktable section, the table is deleted.

Required Access or Privileges

CMKRNL, SHMEM

Required Quota

None

Related Services

$ACQUIRE_GALAXY_LOCK, $CREATE_GALAXY_LOCK_TABLE, $DELETE_GALAXY_LOCK, $DELETE_GALAXY_LOCK, $GET_GALAXY_LOCK_INFO, $GET_GALAXY_LOCK_SIZE, $RELEASE_GALAXY_LOCK

Condition Values Returned

SS$_NORMAL

Normal completion.

SS$_BADPARAM

Bad parameter value.

SS$_IVLOCKID

Invalid lock id.

SS$_IVLOCKTBL

Invalid lock table.

SS$_NOCMKRNL

Operation requires CMKRNL privilege.

SS$_NOSHMEM

Operation requires SHMEM privilege.

$DELETE_INTRUSION

Delete Intrusion Records — Searches for and deletes all records in the intrusion database matching the caller's specifications.

Format

```
SYS$DELETE_INTRUSION user_criteria ,[flags]
```

C Prototype

```
int sys$delete_intrusion (void *user_criteria, unsigned int flags);
```

Arguments

user_criteria

OpenVMS usage:	char_string or item_list_3
type:	character-coded text string or longword (unsigned)

access:	read only
mechanism:	by descriptor–fixed-length string descriptor or by reference

If the CIA$M_ITEMLIST flag is FALSE:

The *user_criteria* argument is the description of intruder or suspect. This 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 1 to 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-Plus address, local failed user name, or local terminal. 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 *VSI OpenVMS Guide to System Security*.

Wildcards are allowed for the *user_criteria* argument. For example, if you specify an asterisk (*) for the*user_criteria* argument, the service deletes all records in the intrusion database.

If the CIA$M_ITEMLIST flag is TRUE:

The *user_criteria* argument is now the address of a 32-bit item list. If the item list is used, one item, the CIA$_USER_CRITERIAL item, must be present in the item list. The ITM$L_BUFADR should point to a buffer containing the specified user criteria.

The following table lists the valid item descriptions for the *user_criteria* argument:

Item	Description
CIA$_SCSNODE_LIST	Address of a list of 8-character null-padded SCS nodenames for which intrusions are to be deleted.
CIA$_USER_CRITERIAL	Address of a buffer, 1-1058 bytes long, containing the intruder or suspect.

flags

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

Functional specification 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 name for the $DELETE_INTRUSION service.

Symbolic Name	Description
CIA$M_IGNORE_RETURN	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.

Symbolic Name	Description
CIA$M_ITEMLIST	If FALSE, the *user_criteria* argument is a character string. If TRUE, this argument is a 32-bit item list.

Description

The Delete Intrusion Records service deletes from the intrusion database a set of records matching the criteria you specify in the *user_criteria* argument. All records matching the criteria you specify are deleted. You do not have to call the service more than once to delete a set of records.

For example, if you specify an asterisk (*) for the *user_criteria* argument, the service deletes all records in the intrusion database with one call.

Required Access or Privileges

$DELETE_INTRUSION requires access to the intrusion database. You must have SECURITY privilege to access the database.

Required Quota

None

Related Services

$SCAN_INTRUSION, $SHOW_INTRUSION

Condition Values Returned

SS$_NORMAL

The service completed successfully.

SS$_ACCVIO

The *user_criteria* argument cannot be read.

SS$_BADBUFLEN

The length of the *user_criteria* argument 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$_CIADBEMPTY

No records in the intrusion database.

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.

$DELETE_PROXY

Delete or Modify Proxy — Deletes an existing proxy or removes the default user or a local user from an existing proxy in the proxy database.

Format

```
SYS$DELETE_PROXY rem_node ,rem_user ,[local_user] ,[flags]
```

D Prototype

```
int sys$delete_proxy
   (void *rem_node, void *rem_user, void *local_user, unsigned int 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 deleted from or modified in the proxy database. 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.

Asterisk (*) and percent sign (%) wildcards are allowed for the remote node specification. If you specify wildcards for the *rem_node* argument, the security server searches for an exact match to the specified remote node first. If it does not find an exact match, the server performs the requested operations on all of the matching proxies in the proxy database.

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 deleted from or modified in the proxy database. 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.

Asterisk (*) and percent sign (%) wildcards are allowed for the remote user specification. If you specify wildcards for the *rem_user* argument, the server searches for an exact match to the specified remote user first. If it does not find an exact match, the server performs the requested operations on all of the matching proxies in the proxy database.

local_user

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

Local user name to delete from the proxy record specified by the *rem_node* and *rem_user* arguments in the proxy database. The *local_user* argument is the address of a character-string descriptor pointing to the local user name.

A local user name consists of 0 to 32 alphanumeric characters, including dollar signs ($) and underscores (_). If the *local_user* argument is not specified or has a length of 0, the server will delete the entire record or records specified by the *rem_node* and *rem_user* arguments from the proxy database.

If the *local_user* argument is specified, the server will delete only the user name specified by the *local_user* argument from the record specified by the *rem_node* and *rem_user* arguments. The *local_user* argument can specify either the proxy's default user or a user name in the proxy's local users list.

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 names.

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 full name. If

Symbolic Name	Description
	this flag is set, it is the caller's responsibility to ensure that the fully expanded node name is passed into the service.
PRX$M_IGNORE_RETURN	The service should not wait for a return status from the security server. No return status from the server's function will be returned to the caller.
PRX$M_EXACT	The service should match exactly the remote node and remote user and ignore wildcards.

Description

The Delete Proxy service deletes a proxy from, or modifies an existing proxy in, the proxy database.

Required Access or Privileges

$DELETE_PROXY requires access to the proxy database. To achieve access, the caller must have either SYSPRV privilege or a UIC group less than or equal to the MAXSYSGRP system parameter.

Required Quota

None

Related Services

$ADD_PROXY, $DISPLAY_PROXY, $VERIFY_PROXY

Condition Values Returned

SS$_NORMAL

The service completed successfully.

SS$_ACCVIO

The *rem_node*, *rem_user*, *local_user*, or *flags* argument cannot be read by the service.

SS$_BADBUFLEN

The length of the *rem_node*, *rem_user*, or *local_user* argument was out of range.

SS$_BADPARAM

An invalid flag was specified in the *flags* argument.

SS$_NOSYSPRV

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:

SECSRV$_BADNODENAMELEN

The node name length is out of range.

SECSRV$_BADREMUSERLEN

The remote user name length is out of range.

SECSRV$_INVALIDDELETE

You attempted to remove the last local user with no default user remaining, or you tried to remove the last default user with no local user remaining. You must have at least one local user or one default user.

SECSRV$_NOSUCHPROXY

The proxy specified by the *rem_node* and *rem_user* arguments does not exist in the proxy database.

SECSRV$_NOSUCHUSER

The specified local user does not exist in the proxy's local user list, or is not the proxy's default user.

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.

$DELETE_REGION_64 (Alpha and Integrity servers)

Delete a Virtual Region — On Alpha and Integrity server systems, deletes a virtual region within the process's address space, including all created virtual addresses within the region. This service accepts 64-bit addresses.

Format

```
SYS$DELETE_REGION_64 region_id_64 ,acmode ,return_va_64 ,return_length_64
```

C Prototype

```
nt sys$delete_region_64
   (struct _generic_64 *region_id_64, unsigned int acmode,
    void *(*(return_va_64)), unsigned __int64 *return_length_64);
```

Arguments

region_id_64

OpenVMS usage:	region identifier
type:	quadword (unsigned)

access:	read only
mechanism:	by 32- or 64-bit reference

The region ID associated with the region to be deleted. The region ID specified must be one returned by the $CREATE_REGION_64 service. You cannot specify VAC_P0, VAC_P1, or VA$C_P2.

acmode

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

Access mode associated with the call to $DELETE_REGION_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. The caller can delete pages only if those pages are owned by an access mode equal to or less privileged than the access mode of the caller.

Once all pages are deleted within the region, the region can be deleted only if the region is owned by an access mode equal to or less privileged than the access mode of the caller.

return_va_64

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

The lowest process virtual address of the pages that $DELETE_REGION_64 has successfully deleted. The *return_va_64* argument is the 32- or 64-bit virtual address of a naturally aligned quadword into which the service returns the virtual address of the first page deleted. Virtual addresses are deleted from low address to high address, regardless of the direction in which virtual addresses expand for that region.

return_length_64

OpenVMS usage:	byte count
type:	quadword (unsigned)
access:	write only

mechanism: by 32- or 64-bit reference

The length of the virtual address range that $DELETE_REGION_64 has successfully deleted. The *return_length_64* argument is the 32- or 64-bit virtual address of a naturally aligned quadword into which the service returns the length of the deleted virtual address range in bytes.

Description

The Delete a Virtual Region service is a kernel mode service that can be called from any mode. This service deletes the user-defined region specified by the *region_id_64* argument. You cannot delete the program (P0), control (P1), or 64-bit program (P2) regions.

The Delete a Virtual Region service also deletes all created virtual addresses within the specified region before deleting the region itself.

If a page within the region is owned by an access mode more privileged than the access mode of the caller, the condition value SS$_PAGOWNVIO is returned. The *return_va_64* and *return_length_64* arguments contain the virtual address range that was actually deleted by $DELETE_REGION_64. In this case, the region is not deleted because there are still some pages mapped within the region.

To delete a virtual region, the caller's access mode must be at least as privileged as the access mode associated with the region. If the caller is not privileged enough to delete the region, the condition value SS$_REGOWNVIO is returned only if all pages were successfully deleted from within the region.

If the condition value SS$_ACCVIO is returned by this service, a value *cannot* be returned in the memory locations pointed to by the *return_va_64* and *return_length_64* arguments. If the condition value SS$_ACCVIO is returned, no pages have been deleted, and the region has not been deleted.

If an error other than SS$_ACCVIO occurs, the returned address and returned length indicate the pages that were successfully deleted before the error occurred. If no pages were deleted, the *return_va_64* argument contains 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

$CREATE_REGION_64, $CRETVA_64, $CRMPSC_FILE_64, $CRMPSC_GFILE_64, $CRMPSC_GPFILE_64, $CRMPSC_GPFN_64, $CRMPSC_PFN_64, $DELTVA_64, $EXPREG_64, $GET_REGION_INFO, $MGBLSC_64, $MGBLSC_GPFN_64

Condition Values Returned

SS$_NORMAL

Successful completion. All pages within the region have been deleted as well as the region itself.

SS$_ACCVIO

The *return_va_64* argument or the *return_length_64* argument cannot be written by the caller.

SS$_IVREGID

Invalid region ID specified. This condition value is returned if P0, P1, or P2 space is specified because these regions cannot be deleted, or if no region exists for the specified ID.

SS$_REGOWNVIO

The region is owned by a more privileged access mode than the access mode of the caller. All pages within the region have been deleted; however, the region has not been deleted.

SS$_PAGOWNVIO

A page within the specified region is owned by a more privileged access mode than the access mode of the caller.

SS$_VA_IN_USE

The existing underlying page cannot be deleted because it is associated with a buffer object.

$DELLNM

Delete Logical Name — Deletes all logical names with the specified name at the specified access mode or outer access mode, or it deletes all the logical names with the specified access mode or outer access mode in a specified table. On Alpha and Integrity server systems, this service accepts 64-bit addresses.

Format

```
SYS$DELLNM tabnam ,[lognam] ,[acmode]
```

C Prototype

```
int sys$dellnm (void *tabnam, void *lognam, unsigned char *acmode);
```

Arguments

tabnam

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

Name of a logical name table or a list of tables to be searched for the logical name to be deleted. The *tabnam* argument is the 32- or 64-bit address of a descriptor that points to the table name. This argument is required.

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

If *tabnam* translates to the name of a list of tables, $DELLNM does the following:

- If you specify the *lognam* argument, $DELLNM searches (in order) each table in the list until it finds the first table that contains the specified logical name. If the logical name is at the specified access mode, $DELLNM then deletes occurrences of the logical name at the specified access mode and at outer access modes within the table.
- If you do not specify the *lognam* argument, $DELLNM deletes all of the logical names at the specified access mode or at outer access modes from the first table in the list whose access mode is equal to or less privileged than the caller's access mode.

lognam

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

Logical name to be deleted. The *lognam* argument is the 32- or 64-bit address of a descriptor that points to the logical name string.

acmode

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

Access mode to be used in the delete operation. The *acmode* argument is the 32- or 64-bit address of a byte containing this access mode. The $PSLDEF macro defines symbolic names for the four access modes.

You determine the access mode actually used in the delete operation by *maximizing* the access mode of the caller with the access mode specified by the *acmode* argument; that is, the less privileged of the two is used.

However, if you have SYSNAM privilege, the delete operation is executed at the specified access mode regardless of the caller's access mode.

If you omit this argument or specify it as 0, the access mode of the caller is used in the delete operation. The access mode used in the delete operation determines which tables are used and which names are deleted.

Description

The Delete Logical Name service deletes all logical names with the specified name at the specified access mode or outer access mode, or it deletes all the logical names with the specified access mode or

outer access mode in a specified table. If any logical names being deleted are also the names of logical name tables, then all of the logical names contained within those tables and all of their subtables are also deleted.

Required Access or Privileges

Depending on the operation, the calling process might need one of the following access or rights privileges to use $DELLNM:

- Write and delete access to the logical name table to delete a logical name
- Write access to the directory table that contains the table name, or SYSPRV privilege, to delete a shareable logical name table
- SYSNAM privilege to delete a logical name or table at an inner access mode
- GRPNAM or SYSPRV privilege to delete a logical name from a group table
- SYSNAM or SYSPRV privilege to delete a logical name from a system table

Required Quota

None

Related Services

$CRELNM, $CRELNT, $TRNLNM

Condition Values Returned

SS$_NORMAL

The service completed successfully.

SS$_ACCVIO

The service cannot access the locations specified by one or more arguments.

SS$_BADPARAM

One or more arguments have an invalid value, or a logical name table name was not specified.

SS$_INSFMEM

There is insufficient dynamic memory to build a message describing the deletion of a clusterwide name.

SS$_IVLOGNAM

The *lognam* argument specifies a string whose length is not in the required range of 1 through 255 characters.

SS$_IVLOGTAB

The *tabnam* argument does not specify a logical name table.

SS$_NOLOGNAM

The specified logical name table does not exist, or a logical name with an access mode equal to or less privileged than the caller's access mode does not exist in the logical name table.

SS$_NOLOGTAB

The specified logical name table does not exist.

SS$_NOPRIV

The caller lacks the necessary privilege to delete the logical name.

SS$_TOOMANYLNAM

The logical name translation of the table name exceeded the allowable depth (10 translations).

$DELMBX

Delete Mailbox — Marks a permanent mailbox for deletion.

Format

```
SYS$DELMBX chan
```

C Prototype

```
int sys$delmbx (unsigned short int chan);
```

Argument

chan

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

Number of the channel assigned to the mailbox that is to be deleted. The *chan* argument is a word containing this number.

Description

The Delete Mailbox service marks a permanent mailbox for deletion. The actual deletion of the mailbox and of its associated logical name assignment occur when no more I/O channels are assigned to the mailbox.

You can delete a mailbox only from an access mode equal to or more privileged than the access mode from which the mailbox channel was assigned. Temporary mailboxes are automatically deleted when their reference count goes to 0.

The $DELMBX service does not deassign the channel assigned by the caller, if any. The caller must deassign the channel with the Deassign I/O Channel ($DASSGN) service.

Required Access or Privileges

You need PRMMBX privilege to delete a permanent mailbox.

Required Quota

None

Related Services

$ALLOC, $ASSIGN, $BRKTHRU, $BRKTHRUW, $CANCEL, $CREMBX, $DALLOC, $DASSGN, $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$_DEVNOTMBX

The specified channel is not assigned to a mailbox.

SS$_IVCHAN

You specified an invalid channel number, that is, a channel number of 0 or a number larger than the number of channels available.

SS$_NOPRIV

The specified channel is not assigned to a device; the process does not have the privilege to delete a permanent mailbox or a mailbox in memory shared by multiple processors; or the access mode of the caller is less privileged than the access mode from which the channel was assigned.

$DELPRC

Delete Process — Allows a process to delete itself or another process.

Format

```
SYS$DELPRC [pidadr] ,[prcnam] ,[flags]
```

C Prototype

```
int sys$delprc (unsigned int *pidadr, void *prcnam);
```

Arguments

pidadr

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

Process identification (PID) of the process to be deleted. The *pidadr* argument is the address of a longword that contains the PID. The *pidadr* argument can refer to a process running on the local node or a process running on another node in the OpenVMS Cluster 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

Process name of the process to be deleted. The *prcnam* is the address of a character string descriptor pointing to the process name string. 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 use the *prcnam* argument to delete only processes in the same UIC group as the calling process, because process names are unique to UIC groups, and 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 groups.

flags

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

The *flags* argument can be used to control whether exit handlers are called by $DELPRC. If the *flags* argument is not specified or is specified with a zero, the system parameter DELPRC_EXIT controls what exit handlers, if any, are called by $DELPRC.

The $DELPRCSYMDEF macro defines a symbolic name for EXIT and NOEXIT. The EXIT flag should be or'd with the access mode defined by the $PSLDEF macro for the initial exit handler.

The following table describes each flag:

Flag	Description
DELPRC$M_EXIT	When set, exit handlers as specified by DELPRC$M_MODE are called. This flag is ignored for a hard suspended process.
DELPRC$M_MODE	2 bit field: values pslc_kernel, pslc_exec, pslc_super, pslc_user (from the $PSLDEF macro)
DELPRC$M_NOEXIT	Set to disable any exit handler execution

Note

Deleting the current process:

When $DELPRC is used to delete the current process, execution cannot continue in the mode from which $DELPRC was called. The first exit handlers that are called will be in the next more privileged mode relative to the mode from which $DELPRC was called (subject to options defined). For example:

- $DELPRC called from user mode can call supervisor mode exit handlers.
- $DELPRC called from exec mode can only execute kernel mode exit handlers.
- $DELPRC called from kernel mode cannot call exit handlers.

Description

The Delete Process service allows a process to delete itself or another process. If you specify neither the *pidadr* nor the *prcnam* argument, $DELPRC deletes the calling process; control is not returned. If the longword at address *pidadr* is 0, the PID of the target process is returned. This system service requires system dynamic memory.

When you delete a process or subprocess, a termination message is sent to its creating process, provided the mailbox to receive the message still exists and the creating process has access to the mailbox. The termination message is sent before the final rundown is initiated; thus, the creating process might receive the message before the process deletion is complete.

Due to the complexity of the required rundown operations, a significant time interval occurs between a delete request and the actual deletion of the process. However, the $DELPRC service returns to the caller immediately after initiating the rundown operation.

If you issue subsequent delete requests for a process currently being deleted, the requests return immediately with a successful completion status.

Process exit handlers are not invoked when a process is deleted. For details on exit handlers, see the $DCLEXH service.

Required Access or Privileges

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

- GROUP privilege to delete processes in the same group that do not have the same UIC

- WORLD privilege to delete any process in the system

Required Quota

None. Deductible resource quotas granted to subprocesses are returned to the creating process when the subprocesses are deleted.

Related Services

$CANEXH, $CREPRC, $DCLEXH, $EXIT, $FORCEX, $GETJPI, $GETJPIW, $HIBER, $PROCESS_SCAN, $RESUME, $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$_INSFMEM

The system dynamic memory is insufficient for completing the operation.

SS$_NONEXPR

The specified process does not exist, or an invalid process identification was specified.

SS$_NOPRIV

The caller does not have the privilege to delete the specified process.

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.)

$DELTVA

Delete Virtual Address Space — Deletes a range of addresses from a process's virtual address space. Upon successful completion of the service, the deleted pages are inaccessible, and references to them cause access violations.

Format

```
SYS$DELTVA inadr ,[retadr] ,[acmode]
```

C Prototype

```
int sys$deltva
   (struct _va_range *inadr, struct _va_range *retadr,
    unsigned int 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 deleted. The *inadr* argument is the address of a 2-longword array containing, in order, the starting and the ending process virtual addresses. If the starting and ending virtual addresses are the same, a single page is deleted. 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.

The $DELTVA service deletes pages starting at the address contained in the second longword of the *inadr* argument and ending at the address in the first longword. Thus, if you use the same address array for both the Create Virtual Address Space ($CRETVA) and the $DELTVA services, the pages are deleted in the reverse order from which they were created.

retadr

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

Starting and ending process virtual addresses of the pages that $DELTVA has deleted. The *retadr* argument is the address of a 2-longword array containing, in order, the starting and ending process virtual addresses.

acmode

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

Access mode on behalf of which the service is to be performed. The *acmode* argument is a longword containing the access mode.

The most privileged access mode used is the access mode of the caller. The calling process can delete pages only if those pages are owned by an access mode equal to or less privileged than the access mode of the calling process.

Description

The Delete Virtual Address Space service deletes a range of addresses from a process's virtual address space. Upon successful completion of the service, the deleted pages are inaccessible, and references to them cause access violations. If any of the pages in the specified range have already been deleted or do not exist, the service continues as if the pages were successfully deleted.

If an error occurs while pages are being deleted, the *retadr* argument specifies the pages that were successfully deleted before the error occurred. If no pages are deleted, both longwords in the return address array contain the value –1.

Required Access or Privileges

None

Required Quota

None

Related Services

$ADJSTK, $ADJWSL, $CRETVA, $CRMPSC, $DGBLSC, $EXPREG, $LCKPAG, $LKWSET, $MGBLSC, $PURGWS, $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, or the return address array cannot be written by the caller.

SS$_NOPRIV

A page in the specified range is in the system address space.

SS$_PAGOWNVIO

A page in the specified range is owned by an access mode more privileged than the access mode of the caller.

SS$_NOSHPTS

The region ID of a shared page table region was specified.

SS$_VA_IN_USE

The existing underlying page cannot be deleted because it is associated with a buffer object.

$DELTVA_64 (Alpha and Integrity servers)

Delete Virtual Address Space — On Alpha and Integrity server systems, deletes a range of virtual addresses from a process's virtual address space. Upon successful completion of the service, the deleted pages are inaccessible, and references to them cause access violations. This service accepts 64-bit addresses.

Format

```
SYS$DELTVA_64
   region_id_64 ,start_va_64 ,length_64 ,acmode ,return_va_64
   ,return_length_64
```

C Prototype

```
int sys$deltva_64
   (struct _generic_64 *region_id_64, void *start_va_64,
    unsigned __int64 length_64, unsigned int acmode,
    void *(*(return_va_64)), unsigned __int64 *return_length_64);
```

Arguments

region_id_64

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

The region ID associated with the region from which to address the VA space.

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

Symbol	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. Also, the region ID that a virtual address is in can be obtained by calling the $GET_REGION_INFO service, specifying the VA$_REGSUM_BY_VA function.

start_va_64

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

The starting virtual address of the pages to be deleted. The specified address must be a CPU-specific page aligned address. If the *region_id_64* argument specifies a shared page table region or if the *start_va_64* argument lies within a shared page table region, the specified address must be a CPU-specific page table page aligned address.

length_64

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

Length of the virtual address space to be deleted. The length specified must be a multiple of CPU-specific pages. If the virtual address space is being deleted from a shared page table region, the specified length must be page table page aligned or include the last page in a memory-resident section.

acmode

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

Access mode associated with the call to $DELTVA_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

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. The calling process can delete pages only if those pages are owned by an access mode equal to or less privileged than the access mode of the calling process.

return_va_64

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

The lowest process virtual address of the deleted virtual address range. The *return_va_64* argument is the 32- or 64-bit virtual address of a naturally aligned quadword into which the $DELTVA_64 service returns the virtual address.

return_length_64

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

The 32- or 64-bit virtual address of a naturally aligned quadword into which the $DELTVA_64 service returns the length in bytes of the virtual address range deleted.

Description

The Delete Virtual Address Space service is a kernel mode service that can be called from any mode. This service deletes a range of addresses from a process's virtual address space. Upon successful completion of the service, the deleted pages are inaccessible, and references to them cause access violations. If any of the pages in the specified range have already been deleted or do not exist, the service continues as if the pages were successfully deleted.

If 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 deleted before the error occurred. If no pages were deleted, 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

$CREATE_REGION_64, $CRETVA_64, $CRMPSC_FILE_64, $CRMPSC_GFILE_64, $CRMPSC_GPFILE_64, $CRMPSC_GPFN_64, $CRMPSC_PFN_64, $DELETE_REGION_64, $EXPREG_64, $MGBLSC_64, $MGBLSC_GPFN_64

Condition Values Returned

SS$_NORMAL

The service completed successfully.

SS$_ACCVIO

The *return_va_64* argument or the *return_length_64* argument cannot be written by the caller.

SS$_IVREGID

Invalid region ID specified. This condition value is returned if P0, P1, or P2 space is specified because these regions cannot be deleted, or if no region exists for the specified ID.

SS$_LEN_NOTPAGMULT

The *length_64* argument is not a multiple of CPU-specific pages; or, for shared page table regions, is not a multiple of CPU-specific page table pages or does not include the last page in a memory-resident global section.

SS$_PAGNOTINREG

A page in the specified range is not within the specified region.

SS$_PAGOWNVIO

A page in the specified range is owned by an access mode more privileged than the access mode of the caller.

SS$_VA_NOTPAGALGN

The *start_va_64* argument is not a CPU-specific page table page aligned address; or, for shared page table regions, is not page table page aligned.

SS$_VA_IN_USE

The existing underlying page cannot be deleted because it is associated with a buffer object.

$DEQ

Dequeue Lock Request — Dequeues (unlocks) granted locks; dequeues the sublocks of a lock; or cancels an ungranted lock request. The calling process must have previously acquired the lock or queued

the lock request by calling the Enqueue Lock Request ($ENQ) service. On Alpha and Integrity server systems, this service accepts 64-bit addresses.

Format

```
SYS$DEQ [lkid] ,[valblk] ,[acmode] ,[flags]
```

C Prototype

```
int sys$deq
   (unsigned int lkid, void *valblk, unsigned int acmode,
    unsigned int flags);
```

Arguments

lkid

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

Lock identification of the lock to be dequeued. The *lkid* argument specifies this lock identification.

Note that if you do not specify the *lkid* argument, you must specify the LCK$M_DEQALL flag in the *flags* argument.

When you specify the LCK$M_DEQALL flag in the *flags* argument, different values (or no value) for the *lkid* argument produce varying behavior:

- When you do not specify the *lkid* argument (or specify it as 0) and you do specify the LCK$M_DEQALL flag, $DEQ dequeues all locks held by the process, at access modes equal to or less privileged than the effective access mode, on all resources. The effective access mode is the least privileged of the caller's access mode and the access mode specified in the *acmode* argument.

- When you specify the *lkid* argument as a nonzero value together with the LCK$M_DEQALL flag, $DEQ dequeues all sublocks of the lock identified by *lkid*; it does not dequeue the lock identified by *lkid*. For this operation, $DEQ ignores the LCK$M_CANCEL flag if it is set. A sublock of a lock is a lock that was created when the *parid* argument in the call to $ENQ was specified, where *parid* is the lock ID of the parent lock.

If you omit the *lkid* argument (or specify it as 0) and the LCK$M_DEQALL flag is not set, the $DEQ service returns the invalid lock ID condition value (SS$_IVLOCKID).

valblk

OpenVMS usage:	lock_value_block
type:	longword (unsigned)
access:	modify

mechanism:	by 32- or 64-bit reference

Lock value block for the resource associated with the lock to be dequeued. The *valblk* argument is the 32- or 64-bit address of the 16-byte lock value block, or, if LCK$M_XVALBLK is specified (on Alpha or Integrity server systems), of the 64-byte lock value block. When you specify the LCK$M_DEQALL flag, you cannot use this argument.

When a protected write (PW) or exclusive (EX) mode lock is being dequeued and you specify a lock value block in the *valblk* argument, the contents of that lock value block are written to the lock value block in the lock database. Further, if the lock value block in the lock database was marked as invalid, that condition is cleared; the block becomes valid.

acmode

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

Access mode of the lock to be dequeued. The *acmode* argument is a longword containing the access mode.

The *acmode* argument is valid only if the LCK$M_DEQALL flag of the *flags* argument is set. 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

When dequeuing locks, $DEQ maximizes the access mode of the caller and the specified *acmode* argument. The maximized access mode is the less privileged of the caller's access mode and the *acmode* argument. If you do not specify the *acmode* argument, $DEQ uses the caller's access mode. Only those locks with an access mode that is equal to or less than the maximized access mode are dequeued.

flags

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

Flags specifying options for the $DEQ operation. The *flags* argument is a longword bit mask that is the logical OR of each bit set, where each bit corresponds to an option.

Note that if you do not specify the *lkid* argument, you must specify the LCK$M_DEQALL flag in the *flags* argument.

A symbolic name for each flag bit is defined by the $LCKDEF macro. The following table describes each flag.

Flag	Description
LCK$M_DEQALL	When you specify this flag, $DEQ dequeues multiple locks, depending on the value of the *lkid* argument. Refer to the description of the *lkid* argument for details. The *acmode* argument is ignored if the LCK$M_DQALL flag is not set. If you specify LCK$M_DEQALL, the LCK$M_CANCEL flag, if set, is ignored.
LCK$M_CANCEL	When you specify this flag, $DEQ attempts to cancel a lock request that was queued by $ENQ. You can cancel only a waiting request. When the request is canceled, $DEQ returns the condition value SS$_NORMAL. If you attempt to cancel a granted lock, the request fails and $DEQ returns the condition value SS$_CANCELGRANT. There are two types of waiting requests that can be canceled: • A request for a new lock • A request to convert an existing lock When canceling a new lock request, the following action is taken: • If a completion asynchronous system trap (AST) was requested, the AST is queued for delivery and SS$_ABORT is stored in the lock status block. When canceling a request to convert an existing lock, the conversion request is canceled. The existing granted lock remains unchanged. The following specific actions are taken: • The blocking AST address specified for the existing granted lock is queued for delivery if the granted mode of the existing lock is blocking other waiting requests. • If a completion AST was specified by the conversion request, the completion AST is queued for delivery with SS$_CANCEL status stored in the lock status block that was specified by the conversion request. If you specify the LCK$M_DEQALL flag, the LCK$M_CANCEL flag is ignored.
LCK$M_INVVALBLK	When you specify this flag, $DEQ marks the lock value block, which is maintained for the resource in the lock database, as invalid. The lock value block remains marked as invalid until it is again written to. The Description section of the $ENQ service provides additional information about lock value block invalidation. This flag is ignored if (1) the lock mode of the lock being dequeued is not protected write or exclusive, or (2) you specify the LCK$M_CANCEL flag.
LCK$M_XVALBLK	When you specify this flag, you must provide a 64-byte lock value block as the *valblk* argument. If you do not specify this flag, only the

Flag	Description
	first 16 bytes of the buffer specified in the *valblk* argument will be written. If the value block is written without this flag, the value block will be flagged so that a future reader who specifies the LCK$M_XVALBLK flag in the $ENQ system service call will receive the warning status SS$_XVALNOTVALID until a future writer writes to the value block specifying this flag. This flag is valid only on Alpha and Integrity server systems.

Description

The Dequeue Lock Request service dequeues (unlocks) granted locks and waiting lock requests. The calling process must have previously acquired the lock or queued the lock request by calling the Enqueue Lock Request ($ENQ) service.

Action taken by the $DEQ service depends on the current state (granted or waiting) and the type of lock request (new lock or conversion request) to be dequeued.

When dequeuing a granted lock, the $DEQ service returns the condition value SS$_NORMAL and the following specific action is taken:

- Any queued blocking ASTs that have not been delivered are removed from the process's AST queues.

There are two types of waiting requests that can be dequeued:

- A request for a new lock
- A request to convert an existing lock

When dequeuing a new lock request, the $DEQ service returns the condition value SS$_NORMAL and the following specific action is taken:

- If a completion AST was requested, the completion AST is queued for delivery with SS$_ABORT stored in the lock status block.

When dequeuing a lock for which there is a conversion request waiting, the existing lock and its conversion request are dequeued. The $DEQ service returns the condition value SS$_NORMAL and the following specific actions are taken:

- If a blocking AST was queued to the process, it is removed from the process's AST queue.
- If a completion AST was specified by the conversion request, the completion AST is queued for delivery with SS$_ABORT status stored in the lock status block that was specified by the conversion request.

When a protected write (PW) or exclusive (EX) mode lock is being dequeued and you specify a lock value block in the *valblk* argument, the contents of that lock value block are written to the lock value block in the lock database.

If you specify the LCK$M_INVVALBLK flag in the *flags* argument and the lock mode of the lock being dequeued is PW or EX, the lock value block in the lock database is marked as invalid whether or not a lock value block was specified in the *valblk* argument.

The $DEQ, $ENQ, $ENQW, and $GETLKI services together provide the user interface to the lock management facility. For additional information about lock management, refer to the descriptions of these other services and to the *VSI OpenVMS Programming Concepts Manual*.

Required Access or Privileges

None

Required Quota

None

Related Services

$ENQ, $ENQW, $GETLKI, $GETLKIW

Condition Values Returned

SS$_NORMAL

The lock was dequeued successfully.

SS$_ACCVIO

The value block specified by the *valblk* argument cannot be accessed by the caller.

SS$_CANCELGRANT

The LCK$M_CANCEL flag in the *flags* argument was specified, but the lock request that $DEQ was to cancel had already been granted.

SS$_ILLRSDM

An illegal attempt to modify a value block was made.

SS$_IVLOCKID

An invalid or nonexistent lock identification was specified or the process does not have the privilege to dequeue a lock at the specified access mode.

SS$_SUBLOCKS

The lock has sublocks and cannot be dequeued.

$DEVICE_PATH_SCAN (Alpha and Integrity servers)

Scan for Device Paths — On Alpha and Integrity server systems, returns the displayable pathname for a given I/O channel or device name. Can be used to return all displayable paths to an I/O device.

Format

```
SYS$DEVICE_PATH_SCAN [chan] [,devnam] ,itmlst [,contxt] [,nullarg]
```

C Prototype

```
int sys$device_path_scan
   (unsigned short int chan, void *devnam, void *itmlst,
    unsigned int *contxt, struct_generic_64 *nullarg);
```

Arguments

chan

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

Number of the I/O channel assigned to the device about which information is desired. The *chan* argument is a word containing this number.

To identify a device to $DEVICE_PATH_SCAN, you can specify either the *chan* or *devnam* parameters, but you should not specify both. If you specify both arguments, the *chan* argument is used.

If you specify neither *chan* nor *devnam*, $DEVICE_PATH_SCAN uses a default value of 0 for *chan*.

devnam

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

The name of the device about which $DEVICE_PATH_SCAN is to return path information. The *devnam* argument is the address of a character string descriptor pointing to this name string.

The device name string can be either a physical device name or a logical name. If the first character in the string is an underscore (_), the string is considered a physical device name; otherwise, the string is considered a logical name and logical name translation is performed until either a physical device name is found or the system default number of translations has been performed.

If the device name string contains a colon (:), the colon and the characters that follow it are ignored.

To identify a device to $DEVICE_PATH_SCAN, you can specify either the *chan* or *devnam* argument, but you should not specify both. If both arguments are specified, the *chan* argument is used.

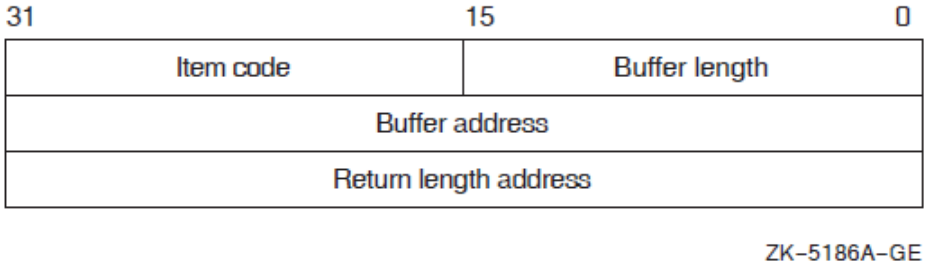
If you specify neither *chan* nor *devnam*, $DEVICE_PATH_SCAN uses a default value of 0 for *chan*.

itmlst

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

mechanism:	by reference

Item list specifying which information about the device is to be returned. The *itmlst* argument is the address of a list of item descriptors, each of which describes an item of information. The list of item descriptors is terminated by a longword of 0. The following diagram depicts the format of a single item descriptor:



See the *itmlst* argument in the $GETDVI system service description for information on the meaning of these fields in the item list.

contxt

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

Value used to indicate the current position of a $DEVICE_PATH_SCAN search. The *contxt* argument is the address of the longword that receives this information. On the initial call, the longword should contain 0.

nullarg

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

Placeholding argument reserved to OpenVMS.

Item Code

DPS$_MP_PATHNAME

When you specify DPS$_MP_PATHNAME, $DEVICE_PATH_SCAN returns the name of one of the Multipath I/O paths connecting to the device named in the *devnam* argument. When the value of the *contxt* argument is 0, the path name for the first established path will be returned. On subsequent calls, with a non-zero *contxt* value, the path names of the remaining available paths to the device will be returned.

In the item code, the Buffer Address field must point to the buffer that will hold the path name to be returned by the service. The Return Length Address field must be point to the buffer that will hold the return length returned by the service.

Upon completion of the command, the buffer pointed to by the Buffer Address field will hold a string identifying the requested path name. The Return Length Address field will point to the length in bytes of the path name being returned. The bytes in the path name buffer beyond the end of the path string will remain in the state they were set by the caller of the service.

The DPSDEF macro contains this item code.

Description

The Scan for Device Paths service returns I/O path information for a given I/O channel or device name. Each call to $DEVICE_PATH_SCAN will return information on a different I/O path connecting with the device specified in the *chan* or *devnam* arguments.

If the *contxt* argument is handled appropriately, the service will return information on the paths in the order in which they were established. On the first call, the *contxt* argument should be set to zero. The *contxt* value will be changed by the service during this call and a new value will be written into *contxt* and returned to the caller. The caller must use this same value in the next call to the service. Following this convention will result in a different path name being returned on each call.

Once the service has returned information on all paths to the named device, any further calls that use the final *contxt* value will result in SS$_NOMOREPATH status being returned.

Required Access or Privileges

None

Required Quota

None

Related Services

$ASSIGN, $DASSGN, $DEVICE_SCAN, $GETDVI, $GETDVIW, $SET_DEVICE

Condition Values Returned

SS$_NORMAL

The service completed successfully.

SS$_ACCVIO

The device name string descriptor, device name string, or *itmlst* argument cannot be read; or the buffer or return length longword cannot be written.

SS$_BADPARAM

The item list contains an invalid item code, or the buffer length field in an item descriptor specified insufficient space for the return length information.

SS$_IVCHAN

You specified an invalid channel number, that is, a channel number larger than the number of channels.

SS$_IVDEVNAM

The device name string contains invalid characters, or neither the *devnam* nor *chan* argument was specified.

SS$_NOPRIV

The specified channel is not assigned or was assigned from a more privileged access mode.

SS$_NOMOREPATH

No more device paths exist for this device.

SS$_NOSUCHDEV

The specified device does not exist on the host system.

$DEVICE_SCAN

Scan for Devices — Returns the names of all devices that match a specified set of search criteria.

Format

```
SYS$DEVICE_SCAN return_devnam ,retlen ,[search_devnam] ,[itmlst] ,[contxt]
```

C Prototype

```
int sys$device_scan
   (void *return_devnam, unsigned short int *retlen, void *search_devnam,
    void *itmlst, struct _generic_64 *contxt);
```

Arguments

return_devnam

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

Buffer to receive the device name. The *return_devnam* argument is the address of a character string descriptor pointing to a buffer into which $DEVICE_SCAN writes the name of the first or next device that matches the specified search criteria. The maximum size of any device name is 64 bytes.

retlen

OpenVMS usage:	word_unsigned
type:	word (unsigned)
access:	write only

mechanism:	by reference

Length of the device name string returned by $DEVICE_SCAN. The *retlen* argument is the address of a word into which $DEVICE_SCAN writes the length of the device name string.

search_devnam

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

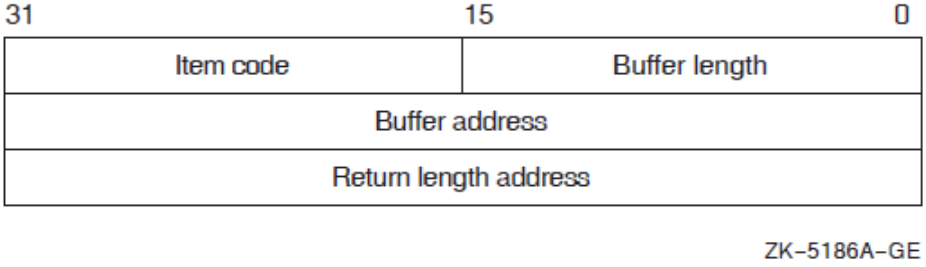
Name of the device for which $DEVICE_SCAN is to search. The *search_devnam* argument accepts the standard wildcard characters, the asterisk (*), which matches any sequence of characters, and the percent sign (%), which matches any one character. If the *search_devnam* argument does not include a wildcard character, an exact match is used for comparison. For example, to match all unit 0 *DU* devices on any controller, specify **DU%0*. This string is compared to the most complete device name (DVI$_ALLDEVNAM). Only uppercase characters are accepted.

itmlst

OpenVMS usage:	item_list_3
type:	longword_unsigned
access:	read only
mechanism:	by reference

Item list specifying search criteria used to identify the device names for return by $DEVICE_SCAN. The *itmlst* argument is the address of a list of item descriptors, each of which describes one search criterion. The list of item descriptors is terminated by a longword of 0.

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



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 from which $DEVICE_SCAN 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.
Item code	A word containing a user-specified symbolic code specifying the item of information that $DEVICE_SCAN is to return. The $DVSDEF macro defines these codes. Each item code is described in the Item Codes section.

Descriptor Field	Definition
Buffer address	A longword containing the address of a longword value that contains item code information. Examples include DC$_DISK and DC$_MAILBOX.
Return length address	A longword containing the address of a word to receive the length (in bytes) of information returned for the output value item code. For the input value item code, this field is not used. VSI recommends the placeholder value be 0.

contxt

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

Value used to indicate the current position of a $DEVICE_SCAN search. The *contxt* argument is the address of the quadword that receives this information. On the initial call, the quadword should contain 0.

Item Codes

DVS$_DEVCLASS

An input value item code that specifies, as an unsigned longword, the device class being searched. The $DCDEF macro defines these classes.

The DVS$_DEVCLASS argument is a longword containing this number; however, DVS$_DEVCLASS uses only the low-order byte of the longword.

DVS$_DEVTYPE

An input value item code that specifies, as an unsigned longword, the device type for which $DEVICE_SCAN is going to search. The $DCDEF macro defines these types.

The DVS$_DEVTYPE argument is a longword containing this number; however, DVS$_DEVTYPE uses only the low-order byte of the longword. DVS$_DEVTYPE should be used in conjunction with $DVS_DEVCLASS to specify the device type being searched for.

Description

The Scan for Devices system service returns the names of all devices that match a specified set of search criteria. The names returned by $DEVICE_SCAN can then be passed to another service; for example, $GETDVI or $MOUNT.

The device names are returned for one process per call. A context value is used to continue multiple calls to $DEVICE_SCAN.

$DEVICE_SCAN allows wildcard searches based on device names, device classes, and device types. It also provides the ability to perform a wildcard search on other device-related services.

$DEVICE_SCAN makes it possible to combine search criteria. For example, to find only RA82 devices, use the following selection criteria:

```
DVS$_DEVCLASS = DC$_DISK and DVS$_DEVTYPE = DT$_RA82
```

To find all mailboxes with *MB* as part of the device name (excluding mailboxes such as NLA0), use the following selection criteria:

```
DVS$_DEVCLASS = DC$_MAILBOX and DEVNAM = *MB*
```

Required Access or Privileges

None

Required Quota

None

Related Services

$ALLOC, $ASSIGN, $BRKTHRU, $BRKTHRUW, $CANCEL, $CREMBX, $DALLOC, $DASSGN, $DELMBX, $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 *search_devnam*, *itmlst*, or *contxt* argument cannot be read by the caller, or the *retlen*, *return_devnam*, or *contxt* argument cannot be written by the caller.

SS$_BADPARAM

The *contxt* argument contains an invalid value, or the item list contains an invalid item code.

SS$_NOMOREDEV

No more devices match the specified search criteria.

SS$_NOSUCHDEV

The specified device does not exist on the host system.

$DGBLSC

Delete Global Section — Marks an existing permanent global section for deletion. The actual deletion of the global section takes place when all processes that have mapped the global section have deleted the mapped pages. On Alpha and Integrity server systems, this service accepts 64-bit addresses.

Format

```
SYS$DGBLSC [flags] ,gsdnam ,[ident]
```

C Prototype

```
int sys$dgblsc (unsigned int flags, void *gsdnam, struct _secid *ident);
```

Arguments

flags

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

Mask indicating global section characteristics. The *flags* argument is a longword value. A value of 0 (the default) specifies a group global section; a value of SEC$M_SYSGBL specifies a system global section; a value of SEC$M_SHMGS on an OpenVMS Galaxy system creates a shared-memory global section.

gsdnam

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

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

For group global sections, 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.

You can specify any name from 1 to 43 characters. All processes mapping to the same global section must specify the same name. Note that the name is case sensitive.

Use of characters valid in logical names is strongly encouraged. Valid values include alphanumeric characters, the dollar sign ($), and the underscore (_). If the name string begins with an underscore (_), the underscore is stripped and the resultant string is considered to be the actual name. Use of the colon (:) is not permitted.

Names are first subject to a logical name translation, after the application of the prefix GBL$ to the name. If the result translates, it is used as the name of the section. If the resulting name does not translate, the name specified by the caller is used as the name of the section.

Additional information on logical name translations and on section name processing is available in the *VSI OpenVMS Programming Concepts Manual*.

ident

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

Identification value specifying the version number of the global section to be deleted and the matching criteria to be applied. The *ident* argument is the 32- or 64-bit address of a quadword structure containing three fields.

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

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

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

If you specify no address or specify it as 0 (the default), the version number and match control fields default to 0.

Description

The Delete Global Section service marks an existing permanent global section for deletion. The actual deletion of the global section takes place when all processes that have mapped the global section have deleted the mapped pages.

After a global section has been marked for deletion, any process that attempts to map it receives the warning return status code SS$_NOSUCHSEC.

Temporary global sections are automatically deleted when the count of processes using the section goes to 0.

Required Access or Privileges

Depending on the operation, the calling process might need one or more of the following privileges:

- SYSGBL privilege to delete a system global section
- PRMGBL privilege to delete a permanent global section
- PFNMAP privilege to delete a page frame section

- SHMEM privilege to delete a global section located in memory shared by multiple processors

Required Quota

None

Related Services

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

The $DGBLSC service does not unmap a global section from a process's virtual address space. To do this, the process should call the Delete Virtual Address Space ($DELTVA or $DELTVA_64) service, which deletes the pages to which the section is mapped.

Condition Values Returned

SS$_NORMAL

The service completed successfully.

SS$_ACCVIO

The global section name or name descriptor or the section identification field cannot be read by the caller.

SS$_INTERLOCK

The bit map lock for allocating global sections from the specified shared memory is locked by another process.

SS$_IVLOGNAM

The global section name has a length of 0 or has more than 15 characters.

SS$_IVSECFLG

You set an invalid flag, reserved flag, or flag requiring a user privilege.

SS$_IVSECIDCTL

The section identification match control field is invalid.

SS$_NOPRIV

The caller does not have the privilege to delete a system global section, does not have read/write access to a group global section, or does not have the privilege to delete a global section located in memory that is shared by multiple processors.

SS$_NOSUCHSEC

The specified global section does not exist, or the identifications do not match.

SS$_NOTCREATOR

The section is in memory shared by multiple processors and was created by a process on another processor.

SS$_TOOMANYLNAM

The logical name translation of the *gsdnam* string exceeded the allowed depth of 10.

$DISCONNECT

Breaks Connection — The Disconnect service breaks the connection between a RAB and a FAB, thereby terminating a record stream. All system resources, such as I/O buffers and data structure space, are deallocated. For additional information about this service, see the *VSI OpenVMS Record Management Services Reference Manual*.

$DISMOU

Dismount Volume — Dismounts a mounted volume or volume sets.

Format

```
SYS$DISMOU devnam ,[flags]
```

C Prototype

```
int sys$dismou (void *devnam, unsigned int flags);
```

Arguments

devnam

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

Device name of the device to be dismounted. The *devnam* argument is the address of a character string descriptor pointing to the device name string. The string can be either a physical device name or a logical name. If it is a logical name, it must translate to a physical device name.

flags

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

A longword bit vector specifying options for the dismount operation. The *flags* argument is a longword bit vector wherein a bit, when set, selects the corresponding option. Each bit has a symbolic name; these names are defined by the $DMTDEF macro. The flags and their meanings are listed in the following table.

Flag	Meaning
DMT$M_ABORT	The volume is to be dismounted even if the caller did not mount the volume. If the volume was mounted with MNT$M_SHARE specified, $DISMOU dismounts the volume for all of the users who mounted it. To specify DMT$M_ABORT, the caller must: (1) have GRPNAM privilege for a group volume, (2) have SYSNAM privilege for a system volume, or (3) either own the volume or have VOLPRO privilege.
DMT$M_CLUSTER	The volume is to be dismounted clusterwide, that is, from all nodes in the OpenVMS Cluster system. $DISMOU dismounts the volume from the caller's node first and then from every other node in the existing cluster. DMT$M_CLUSTER dismounts only system or group volumes. To dismount a group volume clusterwide, the caller must have GRPNAM privilege. To dismount a system volume clusterwide, the caller must have SYSNAM privilege. DMT$M_CLUSTER has no effect if the system is not a member of a cluster. DMT$M_CLUSTER applies only to disks.
DMT$M_FORCE	If connectivity to a device has been lost and the shadow set is in mount verification, this flag causes a named shadow set member to be immediately expelled from the shadow set.
DMT$M_MINICOPY_OPTIONAL	$DISMOU takes place, regardless of whether minicopy is enabled on the disk.
DMT$M_MINICOPY_REQUIRED	$DISMOU fails if minicopy has not been enabled on the disk.
DMT$M_NOUNLOAD	Specifies that the volume is not to be physically unloaded after the dismount. If both the DMT$M_UNLOAD and DMT$M_NOUNLOAD flags are specified, the DMT$M_NOUNLOAD flag is ignored. If neither flag is specified, the volume is physically unloaded, unless the DMT$M_NOUNLOAD flag was specified on the $MOUNT system service or the /NOUNLOAD qualifier was specified on the MOUNT command when the volume was mounted.
DMT$M_OVR_CHECKS	Specifies that the volume should be dismounted without checking for open files, spooled devices, installed images, or installed swap and page files.
DMT$M_UNIT	The specified device, rather than the entire volume set, is dismounted.
DMT$M_UNLOAD	Specifies that the volume is to be physically unloaded after the dismount. If both the DMT$M_UNLOAD and DMT$M_NOUNLOAD flags are specified, the

Flag	Meaning
	DMT$M_NOUNLOAD flag is ignored. If neither flag is specified, the volume is physically unloaded, unless the DMT$M_NOUNLOAD flag was specified on the $MOUNT system service or the /NOUNLOAD qualifier was specified on the MOUNT command when the volume was mounted.

Description

The Dismount Volume service dismounts a mounted volume or volume sets. To dismount a private volume, the caller must own the volume.

When you issue the $DISMOU service, $DISMOU removes the volume from your list of mounted volumes, deletes the logical name (if any) associated with the volume, and decrements the mount count.

If the mount count does not equal 0 after being decremented, $DISMOU does not mark the volume for dismounting (because the volume must have been mounted shared). In this case, the total effect for the issuing process is that the process is denied access to the volume and a logical name entry is deleted.

If the mount count equals 0 after being decremented, $DISMOU marks the volume for dismounting. After marking the volume for dismounting, $DISMOU waits until the volume is idle before dismounting it. A native volume is idle when no user has an open file to the volume, and a foreign volume is idle when no channels are assigned to the volume.

Native volumes are Files-11 structured disks or ANSI-structured tapes. Foreign volumes are not Files-11 or ANSI structured media.

After a volume is dismounted, nonpaged pool is returned to the system. Paged pool is also returned if you mounted the volume using the /GROUP or /SYSTEM qualifier.

If a volume is part of a Files-11 volume set and the flag bit DMT$V_UNIT is not set, the entire volume set is dismounted.

When a Files-11 volume has been marked for dismount, new channels can be assigned to the volume, but no new files can be opened.

Note that the SS$_NORMAL status code indicates only that $DISMOU has successfully performed one or more of the actions just described: decremented the mount count, marked the volume for dismount, or dismounted the volume. The only way to determine that the dismount has actually occurred is to check the device characteristics using the Get Device/Volume Information ($GETDVI) service.

By specifying the DVI$_DEVCHAR item code in a call to $GETDVI, you can learn whether a volume is mounted (it is if the DEV$V_MNT bit is set) or whether it is marked for dismounting (it is if the DEV$M_DMT bit is set). If DEV$V_MNT is clear or if DEV$M_DMT is set, the mount count is 0.

Required Access or Privileges

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

- GRPNAM privilege to dismount a volume mounted with the /GROUP qualifier
- SYSNAM privilege to dismount a volume mounted with the /SYSTEM qualifier

Required Quota

None

Related Services

$ALLOC, $ASSIGN, $BRKTHRU, $BRKTHRUW, $CANCEL, $CREMBX, $DALLOC, $DASSGN, $DELMBX, $DEVICE_SCAN, $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 device name descriptor cannot be read or does not describe a readable device name.

SS$_DEVALLOC

The device is allocated to another process and cannot be dismounted by the caller.

SS$_DEVOFFLINE

The specified device is not available.

SS$_DEVNOTMOUNT

The specified device is not mounted.

SS$_IVDEVNAM

The device name string is not valid.

SS$_IVLOGNAM

The device logical name has a length of 0 or is longer than the allowable logical name length.

SS$_NOGRPNAM

GRPNAM privilege is required to dismount a volume mounted for groupwide access.

SS$_NOIOCHAN

No I/O channel is available. To use $DISMOU, a channel must be assigned to the volume.

SS$_NONLOCAL

The device is on a remote node.

SS$_NOSUCHDEV

The specified device does not exist.

SS$_NOSYSNAM

SYSNAM privilege is required to dismount a volume mounted for systemwide access.

SS$_NOTFILEDEV

The specified device is not file structured.

$DISPLAY

Retrieves File Attribute Information — The Display service retrieves file attribute information about a file and places this information in fields in the FAB, in XABs chained to the FAB, and in a NAM block (if one is requested). For additional information about this service, see the *VSI OpenVMS Record Management Services Reference Manual*.

$DISPLAY_PROXY

Display Proxy Information — Returns information about one or more existing proxies.

Format

```
SYS$DISPLAY_PROXY
   rem_node ,rem_user ,buffer_sizes ,proxy_node ,proxy_user ,default_user
   ,local_users ,flags ,[context]
```

C Prototype

```
int sys$display_proxy
   (void *rem_node, void *rem_user, unsigned short int buffer_sizes [4],
    void *proxy_node, void *proxy_user, void *default_user,
    unsigned int *local_users, unsigned int flags, unsigned int *context);
```

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 about which information is being requested. 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 full name unless the PRX$M_BYPASS_EXPAND flag is set with the *flags* argument.

Asterisk (*) and percent sign (%) wildcards are allowed for the remote node specification. If you specify wildcards for the *rem_node* argument, the server searches the entire proxy database for matches to the

remote node and remote user you specified. If a match is found, information about the matched proxy is returned. See the Description section for additional details on retrieving information about multiple proxies.

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 about which information is being requested. 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.

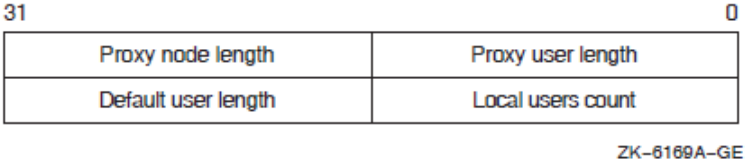
The *rem_user* argument can be specified in user identification code (UIC) format ([*group*, *member*]). Brackets are allowed only if the remote user name string specifies a UIC. Group and member are character-string representations of octal numbers with no leading zeros.

Asterisk (*) and percent sign (%) wildcards are allowed for the remote user specification. If you specify wildcards for the *rem_user* argument, the server searches the entire proxy database for matches to the remote node and remote user you specified. If a match is found, information about the matched proxy is returned. See the Description section for information about retrieving information about multiple proxies.

buffer_sizes

OpenVMS usage:	return length block
type:	array of 4 words (unsigned)
access:	write only
mechanism:	by reference

Array of return lengths for various input buffers. The *buffer_sizes* argument is the address of an array of four words with the following format.



The following table defines the *buffer_sizes* fields.

Descriptor Field	Definition
Proxy user length	Return length (in bytes) of the *rem_user* argument. The proxy user length field contains a value in the range of 0 to 32. A value of 0 in this field indicates that the service has failed or that there was no match for the user specified by the *rem_user* argument.

Descriptor Field	Definition
Proxy node length	Return length (in bytes) of the *rem_node* argument. A value of 0 in this field indicates that the service has failed or that there was no match for the node specified by the *rem_node* argument. The proxy node length field contains values in the range of 0 to 1024.
Local users count	Number of local users associated with the matched proxy. The local users count field contains a value in the range of 0 to 16. A value of 0 indicates that the matched proxy had no local users.
Default user length	Return length (in bytes) of the *default_user* argument. The default user length field contains a value in the range of 0 to 32. A value of 0 in this field indicates that the matched proxy did not have a default user.

proxy_node

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

Node name of a proxy matching the remote node name specified by the *rem_node* argument and the remote user name specified by the *rem_user* argument. The *proxy_node* argument is the address of a character-string descriptor pointing to a buffer to receive the proxy node name.

The descriptor's buffer must be 1024 bytes long to receive a node name. The length of the returned node name is specified by the proxy node length field returned in the buffer specified by the *buffer_sizes* argument.

proxy_user

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

User name of a proxy matching the remote node name specified by the *rem_node* argument and the remote user name specified by the *rem_user* argument. The *proxy_user* argument is a character-string descriptor pointing to a buffer to receive the remote user name of a proxy.

The descriptor's buffer must be 32 bytes long to receive a user name. The length of the returned user name is specified by the proxy user length field returned in the buffer specified by the *buffer_sizes* argument.

default_user

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

Default user of a proxy matching the node name specified by the *rem_node* argument and the remote user name specified by the *rem_user* argument. The *default_user* argument is the address of a character-string descriptor pointing to a buffer to receive the default user name.

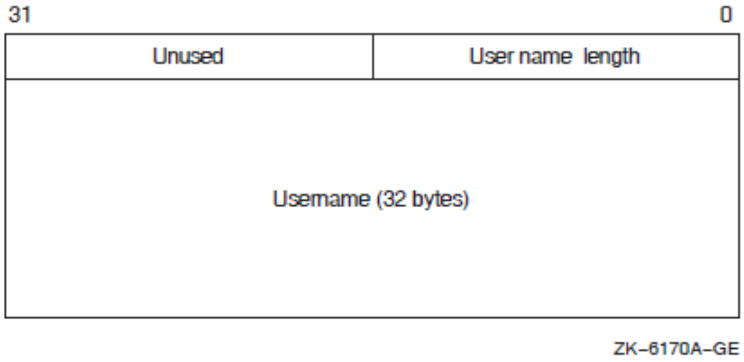
The descriptor's buffer must be 32 bytes long to receive a user name. The length of the returned user name is specified in the default user length field in the buffer specified by the *buffer_sizes* argument.

local_users

OpenVMS usage:	buffer
type:	array of 0 to 16 user name buffers
access:	write only
mechanism:	by reference

Array of local user names associated with a proxy matching the remote node name specified by the *rem_node* argument and the remote user name specified by the *rem_user* argument. The *local_users* argument is the address of a buffer to receive an array of local user names.

Each element in the array is a 36-byte block with the following format.



The following table defines the *local_users* fields.

Descriptor Field	Definition
User name length	Length (in bytes) of the associated user name string. The length can be in the range of 1 to 32 bytes.
Username	A fixed 32-byte blank padded character string containing a local user name associated with the matched proxy.

The buffer specified by the *local_users* argument must be able to contain up to 16 user name buffers. Therefore, the buffer length must be 576 bytes.

The number of elements returned in the buffer is specified in the local users count field returned in the buffer specified by the *buffer_sizes* 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$M_BYPASS_EXPAND	The service should not convert the node name specified in the *rem_node* argument to its corresponding DECnet full name. If this flag is set, it is the caller's responsibility to ensure that the fully expanded node name is passed into the service.
PRX$M_EXACT	The service should match exactly the remote node and remote user and ignore wildcards.

context

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

Context information to keep between related calls to the $DISPLAY_PROXY service. The *context* argument is the address of a longword to receive a context from the $DISPLAY_PROXY service.

The initial value contained in the 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 buffer pointed to by the *context* argument are changed between calls to the $DISPLAY_PROXY service, the service will return SS$_BADCONTEXT. If the contents of the *context* argument are changed between calls to the $DISPLAY_PROXY service, you can change the value of the *context* argument back to 0 to start the search over again.

Contexts become invalid after one-half hour of non-use. This means that if you call the $DISPLAY_PROXY service with a wildcard *rem_node* or *rem_user*, 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 start your search of the proxy database over from its beginning by resetting the context to 0.

Description

The Display Proxy Information service returns to the caller all information about a specified proxy in the proxy database.

Wildcards can be specified for the *rem_node* and *rem_user* arguments. Because $DISPLAY_PROXY can return information about only one matching proxy at a time, you must call this service repeatedly with the *context* argument to retrieve information about all matching proxies. $DISPLAY_PROXY returns SS$_NOMOREITEMS when information about all of the matching proxies has been returned. No proxy information is returned from the call that returns the SS$_NOMOREITEMS status.

Required Access or Privileges

The caller must have SYSPRV privilege or a UIC group less than or equal to the MAXSYSGRP system parameter.

Required Quota

None

Related Services

$ADD_PROXY, $DELETE_PROXY, $VERIFY_PROXY

Condition Values Returned

SS$_NORMAL

The service completed successfully.

SS$_ACCVIO

The *rem_node* or *rem_user* argument cannot be read by the service; or the *buffer_sizes*, *proxy_node*, *proxy_user*, *default_user*, or *local_users* argument cannot be written by the service; or the *context* argument cannot be read or written by the service.

SS$_BADBUFLEN

The length of the *rem_node*, *rem_user*, *proxy_node*, *proxy_user*, *default_user*, or *local_users* argument was out of range.

SS$_BADCONTEXT

The *context* argument did not contain a 0 on the first call to the service, or the *context* argument's value changed between consecutive calls to the service.

SS$_NOMOREITEMS

Information about all proxies matching the specification of the *rem_node* and *rem_user* arguments has been returned by the service.

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$_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

The specified local user does not exist in the proxy's local user list, or is not the proxy's default user.

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.

$DLCEFC

Delete Common Event Flag Cluster — Marks a permanent common event flag cluster for deletion.

Format

```
SYS$DLCEFC name
```

C Prototype

```
int sys$dlcefc (void *name);
```

Argument

name

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

Name of the common event flag cluster to be deleted. The *name* argument is the address of a character string descriptor pointing to the name of the cluster.

The names of event flag clusters are unique to UIC groups, and the UIC group number of the calling process is part of the name.

Description

The Delete Common Event Flag Cluster service marks a permanent common event flag cluster for deletion. The cluster is actually deleted when no more processes are associated with it. The $DLCEFC service does not disassociate a process from a common event flag cluster; the Disassociate Common Event Flag Cluster ($DACEFC) service does this. However, the system disassociates a process from an event flag cluster at image exit.

If the cluster has already been deleted or does not exist, the $DLCEFC service returns the status code SS$_NORMAL.

Required Access or Privileges

Delete access is required.

Required Quota

None

Related Services

$ASCEFC, $CLREF, $DACEFC, $READEF, $SETEF, $WAITFR, $WFLAND, $WFLOR

Condition Values Returned

SS$_NORMAL

The service completed successfully.

SS$_IVLOGNAM

The cluster name string has a length of 0 or has more than 15 characters.

SS$_NOPRIV

The process does not have the privilege to delete a permanent common event flag cluster, or the process does not have the privilege to delete a common event flag cluster in memory shared by multiple processors.

$END_BRANCH

End Branch — Removes a branch from a transaction and returns the outcome of the transaction.

Format

```
SYS$END_BRANCH [efn] ,[flags] ,iosb ,[astadr] ,[astprm] ,tid ,bid
```

C Prototype

```
int sys$end_branch
   (unsigned int efn, unsigned int flags, struct _iosb *iosb,
    void (*astadr)(__unknown_params), int astprm, unsigned int tid [4],
    unsigned int bid [4]);
```

Arguments

efn

OpenVMS usage:	ef_number
type:	longword (unsigned)

access:	read only
mechanism:	by value

Number of the event flag set when the service completes. If this argument is omitted, event flag 0 is used.

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, described in Table 26. All undefined bits must be 0. If this argument is omitted, no flags are used.

Table 26. $END_BRANCH Option Flags

Flag Name	Description
DDTM$M_SYNC	Specifies successful synchronous completion by returning SS$_SYNCH. When SS$_SYNCH is returned, the AST routine is not called, the event flag is not set, and the I/O status block is not filled in.
DDTM$M_NOWAIT	Indicates that the service should return to the caller without waiting for final cleanup. Note that $END_BRANCHW with the DDTM$M_NOWAIT flag set is not equivalent to $END_BRANCH. The latter returns when the operation has been queued. The former does not return until the operation has been initiated. The full range of status values may be returned from a nowait call.

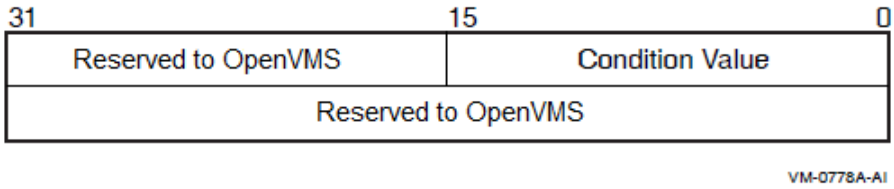
iosb

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

The I/O status block in which the following information is returned:

- The completion status of the service. This is returned as a condition value. See the Condition Values Returned section for more information.
- The outcome of the transaction. If the service completes successfully, the outcome of the transaction is commit. If it returns SS$_ABORT, the outcome of the transaction is abort.
- An abort reason code that gives one reason why the transaction aborted, if the completion status of the service is SS$_ABORT. The $DDTMMSGDEF macro defines symbolic names for these abort reason codes. See $ACK_EVENT for a list of the codes that are currently defined.

The following diagram shows the structure of the I/O status block:



astadr

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

The AST routine executed when the service completes, if SS$_NORMAL is returned in R0. The `astadr` argument is the address of the entry mask of this routine. The routine is executed in the same access mode as that of the caller of the $END_BRANCH service.

astprm

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

The AST parameter passed to the AST routine specified by the `astadr` argument.

tid

OpenVMS usage:	trans_id
type:	octaword (unsigned)
access:	read only
mechanism:	by reference

The identifier (TID) of the transaction from which the branch is to be removed.

bid

OpenVMS usage:	branch_id
type:	octaword (unsigned)
access:	read only
mechanism:	by reference

The identifier (BID) of the branch to be removed from the transaction.

Description

The $END_BRANCH system service:

- Removes the specified branch from the specified transaction.
- Returns the outcome of the specified transaction.

If $END_BRANCH completes successfully, the outcome of the transaction is commit. If it returns SS$_ABORT, the outcome is abort.

Preconditions for the successful completion of $END_BRANCH are:

- The calling process must contain the specified branch of the specified transaction.
- The specified branch must be a synchronized branch.
- The access mode of the caller must be the same as or more privileged than that of any branch of the specified transaction in this process. (See $START_BRANCH and $START_TRANS.)

$END_BRANCH may fail for the following reasons:

- Preconditions were not met.
- An abort event has occurred for the transaction.

Postconditions on successful completion of $END_BRANCH are described in Table 27:

Table 27. Postconditions When $END_BRANCH Completes Successfully

Postcondition	Meaning
The branch that started the transaction has initiated a call to $END_TRANS.	The completion of $END_BRANCH is delayed until this occurs. If the transaction was not started on the local node, the successful completion of $END_BRANCH may be indefinitely postponed by network failure.
Every other authorized and synchronized branch of the transaction has initiated a call to $END_BRANCH.	The completion of $END_BRANCH is delayed until this occurs.
The transaction is ended.	The result is that: • The TID of the transaction is invalid. Calls to any DECdtm system services except $GETDTI and $SETDTI that pass that TID will fail, and calls to resource managers that pass that TID will fail. • The transaction no longer has any application or RM participants on the local node. • All communications about the transaction between the local DECdtm transaction manager and other DECdtm transaction managers are finished (including the final "cleanup" acknowledgments).
The outcome of the transaction is commit.	All the transaction operations that completed successfully before $END_TRANS was called will take effect; that is, the effects of these operations will be made permanent. Operations by any unauthorized branches will be aborted. (An unauthorized branch is one without a matching $ADD_BRANCH.)

Postcondition	Meaning
DECdtm quotas are returned.	All quotas allocated for the transaction by calls on the local node to DECdtm services are now returned.
The transaction is not the default transaction of the calling process.	If the transaction was the default transaction of the calling process, then it is now no longer the default.

Postconditions on completion with the SS$_ABORT error are listed in Table 28. $END_BRANCH does not complete with this error until all branches on the local node have been removed from the transaction. Thus this call to $END_BRANCH cannot complete with the SS$_ABORT error until after every authorized and synchronized branch on the local node has initiated a call to $END_TRANS, $END_BRANCH, or $ABORT_TRANS.

Table 28. Postconditions When $END_BRANCH Completes with the SS$_ABORT Error

Postcondition	Meaning
The transaction is ended.	If DDTM$M_NOWAIT is clear: • The TID of the transaction is invalid. Calls to any DECdtm system services except $GETDTI and $SETDTI that pass that TID will fail, and calls to resource managers that pass that TID will fail. • The transaction no longer has any application or RM participants on the local node. • All communications about the transaction between the local DECdtm transaction manager and other DECdtm transaction managers are finished (including the final cleanup acknowledgments).
The outcome of the transaction is abort.	None of the operations of the transaction will ever take effect. The I/O status block contains one reason why the transaction was aborted. If there are multiple reasons for the transaction aborting, the DECdtm transaction manager returns one of the reasons in the I/O status block. It may return different reasons to different branches in the transaction. For example, if the transaction timeout expires and a communications link fails, then either the DDTM$_TIMEOUT or DDTM$_COMM_FAIL abort reason code may be returned.
DECdtm quotas are returned.	If DDTM$M_NOWAIT is clear, all quotas allocated for the transaction by calls on the local node to DECdtm services are now returned.
The transaction is not the default transaction of the calling process.	If DDTM$M_NOWAIT is clear and the transaction was the default transaction of the calling process, then it is no longer the default.

There is also a wait form of the service, $END_BRANCHW.

Required Privileges

None

Required Quotas

ASTLM

Related Services

$ABORT_TRANS, $ABORT_TRANSW, $ACK_EVENT, $ADD_BRANCH, $ADD_BRANCHW, $CREATE_UID, $DECLARE_RM, $DECLARE_RMW, $END_BRANCHW, $END_TRANS, $END_TRANSW, $FORGET_RM, $FORGET_RMW, $GETDTI, $GETDTIW, $GET_DEFAULT_TRANS, $JOIN_RM, $JOIN_RMW, $SETDTI, $SETDTIW, $SET_DEFAULT_TRANS, $SET_DEFAULT_TRANSW, $START_BRANCH, $START_BRANCHW, $START_TRANS, $START_TRANSW, $TRANS_EVENT, $TRANS_EVENTW

Condition Values Returned

SS$_NORMAL

If returned in R0, the request was successfully queued. If 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$_ABORT

The transaction aborted. See the abort reason code returned in the I/O status block for one reason why the transaction aborted.

SS$_ACCVIO

An argument was not accessible to the caller.

SS$_BADPARAM

Either the options flags were invalid or the *tid* argument was omitted but the *bid* argument was not zero.

SS$_BRANCHENDED

Either the calling process had already called $END_BRANCH or $ABORT_TRANS specifying that BID, or the branch was unsynchronized.

SS$_EXASTLM

The process AST limit (ASTLM) was exceeded.

SS$_ILLEFC

The event flag number was invalid.

SS$_INSFARGS

A required argument was missing.

SS$_INSFMEM

There was insufficient system dynamic memory for the operation.

SS$_NOSUCHBID

The calling process did not contain the branch identified by the BID passed in the bid argument.

SS$_NOSUCHTID

The calling process did not contain any branches in the transaction.

SS$_WRONGACMODE

The access mode of the caller was less privileged than that of a branch of the specified transaction in this process.

$END_BRANCHW

End Branch and Wait —

Format

```
SYS$END_BRANCHW [efn] ,[flags] ,iosb ,[astadr] ,[astprm] ,tid ,bid
```

C Prototype

```
int sys$end_branchw
   (unsigned int efn, unsigned int flags, struct _iosb *iosb,
    void (*astadr)(__unknown_params), int astprm, unsigned int tid [4],
    unsigned int bid [4]);
```

$END_TRANS

End Transaction — Ends a transaction by attempting to commit it, and returns the outcome of the transaction.

Format

```
SYS$END_TRANS [efn] ,[flags] ,iosb [,[astadr] ,[astprm] ,[tid]]
```

C Prototype

```
int sys$end_trans
   (unsigned int efn, unsigned int flags, struct _iosb *iosb,...);
```

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, which are defined in Table 29.

All undefined bits must be 0. If this argument is omitted, no flag is set.

Table 29. $END_TRANS Option Flags

Flag	Description
DDTM$M_SYNC	Set this flag to specify that successful synchronous completion is to be indicated by returning SS$_SYNCH. When SS$_SYNCH is returned, the asynchronous system trap (AST) routine is not called, the event flag is not set, and the I/O status block is not filled in.
DDTM$M_NOWAIT	Indicates that the service should return to the caller without waiting for final cleanup. Note that $END_TRANSW with the DDTM$M_NOWAIT flag set is not equivalent to $END_TRANS. The former does not return until the operation has been initiated, while $END_TRANS returns when the operation has been queued. The full range of status values may be returned from a nowait call.

iosb

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

I/O status block in which the following information is returned:

- The completion status of the service. This is returned as a condition value. See the Condition Value Returned section.
- The outcome of the transaction.

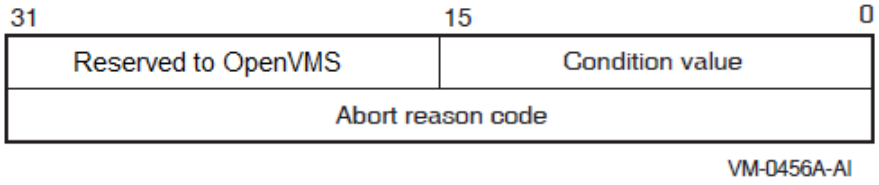
  If the service returns SS$_NORMAL, the outcome of the transaction is commit. If the service returns SS$_ABORT, the outcome of the transaction is abort.
- An abort reason code that gives one reason why the transaction aborted, if the completion status of the service is SS$_ABORT.

The $DDTMMSGDEF macro defines symbolic names for these abort reason codes, which are described in Table 30:

Table 30. Abort Reason Codes

Symbolic Name	Description
DDTM$_ABORTED	The application aborted the transaction.
DDTM$_COMM_FAIL	A communications link failed.
DDTM$_INTEGRITY	A resource manager integrity constraint check failed.
DDTM$_LOG_FAIL	A write operation to the transaction log failed.
DDTM$_ORPHAN_BRANCH	Unauthorized branch caused failure.
DDTM$_PART_SERIAL	A resource manager serialization check failed.
DDTM$_PART_TIMEOUT	The timeout specified by a resource manager expired.
DDTM$_SEG_FAIL	A process or image terminated.
DDTM$_SERIALIZATION	A DECdtm transaction manager serialization check failed.
DDTM$_SYNC_FAIL	The transaction was not globally synchronized (an authorized branch had not been added).
DDTM$_TIMEOUT	The timeout specified on $START_TRANS expired.
DDTM$_UNKNOWN	The reason is unknown.
DDTM$_VETOED	A resource manager was unable to commit the transaction.

The following diagram shows the structure of the I/O status block.



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. The 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:	trans_id
type:	octaword (unsigned)
access:	read only
mechanism:	by reference

Identifier (TID) of the transaction to be ended.

If this argument is omitted, $END_TRANS ends the default transaction of the calling process.

Description

The End Transaction service ends a transaction by attempting to commit it, and returns the outcome of the transaction.

The $END_TRANS service:

- Initiates the commit protocol to determine whether the outcome of the transaction is commit or abort.

  ### Caution

  Do not call $END_TRANS while any transaction operation is in progress. If any of these operations is in progress when $END_TRANS is called, an unintended set of operations can be committed. This can invalidate application data managed by the resource managers participating in the transaction.

  Provided that no abort event has occurred, the DECdtm transaction manager delivers a prepare event to each Resource Manager (RM) participant in the transaction that:

  - Is associated with a Resource Manager instance (RMI) that requests prepare events.
  - Did not set the DDTM$M_COORDINATOR flag when it was added to the transaction.

  If there is only one such RM participant, the DECdtm transaction manager delivers a one-phase commit event, not prepare event, to that RM participant.

- $END_TRANS returns the outcome of the transaction. If it completes successfully, the outcome of the transaction is commit. If it returns the SS$_ABORT error, the outcome is abort.
- $END_TRANS removes from the specified transaction the branch that started the transaction.

Preconditions for the successful completion of $END_TRANS are:

- The calling process must contain the branch that started the transaction.
- The access mode of the caller must be the same as or more privileged than that of any branch of the specified transaction within this process. (See $START_TRANS and $START_BRANCH.)
- $START_BRANCH must have been performed for each authorized branch of the specified transaction.

$END_TRANS may fail for various reasons, including:

- The preconditions were not met.
- An abort event has occurred for the transaction.

Postconditions on successful completion of $END_TRANS are listed in Table 31. $END_TRANS will not complete successfully (that is, the event flag will not be set, the AST routine will not be called, and the I/O status block will not be filled in) until after each authorized and synchronized branch of the transaction has initiated a call to $END_BRANCH.

$END_TRANS will not complete successfully (that is, the event flag will not be set, the AST routine will not be called, and the I/O status block will not be filled in) while the calling process is either:

- In an access mode that is more privileged than the DECdtm calls made by any resource manager participant in the transaction. RMS Journaling calls DECdtm in executive mode. Oracle Rdb and Oracle CODASYL DBMS call DECdtm in user mode.
- At AST level (in any access mode).

For example, if Oracle Rdb is a participant in the transaction, $END_TRANS will not complete successfully while the calling process is in supervisor, executive, or kernel mode, or while the calling process is at AST level. Successful completion of $END_TRANS is not indefinitely postponed by network failure.

Table 31. Postconditions When $END_TRANS Completes Successfully

Postcondition	Meaning
The transaction is ended.	The meanings are: • The TID of the transaction is invalid. Calls to any DECdtm system services except $GETDTI and $SETDTI that pass that TID will fail, and calls to resource managers that pass the TID will fail. • The transaction no longer has any application or RM participants on the local node. • All communications about the transaction between the local DECdtm transaction manager and other DECdtm transaction managers are finished (including the final cleanup acknowledgments).
The outcome of the transaction is commit.	All the transaction operations by authorized branches that completed successfully before $END_TRANS was called will take effect. That is,

Postcondition	Meaning
	the effects of these operations will be made permanent. Operations by unauthorized branches will be aborted. (An unauthorized branch is one without a matching $ADD_BRANCH.)
DECdtm quotas are returned.	All quotas allocated for the transaction by calls on the local node to DECdtm services are now returned.
The transaction is not the default transaction of the calling process.	If the transaction was the default transaction of the calling process, then it is now no longer the default.

Postconditions on completion with SS$_ABORT error are listed in Table 32. $END_TRANS does not complete with the SS$_ABORT error until all branches on the local node have been removed from the transaction. Thus it does not complete with this error until after each authorized and synchronized branch on the local node has initiated a call to either $END_BRANCH or $ABORT_TRANS.

Note that the completion of $END_TRANS with the SS$_ABORT error is not indefinitely postponed by network failure.

Table 32. Postconditions When $END_TRANS Completes with SS$_ABORT Error

Postcondition	Meaning
The transaction is ended.	If DDTM$M_NOWAIT is clear: • The TID of the transaction is invalid. Calls to any DECdtm system services except $GETDTI and $SETDTI that pass the TID will fail, and calls to resource managers that pass the TID will fail. • The transaction no longer has any application or RM participants on the local node. • All communications about the transaction between the local DECdtm transaction manager and other DECdtm transaction managers are finished (including the final "cleanup" acknowledgments).
The outcome of the transaction is abort.	None of the operations of the transaction will ever take effect. The I/O status block contains one reason why the transaction was aborted. Note that if there are multiple reasons for the transaction aborting, the DECdtm transaction manager returns one of the reasons in the I/O status block. It may return different reasons to different branches in the transaction. For example, if the transaction timeout expires and a communications link fails, then either the DDTM$_TIMEOUT or DDTM$_COMM_FAIL abort reason code may be returned.
DECdtm quotas are returned.	If DDTM$M_NOWAIT is clear, all quotas allocated for the transaction by calls on the local node to DECdtm services are now returned.
The transaction is not the default transaction of the calling process.	If DDTM$M_NOWAIT is clear then, if the transaction was the default transaction of the calling process, then the transaction is now no longer the default.

There is also a wait form of the service, $END_TRANSW.

Required Access or Privileges

None

Required Quotas

ASTLM

Related Services

$ABORT_TRANS, $ABORT_TRANSW, $ACK_EVENT, $ADD_BRANCH, $ADD_BRANCHW, $CREATE_UID, $DECLARE_RM, $DECLARE_RMW, $END_BRANCH, $END_BRANCHW, $END_TRANSW, $FORGET_RM, $FORGET_RMW, $GETDTI, $GETDTIW, $GET_DEFAULT_TRANS, $JOIN_RM, $JOIN_RMW, $SETDTI, $SETDTIW, $SET_DEFAULT_TRANS, $SET_DEFAULT_TRANSW, $START_BRANCH, $START_BRANCHW, $START_TRANS, $START_TRANSW, $TRANS_EVENT, $TRANS_EVENTW

Condition Values Returned

SS$_NORMAL

If returned in R0, the request was successfully queued. If 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$_ABORT

The transaction aborted. See the abort reason code returned in the I/O status block for one reason why the transaction aborted.

SS$_ACCVIO

An argument was not accessible to the caller.

SS$_BADPARAM

The options flags were invalid.

SS$_CURTIDCHANGE

The *tid* argument was omitted 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$_ILLEFC

The event flag number was invalid.

SS$_INSFARGS

A required argument was missing.

SS$_INSFMEM

There was insufficient system dynamic memory for the operation.

SS$_NOCURTID

An attempt was made to end the default transaction (the *tid* argument was omitted), but the calling process did not have a default transaction.

SS$_NOSUCHTID

The calling process did not contain any branches in the transaction.

SS$_NOTORIGIN

The calling process did not start the transaction.

SS$_NOSUCHTID

A transaction with the specified transaction identifier does not exist.

SS$_NOTORIGIN

The calling process did not start the transaction.

SS$_WRONGACMODE

The access mode of the caller was less privileged than that of a branch of the transaction in this process.

SS$_WRONGSTATE

The calling process had already called either $ABORT_TRANS with a zero BID or $END_TRANS to end the transaction, and processing had not completed.

$END_TRANSW

End Transaction and Wait — Ends a transaction by attempting to commit it, and returns the outcome of the transaction. $END_TRANSW always waits for the request to complete before returning to the caller. Other than this, it is identical to $END_TRANS. Do not call $END_TRANSW from asynchronous system trap (AST) level, or from an access mode that is more privileged than the DECdtm calls made by any resource manager participant in the transaction. If you do, the $END_TRANSW service will wait indefinitely.

Format

```
SYS$END_TRANSW [efn] ,[flags] ,iosb [,[astadr] ,[astprm] ,[tid]]
```

C Prototype

```
int sys$end_transw
```

```
   (unsigned int efn, unsigned int flags, struct _iosb *iosb,...);
```

$ENQ

Enqueue Lock Request — Queues a new lock or lock conversion on a resource. The $ENQ, $ENQW, $DEQ (Dequeue Lock Request), and $GETLKI (Get Lock Information) services together provide the user interface to the Lock Management facility. For additional information about lock management, see the descriptions of these other services. On Alpha and Integrity server systems, this service accepts 64-bit addresses. For additional information about system service completion, see the Synchronize ($SYNCH) service.

Format

```
SYS$ENQ
   [efn] ,lkmode ,lksb ,[flags] ,[resnam] ,[parid] ,[astadr] ,[astprm]
   ,[blkast] ,[acmode] ,[rsdm_id] ,[nullarg]
```

C Prototype

```
int sys$enq
   (unsigned int efn, unsigned int lkmode, struct _lksb *lksb,
    unsigned int flags, void *resnam, unsigned int parid,
    void (*astadr)(__unknown_params), int astprm,
    void (*blkast)(__unknown_params), unsigned int acmode,
    unsigned int rsdm_id,...);
```

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 request has been granted or canceled. Cancellation occurs if you use $DEQ with the cancel modifier or if the waiting request is chosen to break a deadlock. The *efn* argument is a longword containing this number; however, $ENQ uses only the low-order byte.

Upon request initiation, $ENQ clears the specified event flag (or event flag 0 if *efn* was not specified). Then, when the lock request is granted, the specified event flag (or event flag 0) is set unless you specified the LCK$M_SYNCSTS flag in the *flags* argument.

lkmode

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

Lock mode requested. The *lkmode* argument is a longword specifying this lock mode.

Each lock mode has a symbolic name. The $LCKDEF macro defines these symbolic names. The following table gives the symbolic name and description for each lock mode.

Lock Mode	Description
LCK$K_NLMODE	Null mode. This mode grants no access to the resource but serves as a placeholder and indicator of future interest in the resource. The null mode does not inhibit locking at other lock modes; further, it prevents the deletion of the resource and lock value block, which would otherwise occur if the locks held at the other lock modes were dequeued.
LCK$K_CRMODE	Concurrent read. This mode grants the caller read access to the resource while permitting write access to the resource by other users. This mode is used to read data from a resource in an unprotected manner, because other users can modify that data as it is being read. This mode is typically used when additional locking is being performed at a finer granularity with sublocks.
LCK$K_CWMODE	Concurrent write. This mode grants the caller write access to the resource while permitting write access to the resource by other users. This mode is used to write data to a resource in an unprotected fashion, because other users can simultaneously write data to the resource. This mode is typically used when additional locking is being performed at a finer granularity with sublocks.
LCK$K_PRMODE	Protected read. This mode grants the caller read access to the resource while permitting only read access to the resource by other users. Write access is not allowed. This is the traditional *share lock.*
LCK$K_PWMODE	Protected write. This mode grants the caller write access to the resource while permitting only read access to the resource by other users; the other users must have specified concurrent read mode access. No other writers are allowed access to the resource. This is the traditional *update lock.*
LCK$K_EXMODE	Exclusive. The exclusive mode grants the caller write access to the resource and allows no access to the resource by other users. This is the traditional *exclusive lock.*

The following table shows the compatibility of lock modes:

Table 33. Compatibility of Lock Modes

Mode of Requested Lock	Mode of Currently Granted Locks					
	NL	**CR**	**CW**	**PR**	**PW**	**EX**
NL	Yes	Yes	Yes	Yes	Yes	Yes
CR	Yes	Yes	Yes	Yes	Yes	No
Key to Lock Modes **NL—Null** **CR—Concurrent read** **CW—Concurrent write** **PR—Protected read** **PW—Protected write** **EX—Exclusive**						

Mode of Requested Lock	Mode of Currently Granted Locks					
	NL	CR	CW	PR	PW	EX
CW	Yes	Yes	Yes	No	No	No
PR	Yes	Yes	Yes	Yes	No	No
PW	Yes	Yes	No	No	No	No
EX	Yes	No	No	No	No	No

Key to Lock Modes

NL—Null
CR—Concurrent read
CW—Concurrent write
PR—Protected read
PW—Protected write
EX—Exclusive

lksb

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

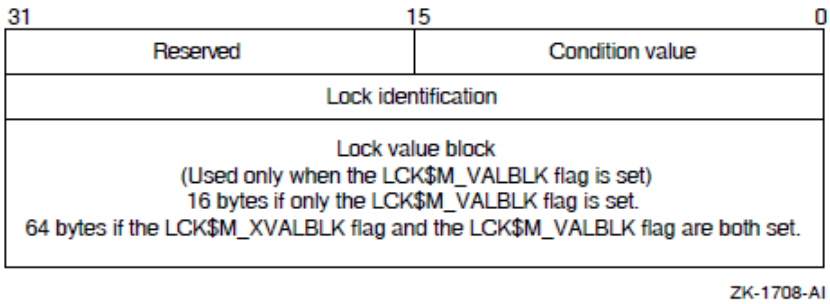
Lock status block in which $ENQ writes the final completion status of the operation. The *lksb* argument is the 32- or 64-bit address of the 8-byte lock status block.

The lock status block can optionally contain a 16-byte or a 64-byte lock value block. The initial value of the lock value block is zero (0).

- When you specify the LCK$M_VALBLK flag in the *flags* argument, the lock status block contains a lock value block. In this case, the 16-byte lock value block appears at the beginning of the first byte following the eighth byte of the lock status block, bringing the total length of the lock status block to 24 bytes.
- When you specify the LCK$M_XVALBLK flag together with the LCK$M_VALBLK flag in the *flags* argument, the lock status block contains an extended lock value block. In this case, the 64-byte lock value block appears at the beginning of the first byte following the eighth byte of the lock status block, bringing the total length of the lock status block to 72 bytes.

  The LCK$M_XVALBLK flag is valid only on Alpha and Integrity server systems.

The following diagram shows the format of the lock status block and the optional lock value block.



The following table defines the status block fields.

Status Block Field	Definition
Condition value	A word in which $ENQ writes a condition value describing the final disposition of the lock request; for example, whether the lock was granted, converted, and so on. The condition values returned in this field are described in the Condition Values Returned in the Lock Status Block section, which appears following the list of condition values returned in R0.
Reserved	A word reserved to OpenVMS.
Lock identification	A longword containing the identification of the lock. For a new lock, $ENQ writes the lock identification of the requested lock into this longword when the lock request is queued. For a lock conversion on an existing lock, you must supply the lock identification of the existing lock in this field.
Lock value block	A user-defined structure containing information about the resource. This information is interpreted only by the user program. The length of the user data structure is 16 bytes if only the LCK$M_VALBLK flag is specified. The user data structure is 64 bytes if both the LCK$M_VALBLK and LCK$M_XVALBLK flags are specified. The length of the system copy of the lock value block structure is always 64 bytes on OpenVMS and Integrity server systems beginning with OpenVMS Version 8.2. Refer to the *VSI OpenVMS Programming Concepts Manual* for information about using the LCK$M_XVALBLK flag in a mixed-version cluster. When a process acquires a lock on a resource, the lock management facility provides that process with a process-private copy of the lock value block associated with the resource, provided that process has specified the LCK$M_VALBLK flag in the `flags` argument. The copy provided to the process is a copy of the lock value block stored in the lock manager's database. The copy of the lock value block maintained in the lock database is either read into or updated from the caller's lock value block. The method used depends on the lock mode of the lock that was granted, and on the mode of the original lock held, if the operation was a conversion. In general, a grant or a conversion to an equal-level or higher-level lock mode reads the lock value from the lock database into the caller's lock value block. When a lock conversion from EX-mode or PW-mode to an equal-level or lower-level lock mode occurs, the contents of the caller's lock value block are written into the lock database. The specific behavior of the lock conversion is documented in the *VSI OpenVMS Programming Concepts Manual* in the table entitled, "Effect of Lock Conversion on Lock Value Block".

Callers of $ENQ are provided with copies of the updated lock value block from the lock database in the following way: when $ENQ grants a new lock to the caller or converts the caller's existing lock to the same lock mode or a higher lock mode, $ENQ copies the lock value block from the lock database to the caller's lock value block, provided the caller has specified the LCK$M_VALBLK flag.

The Description section describes events that can cause the lock value block to become invalid.

flags

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

Flags specifying options for the $ENQ operation. The *flags* argument is a longword bit mask that is the logical OR of each bit set, where each bit corresponds to an option.

The $LCKDEF macro defines a symbolic name for each flag bit. The following table describes each flag.

Flag	Description
LCK$M_NOQUEUE	When this flag is specified, $ENQ does not queue the lock request unless the lock can be granted immediately. By default, $ENQ always queues the request. If you specify LCK$M_NOQUEUE in a lock conversion operation and the conversion cannot be granted immediately, the lock remains in the original lock mode.
LCK$M_SYNCSTS	When you specify this flag, $ENQ returns the successful condition value SS$_SYNCH in R0 if the lock request is granted immediately; in this case, no completion asynchronous system trap (AST) is delivered and no event flag is set. If the lock request is queued successfully but cannot be granted immediately, $ENQ returns the condition value SS$_NORMAL in R0; then when the request is granted, $ENQ sets the event flag and queues an AST if the *astadr* argument was specified.
LCK$M_SYSTEM	When you specify this flag, the resource name is interpreted as systemwide. By default, resource names are qualified by the user identification code (UIC) group number of the creating process. This flag is ignored in lock conversions.
LCK$M_VALBLK	When you specify this flag, the lock status block contains a lock value block. The initial value of the lock value block is zero (0). See the description of the *lksb* argument and the LCK$M_XVALBLK flag for more information.
LCK$M_CONVERT	When you specify this flag, $ENQ performs a lock conversion. In this case, the caller must supply (in the second longword of the lock status block) the lock identification of the lock to be converted.
LCK$M_NODLCKWT	By specifying this flag, a process indicates to the lock management services that it is not blocked from execution while waiting for the lock request to complete. For example, a lock request might be left outstanding on the waiting queue as a signaling device between processes. This flag helps to prevent false deadlocks by providing the lock management services with additional information about the

Flag	Description
	process issuing the lock request. When you set this flag, the lock management services do not consider this lock when trying to detect deadlock conditions.
	A process should specify the LCK$M_NODLCKWT flag only in a call to the $ENQ system service. The $ENQW system service waits for the lock request to be granted before returning to the caller; therefore, specifying the LCK$M_NODLCKWT flag in a call to the $ENQW system service defeats the purpose of the flag and can result in a genuine deadlock being ignored. The lock management services make use of the LCK$M_NODLCKWT flag only when the lock specified by the call to $ENQ is in either the waiting or the conversion queue. Improper use of the LCK$M_NODLCKWT flag can result in the lock management services ignoring genuine deadlocks.
LCK$M_NODLCKBLK	By specifying this flag, a process indicates to the lock management services that, if this lock is blocking another lock request, the process intends to give up this lock on demand. When you specify this flag, the lock management services do not consider this lock as blocking other locks when trying to detect deadlock conditions. A process typically specifies the LCK$M_NODLCKBLK flag only when it also specifies a blocking AST. Blocking ASTs notify processes with granted locks that another process with an incompatible lock mode has been queued to access the same resource. Use of blocking ASTs can cause false deadlocks, because the lock management services detect a blocking condition, even though a blocking AST has been specified; however, the blocking condition will disappear as soon as the process holding the lock executes, receives the blocking AST, and dequeues the lock. Specifying the LCK$M_NODLCKBLK flag prevents this type of false deadlock.
	To enable blocking ASTs, the *blkast* argument of the $ENQ system service must contain the address of a blocking AST service routine. If the process specifies the LCK$M_NODLCKBLK flag, the blocking AST service routine should either dequeue the lock or convert it to a lower lock mode without issuing any new lock requests. If the blocking AST routine does otherwise, a genuine deadlock could be ignored. The lock management services make use of the LCK$M_NODLCKBLK flag only when the lock specified by the call to $ENQ has been granted. Improper use of the LCK$M_NODLCKBLK flag can result in the lock management services ignoring genuine deadlocks.
LCK$M_NOQUOTA	This flag is reserved to OpenVMS. When you set this flag, the calling process is not charged Enqueue Limit (ENQLM) quota for this new lock. The calling process must be running in executive

Flag	Description
	or kernel mode to set this flag. This flag is ignored for lock conversions.
LCK$M_CVTSYS	This flag is reserved to OpenVMS. When you set this flag, the lock is converted from a process-owned lock to a system-owned lock. The calling process must be running in executive or kernel mode to set this flag.
LCK$M_EXPEDITE	This flag is valid only for new lock requests. Specifying this flag allows a request to be granted immediately, provided the requested mode when granted would not block any currently queued requests in the resource conversion and wait queues. Currently, this flag is valid only for NLMODE requests. If this flag is specified for any other lock mode, the request will fail and an error of SS$_UNSUPPORTED will be returned.
LCK$M_QUECVT	This flag is valid only for conversion operations. A conversion request with the LCK$M_QUECVT flag set will be forced to wait behind any already queued conversions. The conversion request is granted immediately, if there are no already queued conversions. The QUECVT behavior is valid only for a subset of all possible conversions. Table 34 defines the legal set of conversion requests for LCK$M_QUECVT. Illegal conversion requests are failed with SS$_BADPARAM returned.
LCK$M_XVALBLK	This flag is valid only if it is used in conjunction with the LCK$M_VALBLK flag. When you specify the LCK$M_XVALBLK flag, you must provide a 64-byte lock value block at the end of the lock states block specified in the *lksb* argument. If you do not specify this flag, only the first 16 bytes of the lock value block buffer specified as part of the lock status block in the *lksb* argument will be read or written. If the value block is written without this flag, the value block will be flagged so that a future reader who specifies the LCK$M_XVALBLK flag in the $ENQ system service call will receive the warning status SS$_XVALNOTVALID until a future writer writes to the value block specifying this flag.

Table 34. Legal QUECVT Conversions

Lock Mode at Which Lock Is Held	Lock Mode to Which Lock Is Converted					
	NL	**CR**	**CW**	**PR**	**PW**	**EX**
NL	No	Yes	Yes	Yes	Yes	Yes
CR	No	No	Yes	Yes	Yes	Yes
CW	No	No	No	Yes	Yes	Yes
PR	No	No	Yes	No	Yes	Yes
PW	No	No	No	No	No	Yes
EX	No	No	No	No	No	No
Key to Lock Modes **NL—Null lock** **CR—Concurrent read** **CW—Concurrent write** **PR—Protected read** **PW—Protected write** **EX—Exclusive lock**						

resnam

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

Name of the resource to be locked by this lock. The *resnam* argument is the 32- or 64-bit address of a character string descriptor pointing to this name. The name string can be from 1 to 31 bytes in length.

If you are creating a new lock, the *resnam* argument should be specified because the default value for the *resnam* argument produces an error when it is used to create a lock. The *resnam* argument is ignored for lock conversions.

parid

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

Lock identification of the parent lock. The *parid* argument is a longword containing this identification value.

If you do not specify this argument or specify it as 0, $ENQ assumes that the lock does not have a parent lock. This argument is optional for new locks and is ignored for lock conversions.

astadr

OpenVMS usage:	ast_procedure

type:	procedure value
access:	call without stack unwinding
mechanism:	by 32- or 64-bit reference

AST service routine to be executed when the lock is either granted or converted. The *astadr* argument is the 32- or 64-bit address of this routine. The AST is also delivered when the lock or conversion request is canceled. Cancellation occurs if you use $DEQ with the cancel modifier or if the waiting request is chosen to break a deadlock.

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

astprm

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

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

blkast

OpenVMS usage:	ast_procedure
type:	procedure value
access:	call without stack unwinding
mechanism:	by 32- or 64-bit reference

Blocking AST routine to be called whenever this lock is granted and is blocking any other lock requests. The *blkast* argument is the 32- or 64-bit address of this routine. Locks that are converting to a new mode, but that are not yet granted in the new mode, do not receive blocking ASTs.

You can pass a parameter to this routine by using the *astprm* argument.

acmode

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

Access mode to be associated with the resource name. The *acmode* argument indicates the least privileged access mode from which locks can be queued on the resource.

This argument does not affect the access mode associated with the lock or its blocking and completion ASTs. The *acmode* argument is a longword containing the access mode. The $PSLDEF macro defines the following symbols for the four access modes.

Symbol	Access Mode
PSL$C_KERNEL	Kernel

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

The $ENQ service associates an access mode with the lock in the following way:

- If you specified a parent lock (with the *parid* argument), $ENQ uses the access mode associated with the parent lock and ignores both the *acmode* argument and the caller's access mode.
- If the lock has no parent lock (you did not specify the *parid* argument or specified it as 0), $ENQ uses the least privileged of the caller's access mode and the access mode specified by the *acmode* argument. If you do not specify the *acmode* argument, $ENQ uses the caller's access mode.

rsdm_id

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

Resource domain identification. The *rsdm_id* argument is a longword specifying the resource domain association through which a new lock is to be taken. This argument is ignored for lock conversions and sublocks (*parid* is nonzero). Valid resource domain identifiers are returned from the $SET_RESOURCE_DOMAIN service, or by the constants RSDM$K_SYSTEM_RSDM_ID or RSDM$K_PROCESS_RSDM_ID, which are defined by the $RSDMDEF macro in STARLET.

nullarg

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

Placeholding argument reserved to OpenVMS.

Description

The Enqueue Lock Request service queues a new lock or lock conversion on a resource. The $ENQ service completes asynchronously; that is, it returns to the caller after queuing the lock request without waiting for the lock to be either granted or converted. For synchronous completion, use the Enqueue Lock Request and Wait ($ENQW) service. The $ENQW service is identical to the $ENQ service in every way except that $ENQW returns to the caller when the lock is either granted or converted.

The $ENQ service uses system dynamic memory for the creation of the lock and resource blocks.

When $ENQ queues a lock request, it returns the status of the request in R0 and writes the lock identification of the lock in the lock status block. Then, when the lock request is granted, $ENQ writes the final completion status in the lock status block, sets the event flag, and calls the AST routine if this has been requested.

When $ENQW queues a lock request, it returns status in R0 and in the lock status block when the lock has been either granted or converted. Where applicable, it simultaneously sets the event flag and calls the AST routine.

Invalidation of the Lock Value Block

In some situations, the lock value block can become invalid. In these situations, $ENQ warns the caller by returning the condition value SS$_VALNOTVALID in the lock status block, provided the caller has specified the flag LCK$M_VALBLK in the *flags* argument.

The SS$_VALNOTVALID condition value is a warning message, not an error message; therefore, the $ENQ service grants the requested lock and returns this warning on all subsequent calls to $ENQ until either a new lock value block is written to the lock database or the resource is deleted. Resource deletion occurs when no locks are associated with the resource.

The following events can cause the lock value block to become invalid:

- If any process holding a protected write or exclusive mode lock on a resource is terminated abnormally or exits before explicitly dequeuing the lock or converting it to a lover-level lock mode, the lock value block becomes invalid.
- If a process holding a protected write or exclusive mode lock on the resource calls the Dequeue Lock Request ($DEQ) service to dequeue this lock and specifies the flag LCK$M_INVVALBLK in the *flags* argument, the lock value block maintained in the lock database is marked invalid.
- If a node in an OpenVMS Cluster system fails, and a process on that node was holding or might have been holding a protected write or exclusive mode lock on the resource, the lock value block becomes invalid.

  This situation is dependant on which cluster node is the master node for the resource. The following describes the two ways in which this situation can arise:

  1. If a node was holding a protected write or exclusive mode lock on the resource:

     If the node that failed was not the master of the resource tree, the master node will know what locks were held by the node that failed. These locks will be removed from the resource and if a removed lock was for protected write or exclusive mode, the lock value block becomes in valid.

  2. If a node might have been holding a protected write or exclusive mode lock on the resource:

     If the node that failed was the master of the resource, a remaining node in the cluster will become the new master. If the new master finds that all granted locks on the resource were for null mode locks or for concurrent read locks, or for both, then the new master does not know if the failed node held a protected write or exclusive mode lock. The lock value block will be set to invalid in this case.

     If the resource has any granted lock with a mode other than null mode or concurrent read, then the failed node could not have held a protected write or exclusive mode lock, and the lock value block will not be marked as invalid.

Invalidation of the Extended Lock Value Block

The extended lock value block can be marked invalid in the following situation: If a program updates the lock block specifying only LCK$M_VALBLK without LCK$M_XVALBLK, only the first 16 bytes of the lock value block will be written. The remaining 48 bytes will not be modified. A reader who, in the

future, specifies the LCK$M_XVALBLK flag in the $ENQ system service call will be given all 64 bytes but will receive the warning status SS$_XVALNOTVALID flag.

If the entire lock status block is invalid as described in the Invalidation of the Lock Value Block section, the SS$_VALNOTVALID status will be returned, overriding the SS$_XVALNOTVALID status.

Required Access or Privileges

To queue a lock on a systemwide resource, the calling process must either have SYSLCK privilege or be executing in executive or kernel mode.

To specify a parent lock when queuing a lock, the access mode of the caller must be equal to, or less privileged than, the access mode associated with the parent lock.

To queue a lock conversion, the access mode associated with the lock being converted must be equal to, or less privileged than, the access mode of the calling process.

Required Quota

- Enqueue limit (ENQLM) quota
- AST limit (ASTLM) quota in lock conversion requests that you specify either the *astadr* or *blkast* argument

Related Services

$DEQ, $ENQW, $GETLKI, $GETLKIW, $SET_RESOURCE_DOMAIN

Condition Values Returned

SS$_NORMAL

The service completed successfully; the lock request was successfully queued.

SS$_SYNCH

The service completed successfully; the LCK$M_SYNCSTS flag in the *flags* argument was specified, and $ENQ was able to grant the lock request immediately.

SS$_ACCVIO

The lock status block or the resource name cannot be read.

SS$_BADPARAM

You specified an invalid lock mode in the *lkmode* argument.

SS$_CVTUNGRANT

You attempted a lock conversion on a lock that is not currently granted.

SS$_EXDEPTH

The limit of levels of sublocks has been exceeded.

SS$_EXENQLM

The process has exceeded its enqueue limit (ENQLM) quota.

SS$_INSFMEM

The system dynamic memory is insufficient for creating the necessary data structures.

SS$_IVBUFLEN

The length of the resource name was either 0 or greater than 31.

SS$_IVLOCKID

You specified an invalid or nonexistent lock identification, or the lock identified by the lock identification has an associated access mode that is more privileged than the caller's, or the access mode of the parent was less privileged than that of the caller.

SS$_NOLOCKID

No lock identification was available for the lock request.

SS$_NOSYSLCK

The LCK$M_SYSTEM flag in the *flags* argument was specified, but the caller lacks the necessary SYSLCK privilege.

SS$_NOTQUEUED

The lock request was not queued; the LCK$M_NOQUEUE flag in the *flags* argument was specified, and $ENQ was not able to grant the lock request immediately.

SS$_PARNOTGRANT

The parent lock specified in the *parid* argument was not granted.

Condition Values Returned in the Lock Status Block

SS$_NORMAL

The service completed successfully; the lock was successfully granted or converted.

SS$_ABORT

The lock was dequeued (by the $DEQ service) before $ENQ could grant the lock.

SS$_CANCEL

The lock conversion request has been canceled and the lock has been regranted at its previous lock mode. This condition value is returned when $ENQ queues a lock conversion request, the request has not been granted yet (it is in the conversion queue), and, in the interim, the $DEQ service is called (with the LCK$M_CANCEL flag specified) to cancel this lock conversion request. If the lock is granted before $DEQ can cancel the conversion request, the call to $DEQ returns the condition value SS$_CANCELGRANT, and the call to $ENQ returns SS$_NORMAL.

SS$_DEADLOCK

A deadlock was detected.

SS$_ILLRSDM

The operation attempted is not allowed on the resource. Use SHOW SECURITY to verify the access allowed to the specified resource domain.

SS$_NODOMAIN

The RSDM_ID argument passed to the $ENQ call either does not correspond to a valid resource domain for your process, or the system is not running the audit server process.

SS$_VALNOTVALID

The lock value block is marked invalid. This warning message is returned only if the caller has specified the flag LCK$M_VALBLK in the *flags* argument. Note that the lock has been successfully granted despite the return of this warning message. See the Description section for a complete discussion of lock value block invalidation.

SS$_XVALNOTVALID

The extended value block has been marked invalid because the previous writer has written the value block without specifying the LCK$M_XVALBLK flag. This warning message is returned only if the caller has specified the LCK$M_XVALBLK flag in the *flags* argument. Note that the lock is successfully granted despite the return of this warning message.

For a detailed discussion of extended lock value block invalidation, see the Invalidation of the Extended Lock Value Block section.

$ENQW

Enqueue Lock Request and Wait — Queues a lock on a resource. The $ENQW service completes synchronously; that is, it returns to the caller when the lock has been either granted or converted. For asynchronous completion, use the Enqueue Lock Request ($ENQ) service; $ENQ returns to the caller after queuing the lock request, without waiting for the lock to be either granted or converted. In all other respects, $ENQW is identical to $ENQ. See the $ENQ description for all other information about the $ENQW service. For additional information about system service completion, see the Synchronize ($SYNCH) service documentation. The $ENQ, $ENQW, $DEQ, and $GETLKI services together provide the user interface to the Lock Management facility. On Alpha and Integrity server systems, this service accepts 64-bit addresses.

Format

```
SYS$ENQW
   [efn] ,lkmode ,lksb ,[flags] ,[resnam] ,[parid] ,[astadr]
   ,[astprm] ,[blkast] ,[acmode] ,[rsdm_id]
```

C Prototype

```
int sys$enqw
   (unsigned int efn, unsigned int lkmode, struct _lksb *lksb,
    unsigned int flags, void *resnam, unsigned int parid,
    void (*astadr)(__unknown_params), unsigned __int64 astprm,
    void (*blkast)(__unknown_params), unsigned int acmode,
    unsigned int rsdm_id,...);
```

$ENTER

Inserts File Name in Directory — The Enter service inserts a file name in a directory. For additional information about this service, see the *VSI OpenVMS Record Management Services Reference Manual*.

$ERAPAT

Get Security Erase Pattern — Generates a security erase pattern.

Format

```
SYS$ERAPAT [type] ,[count] ,[patadr]
```

C Prototype

```
int sys$erapat (int type, unsigned int count, unsigned  int *patadr);
```

Arguments

type

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

Type of storage to be written over with the erase pattern. The *type* argument is a longword containing the type of storage. The three storage types, together with their symbolic names, are defined by the $ERADEF macro and are listed in the following table.

Storage Type	Symbolic Name
Main memory	ERA$K_MEMORY
Disk	ERA$K_DISK
Tape	ERA$K_TAPE

count

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

Number of times that $ERAPAT has been called in a single security erase operation. The *count* argument is a longword containing the iteration count.

You should call the $ERAPAT service initially with the *count* argument set to 1, the second time with the *count* argument set to 2, and so on, until the status code SS$_NOTRAN is returned.

patadr

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

Security erase pattern to be written. The *patadr* argument is the address of a longword into which the security erase pattern is to be written.

Description

The Get Security Erase Pattern service generates a security erase pattern that can be written into memory areas containing outdated but sensitive data to make it unreadable. This service is used primarily by the operating system, but it can also be used by users who want to perform security erase operations on foreign disks.

You should call the $ERAPAT service iteratively until the completion status SS$_NOTRAN is returned.

The following example demonstrates how to use the $ERAPAT service to perform a security erase to a disk. Note that, after each call to $ERAPAT, a test for the status SS$_NOTRAN is made. If SS$_NOTRAN has not been returned, $QIO is called to write the pattern returned by $ERAPAT onto the disk. After this write, $ERAPAT is called again and the cycle is repeated until the code SS$_NOTRAN is returned, at which point the security erase procedure is complete.

```
; Code fragment that erases 20 blocks (blocks 15 through 34) on a disk
;
PATTERN:
        .LONG   0                       ; Cell to contain output from $ERAPAT
CHANNEL:
        .WORD   0                       ; Channel assigned to disk device
DEVICE: .ASCID  /DISK:/                 ; Disk device name
                .
                .
                .
        $ASSIGN_S DEVNAM=DISK,-         ; Assign a channel to the device
                CHAN=CHANNEL
        BLBC    R0, EXIT                ; Branch if error
                .
                .
                .
        MOVL    #1, R2                  ; Set initial count

        $ERADEF                         ; Macro to define names
                                        ; used by $ERAPAT

10$:    $ERAPAT_S -                     ; Call the $ERAPAT service
                COUNT=R2,-
                TYPE=#ERA$K_DISK,-
                PATADR=PATTERN
        BLBC    R0, EXIT                ; Branch if error
        CMPL    #SS$_NOTRAN, R0         ; Are we done?
        BEQL    EXIT                    ; Branch if so
        $QIO_S  CHAN=CHANNEL,-
                FUNC=#IO$_WRITELBLK!IO$M_ERASE,-        ; Call
                P1=PATTERN,-            ; to the $QIO service
                P2=#<20*512>,-          ; to write the erase
                P3=#15                  ; pattern

        INCL    R2                      ; Increase count

        BRB     10$
```

```
EXIT:           .
                .
                .
```

Required Access or Privileges

None

Required Quota

None

Related Services

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

Condition Values Returned

SS$_NORMAL

The service completed successfully; proceed with the next erase step.

SS$_NOTRAN

The service completed successfully; security erase completed.

SS$_ACCVIO

The *patadr* argument cannot be written by the caller.

SS$_BADPARAM

The *type* argument or *count* argument is invalid.

$ERASE

Deletes a Disk File — The Erase service deletes a disk file and removes the file's directory entry specified in the path to the file. If additional directory entries have been created for this file by the Enter service, you must use the Remove service to delete them. For additional information about this service, see the *VSI OpenVMS Record Management Services Reference Manual*.

$EXIT

Exit — Initiates image rundown when the current image in a process completes execution. Control normally returns to the command interpreter.

Format

```
SYS$EXIT [code]
```

C Prototype

```
int sys$exit (unsigned int code);
```

Argument

code

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

Longword value to be saved in the process header as the completion status of the current image. If you do not specify this argument in a macro call, a value of 1 is passed as the completion code for VAX MACRO and VAX BLISS–32, and a value of 0 is passed for other languages. You can test this value at the command level to provide conditional command execution.

Description

The $EXIT service is unlike all other system services in that it does not return status codes in R0 or anywhere else. The $EXIT service does not return control to the caller; it performs an exit to the command interpreter or causes the process to terminate if no command interpreter is present.

Required Access or Privileges

None

Required Quota

None

Related Services

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

$EXPREG

Expand Program/Control Region — Adds a specified number of new virtual pages to a process's program region or control region for the execution of the current image. Expansion occurs at the current end of that region's virtual address space.

Format

```
SYS$EXPREG pagcnt ,[retadr] ,[acmode] ,[region]
```

C Prototype

```
int sys$expreg
   (unsigned int pagcnt, struct _va_range *retadr,  unsigned int acmode,
    char region);
```

Arguments

pagcnt

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

Number of pagelets to add to the current end of the program or control region. The *pagcnt* argument is a longword value containing this number.

On Alpha and Integrity server systems, the specified value is rounded up to an even multiple of the CPU-specific page size.

retadr

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

Starting and ending process virtual addresses of the pages that $EXPREG has actually added. The *retadr* argument is the address of a 2-longword array containing, in order, the starting and ending process virtual addresses.

acmode

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

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

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

The newly added pages are given the following protection: (1) read and write access for access modes equal to or more privileged than the access mode used in the call, and (2) no access for access modes less privileged than that used in the call.

region

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

Number specifying which program region is to be expanded. The *region* argument is a longword value. A value of 0 (the default) specifies that the program region (P0 region) is to be expanded. A value of 1 specifies that the control region (P1 region) is to be expanded.

Description

The Expand Program/Control Region service adds a specified number of new virtual pages to a process's program region or control region for the execution of the current image. Expansion occurs at the current end of that region's virtual address space.

The new pages, which were previously inaccessible to the process, are created as demand-zero pages.

Because the bottom of the user stack is normally located at the end of the control region, expanding the control region is equivalent to expanding the user stack. The effect is to increase the available stack space by the specified amount.

The starting address returned is always the first available page in the designated region; therefore, the ending address is smaller than the starting address when the control region is expanded and is larger than the starting address when the program region is expanded.

If an error occurs while pages are being added, the *retadr* argument (if specified) indicates the pages that were successfully added before the error occurred. If no pages were added, both longwords of the *retadr* argument contain the value –1.

Required Access or Privileges

None

Required Quota

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

Related Services

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

Typically, the information returned in the location addressed by the *retadr* argument (if specified) can be used as the input range to the Delete Virtual Address Space ($DELTVA) service.

Condition Values Returned

SS$_NORMAL

The service completed successfully.

SS$_ACCVIO

The return address array cannot be written by the caller.

SS$_EXQUOTA

The process exceeded its paging file quota.

SS$_ILLPAGCNT

The specified page count was less than 1 or would cause the program or control region to exceed its maximum size.

SS$_INSFWSL

The process's working set limit is not large enough to accommodate the increased virtual address space.

SS$_VASFULL

The process's virtual address space is full. No space is available in the process page table for the requested regions.

$EXPREG_64 (Alpha and Integrity servers)

Expand Virtual Address Space — On Alpha and Integrity server systems, adds a specified number of demand-zero allocation pages to a process's virtual address space for the execution of the current image. Expansion occurs at the next free available address within the specified region. This service accepts 64-bit addresses.

Format

```
SYS$EXPREG_64
   region_id_64 ,length_64 ,acmode ,flags ,return_va_64 ,return_length_64
```

C Prototype

```
int sys$expreg_64
   (struct _generic_64 *region_id_64, unsigned __int64 length_64,
    unsigned int acmode, unsigned int flags, void *(*(return_va_64)),
    unsigned __int64 *return_length_64);
```

Arguments

region_id_64

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

The region ID associated with the virtual address range to be expanded. 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.

length_64

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

Length of the virtual address space to be created. The length specified must be a multiple of CPU-specific pages.

acmode

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

Access mode associated with the call to $EXPREG_64. The access mode determines the owner mode of the pages as well as the read and write protection on the pages. The *acmode* argument is a longword containing the access mode. The $PSLDEF macro defines symbols for the four access modes.

The $EXPREG_64 service uses whichever of the following two access modes is least privileged:

- The access mode specified by the *acmode* argument
- The access mode of the caller. The protection of the pages is read/write for the resultant access mode and those more privileged.

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 controlling the characteristics of the demand-zero pages created. The *flags* argument is a longword bit vector in which each bit corresponds to a flag. The $VADEF macro and the VADEF.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.

All bits in the *flags* argument are reserved to OpenVMS for future use and should be specified as 0. The condition value SS$_IVVAFLG is returned if any bits are set.

return_va_64

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

The lowest process virtual address of a created virtual address range. The *return_va_64* argument is the 32- 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- or 64-bit reference

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

Description

The Expand Virtual Address Space service is a kernel mode service that can be called from any mode. This service adds a range of demand-zero allocation pages to a process's virtual address space for the execution of the current image. Expansion occurs at the next free available address within the specified region. The new pages, which were previously inaccessible to the process, are created as demand-zero pages.

The returned address is always the lowest virtual address in the range of pages created. The returned length is always an unsigned byte count indicating the length of the range of pages created.

Successful return status from $EXPREG_64 Expand Virtual Address service means that the specified region's virtual address space was expanded by the number of bytes specified in the *length_64* argument.

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 added before the error occurred. If no pages were added, 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

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

Related Services

$CREATE_BUFOBJ_64, $CREATE_REGION_64, $CRETVA_64, $DELETE_REGION_64, $DELTVA_64, $LCKPAG_64, $LKWSET_64, $PURGE_WS, $SETPRT_64, $ULKPAG_64, $ULWSET_64

Condition Values Returned

SS$_NORMAL

The service completed successfully.

SS$_ACCVIO

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.

SS$_INSFWSL

The process's working set limit 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$_IVREGID

An invalid region ID was specified.

SS$_IVVAFLG

An invalid flag, a reserved flag, or an invalid combination of flags and arguments was specified.

SS$_LEN_NOTPAGMULT

The *length_64* argument is not a multiple of CPU-specific pages.

SS$_NOSHPTS

The region ID of a shared page table region was specified.

SS$_REGISFULL

The specified virtual region is full.

$EXTEND

Extend File Disk Space — The Extend service increases the amount of space allocated to a disk file. This service is most useful for extending relative files and indexed files when you are doing block I/O transfers using the Write service. For additional information about this service, see the *VSI OpenVMS Record Management Services Reference Manual*.

$FAO/$FAOL

Formatted ASCII Output Services — Converts a binary value into an ASCII character string in decimal, hexadecimal, or octal notation; returns the character string in an output string; and inserts variable character-string data into an output string.

The Formatted ASCII Output with List Parameter ($FAOL) service provides an alternate method for specifying input parameters when calling the $FAO system service.
The formats for both services are shown in the Format section.
On Alpha and Integrity server systems, this service accepts 64-bit addresses.

Format

```
SYS$FAO ctrstr ,[outlen] ,outbuf ,[p1]...[pn]

SYS$FAOL ctrstr ,[outlen] ,outbuf ,[prmlst]
```

C Prototype

```
int sys$fao (void *ctrstr, unsigned short int *outlen, void *outbuf,...);

int sys$faol (void *ctrstr, unsigned short int *outlen, void *outbuf, void
 *prmlst);
```

Arguments

ctrstr

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

Control string passed to $FAO that contains the text to be output together with one or more $FAO directives. $FAO directives are used to specify repeat counts or the output field length, or both, and they are preceded by an exclamation point (!). The *ctrstr* argument is the 32- or 64-bit address of a character string descriptor pointing to the control string. The formatting of the $FAO directives is described in the Description section.

There is no restriction on the length of the control string or on the number of $FAO directives it can contain. However, if an exclamation point must appear in the output string, it must be represented in the control string by a double exclamation point (!!). A single exclamation point in the control string indicates to $FAO that the next characters are to be interpreted as FAO directives.

When $FAO processes the control string, it writes to the output buffer each character that is not part of an $FAO directive.

If the $FAO directive is valid, $FAO processes it. If the directive requires a parameter, $FAO processes the next consecutive parameter in the specified parameter list. If the $FAO directive is not valid, $FAO terminates and returns a condition value in R0.

Table 35 lists and describes the $FAO directives. Table 36 shows the $FAO output field lengths and their fill characters.

The $FAO service reads parameters from the argument list specified in the call; these arguments have the names *p1*, *p2*, *p3*, and so on, up to *p17*. Each argument specifies one parameter. Because $FAO accepts a maximum of 17 parameters in a single call, you must use $FAOL if the number of parameters exceeds 17. The $FAOL service accepts any number of parameters used with the *prmlst* argument.

outlen

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

Length in bytes of the fully formatted output string returned by $FAO. The *outlen* argument is the 32- or 64-bit address of a word containing this value.

outbuf

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

Output buffer into which $FAO writes the fully formatted output string. The *outbuf* argument is the 32- or 64-bit address of a character string descriptor pointing to the output buffer. The maximum number of bytes written is limited to 64K.

p1 to pn

OpenVMS usage:	varying_arg
type:	quadword (signed)
access:	read only
mechanism:	by value

$FAO directive parameters. The *p1* argument is a quadword containing the parameter needed by the first $FAO directive encountered in the control string, the *p2* argument is a quadword containing the parameter needed for the second $FAO directive, and so on for the remaining arguments up to *p17*. If an $FAO directive does not require a parameter, that $FAO directive is processed without reading a parameter from the argument list.

Depending on the directive, a parameter can be a value to be converted, a 32- or 64-bit address of a string to be inserted into the output string, or a length or argument count. Each directive in the control string might require a corresponding parameter or parameters.

prmlst

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

List of $FAO directive parameters to be passed to $FAOL. The *prmlst* argument is the 32- or 64-bit address of a list of longwords wherein each longword is a parameter. The $FAOL service processes these parameters sequentially as it encounters, in the control string, $FAO directives that require parameters.

The parameter list can be a data structure that already exists in a program and from which certain values are to be extracted.

Description

The Formatted ASCII Output ($FAO) service converts a binary value into an ASCII character string in decimal, hexadecimal, or octal notation and returns the character string in an output string, and inserts variable character string data into an output string.

The Formatted ASCII Output with List Parameter ($FAOL) service provides an alternate way to specify input parameters for a call to the $FAO system service. The formats for both $FAO and $FAOL are shown in the Format section.

The $FAO_S macro form uses a PUSHL instruction for all parameters (*p1* through *p17*) passed to the service; if you specify a symbolic address, it must be preceded with a number sign (#) or loaded into a register.

You can specify a maximum of 17 parameters on the $FAO macro. If more than 17 parameters are required, use the $FAOL macro.

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

$FAO Directives

$FAO directives can appear anywhere in the control string. The general format of an $FAO directive is as follows:

```
!DD
```

The exclamation point (!) specifies that the following characters are to be interpreted as an $FAO directive, and the characters *DD* represent a 1- or 2-character $FAO directive.

Note

When the characters of the $FAO directive are alphabetic, they must be uppercase.

An $FAO directive can optionally specify the following:

- A repeat count. The format is as follows:

  ```
  !n(DD)
  ```

  In this case `n` is a decimal value specifying the number of times that $FAO is to repeat the directive. If the directive requires a parameter or parameters, $FAO uses successive parameters from the parameter list for each repetition of the directive; it does not use the same parameters for each repetition. The parentheses are required syntax.

- An output field length. The format is as follows:

  ```
  !mDD
  ```

  In this case `m` is a decimal value specifying the length of the field (within the output string) into which $FAO is to write the output resulting from the directive. The length is expressed as a number of characters.

- Both a repeat count and output field length. In this case the format is as follows:

```
!n(mDD)
```

You can specify repeat counts and output field lengths as variables by using a number sign (#) in place of an absolute numeric value.

- If you specify a number sign for a repeat count, the next parameter passed to $FAO must contain the count.
- If you specify a number sign for an output field length, the next parameter must contain the length value.
- If you specify a number sign for both the output field length and for the repeat count, only one length parameter is required; each output string will have the specified length.
- If you specify a number sign for the repeat count, the output field length, or both, the parameters specifying the count, length, or both must precede other parameters required by the directive.

Numeric FAO output directives (B, W, L, Q, I, A, H, J) can include the indirect directive @. This immediately precedes the directive (@DD), and indicates that the next parameter is the address of the value instead of the value itself. This directive must be used with any directive that can produce a quadword output when using $FAOL; otherwise, $FAOL creates a 64-bit sign-extended value. This includes the Q, A, I, H, and J directives.

- The indirect directive can be used with repeat counts and output field lengths. In this case the format is as follows:

```
!n(m@DD)
```

To ensure that addresses and integers are displayed properly on the system, use the following conventions when using the $FAO and $FAOL system services:

- Identify longword data as !*x*L (where *x* is O, X, Z, U, or S).
- On Alpha and Integrity server systems, identify quadword data as !*x*Q for $FAO and $FAOL_64 or !@*x*Q for $FAOL (where *x* is O, X, Z, U, or S). Omitting the indirect directive for $FAOL can result in a 64-bit sign-extended value being created.
- If the size of an address is determined by operating system software (64-bits on Alpha and Integrity server systems), identify the address as !*x*A for $FAO and $FAOL_64 or !@*x*A for $FAOL (where *x* is O, X, Z, U, or S).
- If the size of an address is determined by the hardware architecture (64 bits on Alpha and Integrity servers, identify the address as !*x*H for $FAO and $FAOL_64 or !@*x*H for $FAOL (where *x* is O, X, Z, U, or S). Omitting the indirect directive for $FAOL can result in a 64-bit sign-extended value being created.
- If the size of an integer is determined by operating system software (32 bits on Alpha and Integrity server systems), identify the integer as !*x*I for $FAO and $FAOL_64 or !@*x*I for $FAOL (where *x* is O, X, Z, U, or S).
- If the size of an integer is determined by the hardware architecture (64 bits on Alpha and Integrity servers, identify the address as !*x*J for $FAO and $FAOL_64 or !@*x*J for $FAOL (where *x* is O,

X, Z, U, or S). Omitting the indirect directive for $FAOL can result in a 64-bit sign-extended value being created.

Table 35 lists $FAO directives.

Table 35. $FAO Directives

Directive	Description
Directives for Character String Substitution	
!AC	Inserts a counted ASCII string. It requires one parameter: the address of the string to be inserted. The first byte of the string must contain the length (in characters) of the string.
!AD	Inserts an ASCII string. It requires two parameters: the length of the string and the address of the string. Each of these parameters is a separate argument.
!AF	Inserts an ASCII string and replaces all nonprintable ASCII codes with periods (.). It requires two parameters: the length of the string and the address of the string. Each of these parameters is a separate argument.
!AS	Inserts an ASCID string. It requires one parameter: the address of a character string descriptor pointing to the string. $FAO assumes that the descriptor is a CLASS_S (static) or CLASS_D (dynamic) string descriptor. Other descriptor types might give incorrect results.
!AZ	Inserts a zero-terminated (ASCIZ) string. It requires one parameter: the address of a zero-terminated string.
Directives for Zero-Filled Numeric Conversion	
!OB	Converts a byte value to the ASCII representation of the value's octal equivalent. It requires one parameter: the value to be converted. $FAO uses only the low-order byte of the longword parameter.
!OW	Converts a word value to the ASCII representation of the value's octal equivalent. It requires one parameter: the value to be converted. $FAO uses only the low-order word of the longword parameter.
!OL	Converts a longword value to the ASCII representation of the value's octal equivalent. It requires one parameter: the value to be converted.
!OQ	Converts on Alpha and Integrity server systems a quadword to the ASCII representation of its octal equivalent. Must use the indirect directive @ to output the quadword value for $FAOL; otherwise, a 64-bit sign-extended value is written to the output buffer. It receives one parameter: the address of the value to be converted. This directive cannot be used from DCL.
!OA	Converts an address to the ASCII representation of its octal equivalent. Must use the indirect directive @ to output the quadword value for $FAOL; otherwise, a 32-bit sign-extended value is written to the output buffer. It receives one parameter: the address of the value to be converted.[1]
!OI	Converts an integer to the ASCII representation of its octal equivalent. Must use the indirect directive @ to output the quadword value for $FAOL; otherwise, a 32-bit sign-extended value is written to the output buffer. It receives one parameter: the address of the value to be converted.[1]
!OH	Converts an address to the ASCII representation of its octal equivalent. Must use the indirect directive @ to output the quadword value for $FAOL; otherwise, a 64-bit sign-extended value is written to the output buffer. It receives one parameter: the address of the value to be converted.[2]

Directive	Description
!OJ	Converts an integer to the ASCII representation of its octal equivalent. Must use the indirect directive @ to output the quadword value for $FAOL; otherwise, a 64-bit sign-extended value is written to the output buffer. It receives one parameter: the address of the value to be converted.[2]
!XB	Converts a byte value to the ASCII representation of the value's hexadecimal equivalent. It requires one parameter: the value to be converted. $FAO uses only the low-order byte of the longword parameter.
!XW	Converts a word value to the ASCII representation of the value's hexadecimal equivalent. It requires one parameter: the value to be converted. $FAO uses only the low-order word of the longword parameter.
!XL	Converts a longword value to the ASCII representation of the value's hexadecimal equivalent. It requires one parameter: the value to be converted.
!XQ	Converts a quadword to the ASCII representation of its hexadecimal equivalent. Must use the indirect directive @ to output the quadword value for $FAOL; otherwise, a 64-bit sign-extended value is written to the output buffer.
!XA	Converts an address to the ASCII representation of its hexadecimal equivalent. Must use the indirect directive @ to output the quadword value for $FAOL; otherwise, a 32-bit sign-extended value is written to the output buffer. It receives one parameter: the address of the value to be converted.[1]
!XI	Converts an integer to the ASCII representation of its hexadecimal equivalent. Must use the indirect directive @ to output the quadword value for $FAOL; otherwise, a 32-bit sign-extended value is written to the output buffer. It receives one parameter: the address of the value to be converted.[1]
!XH	Converts an address to the ASCII representation of its hexadecimal equivalent. Must use the indirect directive @ to output the quadword value for $FAOL; otherwise, a 64-bit sign-extended value is written to the output buffer. It receives one parameter: the address of the value to be converted.[2]
!XJ	Converts an integer to the ASCII representation of its hexadecimal equivalent. Must use the indirect directive @ to output the quadword value for $FAOL; otherwise, a 64-bit sign-extended value is written to the output buffer. It receives one parameter: the address of the value to be converted.[2]
!ZB	Converts an unsigned byte value to the ASCII representation of the value's decimal equivalent. It requires one parameter: the value to be converted. $FAO uses only the low-order byte of the longword parameter.
!ZW	Converts an unsigned word value to the ASCII representation of the value's decimal equivalent. It requires one parameter: the value to be converted. $FAO uses only the low-order word of the longword parameter.
!ZL	Converts an unsigned longword value to the ASCII representation of the value's decimal equivalent. It requires one parameter: the value to be converted.
!ZQ	Converts an unsigned quadword to the ASCII representation of its decimal equivalent. Must use the indirect directive @ to output the quadword value for $FAOL; otherwise, a 64-bit zero-extended value is written to the output buffer. It receives one parameter: the address of the value to be converted.
!ZA	Converts an unsigned address to the ASCII representation of its decimal equivalent. Must use the indirect directive @ to output the quadword value for $FAOL; otherwise,

Directive	Description
	a 32-bit value is written to the output buffer. It receives one parameter: the address of the value to be converted.[1]
!ZI	Converts an unsigned integer to the ASCII representation of its decimal equivalent. Must use the indirect directive @ to output the quadword value for $FAOL; otherwise, a 32-bit value is written to the output buffer. It receives one parameter: the address of the value to be converted.[1]
!ZH	Converts an unsigned address to the ASCII representation of its decimal equivalent. Must use the indirect directive @ to output the quadword value for $FAOL; otherwise, a 64-bit zero-extended value is written to the output buffer. It receives one parameter: the address of the value to be converted.[2]
!ZJ	Converts an unsigned integer to the ASCII representation of its decimal equivalent. Must use the indirect directive @ to output the quadword value for $FAOL; otherwise, a 64-bit zero-extended value is written to the output buffer. It receives one parameter: the address of the value to be converted.[2]
Directives for Blank-Filled Numeric Conversion	
!UB	Converts an unsigned byte value to the ASCII representation of the value's decimal equivalent. It requires one parameter: the value to be converted. $FAO uses only the low-order byte of the longword parameter.
!UW	Converts an unsigned word value to the ASCII representation of the value's decimal equivalent. It requires one parameter: the value to be converted. $FAO uses only the low-order word of the longword parameter.
!UL	Converts an unsigned longword value to the ASCII representation of the value's decimal equivalent. It requires one parameter: the value to be converted.
!UQ	Converts an unsigned quadword to the ASCII representation of its decimal equivalent. Must use the indirect directive @ to output the quadword value for $FAOL; otherwise, a 64-bit value is written to the output buffer. It receives one parameter: the address of the value to be converted.[1]
!UA	Converts an unsigned address to the ASCII representation of its decimal equivalent. Must use the indirect directive @ to output the quadword value for $FAOL; otherwise, a 32-bit value is written to the output buffer. It receives one parameter: the address of the value to be converted.[1]
!UI	Converts an unsigned integer to the ASCII representation of its decimal equivalent. Must use the indirect directive @ to output the quadword value for $FAOL; otherwise, a 32-bit value is written to the output buffer. It receives one parameter: the address of the value to be converted.[1]
!UH	Converts an unsigned address to the ASCII representation of its decimal equivalent. Must use the indirect directive @ to output the quadword value for $FAOL; otherwise, a 64-bit value is written to the output buffer. It receives one parameter: the address of the value to be converted.[2]
!UJ	Converts an unsigned integer to the ASCII representation of its decimal equivalent. Must use the indirect directive @ to output the quadword value for $FAOL; otherwise, a 64-bit value is written to the output buffer. It receives one parameter: the address of the value to be converted.[2]
!SB	Converts a signed byte value to the ASCII representation of the value's decimal equivalent. It requires one parameter: the value to be converted. $FAO uses only the low-order byte of the longword parameter.

Directive	Description
!SW	Converts a signed word value to the ASCII representation of the value's decimal equivalent. It requires one parameter: the value to be converted. $FAO uses only the low-order word of the longword parameter.
!SL	Converts a signed longword value to the ASCII representation of the value's decimal equivalent. It requires one parameter: the value to be converted.
!SQ	Converts on Alpha and Integrity server systems a signed quadword to the ASCII representation of its decimal equivalent. Must use the indirect directive @ to output the quadword value for $FAOL; otherwise, a 32-bit value is written to the output buffer. It receives one parameter: the address of the value to be converted. This directive cannot be used from DCL.
!SA	Converts a signed address to the ASCII representation of its decimal equivalent. Must use the indirect directive @ to output the quadword value for $FAOL; otherwise, a 32-bit value is written to the output buffer. It receives one parameter: the address of the value to be converted.[1]
!SI	Converts a signed integer to the ASCII representation of its equivalent. Must use the indirect directive @ to output the quadword value for $FAOL; otherwise, a 32-bit value is written to the output buffer. It receives one parameter: the address of the value to be converted.[1]
!SH	Converts a signed address to the ASCII representation of its decimal equivalent. Must use the indirect directive @ to output the quadword value for $FAOL; otherwise, a 32-bit value is written to the output buffer. It receives one parameter: the address of the value to be converted.[2]
!SJ	Converts a signed integer to the ASCII representation of its decimal equivalent. Must use the indirect directive @ to output the quadword value for $FAOL; otherwise, a 32-bit value is written to the output buffer. It receives one parameter: the address of the value to be converted.[2]
Directives for Output String Formatting	
!/	Inserts a new line, that is, a carriage return and line feed. It takes no parameters.
!_	Inserts a tab. It takes no parameters.
!^	Inserts a form feed. It takes no parameters.
!!	Inserts an exclamation point. It takes no parameters.
!%S	Inserts the letter *S* if the most recently converted numeric value is not 1. An uppercase *S* is inserted if the character before the !%S directive is an uppercase character; a lowercase *s* is inserted if the character is lowercase.
!%T	Inserts the system time. It takes one parameter: the address of a quadword time value to be converted to ASCII. If you specify 0, the current system time is inserted.
!%U	Converts a longword integer UIC to a standard UIC specification in the format [xxx,yyy], where *xxx* is the group number and *yyy* is the member number. It takes one parameter: a longword integer. The directive inserts the surrounding brackets ([ ]) and comma (,).
!%I	Converts a longword to the appropriate alphanumeric identifier. If the longword represents a UIC, surrounding brackets ([ ]) and comma (,) are added as necessary. If no identifier exists and the longword represents a UIC, the longword is formatted as in !%U. Otherwise it is formatted as in !XL with a preceding !%X added to the formatted result.

Directive	Description
!%D	Inserts the system date and time. It takes one parameter: the address of a quadword time value to be converted to ASCII. If you specify 0, the current system date and time is inserted.
!n%C	Inserts a character string when the most recently evaluated argument has the value *n*. (Recommended for use with multilingual products.)
!%E	Inserts a character string when the value of the most recently evaluated argument does not match any preceding !n%C directives. (Recommended for use with multilingual products.)
!%F	Makes the end of a plurals statement.
!n<	See description of next directive (!>).
!>	This directive and the preceding one (!n<) are used together to define an output field width of "n" characters within which all data and directives to the right of !n< and to the left of !> are left-justified and blank-filled. It takes no parameters.
!n*c	Repeats the character `c` in the output string `n` times.
Directives for Parameter Interpretation	
!–	Causes $FAO to reuse the most recently used parameter in the list. It takes no parameters.
!+	Causes $FAO to skip the next parameter in the list. It takes no parameters.

[1]Determined by the operating system. On Alpha or Integrity server systems, this is 32 bits.

[2]Determined by the hardware architecture. On Alpha and Integrity server systems, this is 64 bits.

Table 36 shows the $FAO output field lengths and their fill characters.

Table 36. $FAO Output Field Lengths and Fill Characters

Conversion/ Substitution Type	Default Length of Output Field	Action When Explicit Output Field Length Is Longer Than Default	Action When Explicit Output Field Length Is Shorter Than Default
Hexadecimal			
Byte Word Longword Quadword	2 (zero-filled) 4 (zero-filled) 8 (zero-filled) 16 (zero-filled)	ASCII result is right-justified and blank-filled to the specified length.	ASCII result is truncated on the left.
Octal			
Byte Word Longword Quadword	3 (zero-filled) 6 (zero-filled) 11 (zero-filled) 22 (zero-filled)	Hexadecimal or octal output is always zero-filled to the default output field length, then blank-filled to specified length.	
Signed or unsigned decimal			
Signed or unsigned decimal	As many characters as necessary	ASCII result is right-justified and blank-filled to the specified length.	Signed and unsigned decimal output fields and completely filled with asterisks (*).
Unsigned zero-filled decimal			

Conversion/ Substitution Type	Default Length of Output Field	Action When Explicit Output Field Length Is Longer Than Default	Action When Explicit Output Field Length Is Shorter Than Default
Unsigned zero-filled decimal	As many characters as necessary	ASCII result is right-justified and zero-filled to the specified length.	
ASCII string substitution			
ASCII string substitution	Length of input character string	ASCII string is left-justified and blank-filled to the specified length.	ASCII string is truncated on the right.

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

SS$_BUFFEROVF

The service completed successfully. The formatted output string overflowed the output buffer and has been truncated.

SS$_NORMAL

The service completed successfully.

SS$_ACCVIO

The *ctrstr*, *p1* through *pn*, or *prmlst* arguments cannot be read, or the *outlen* argument cannot be written (it can specify 0).

SS$_BADPARAM

You specified an invalid directive in the $FAO control string.

SS$_OVERMAXARG

Maximum parameter count exceeded.

$FAO Control String Examples

Each of the following examples shows an $FAO control string with several directives, parameters defined as input for the directives, and the calls to $FAO to format the output strings.

Each example is accompanied by notes. These notes show the output string created by the call to $FAO and describe in more detail some considerations for using directives. The sample output strings show the underscore character (_) for each space in all places where $FAO output contains multiple spaces.

Each of the first 10 examples (numbered 1 through 10) refers to the following output fields but does not include these fields within the examples.

```
int     status,                     /* Status of system calls */
        outlen;                     /* Length of output string from $FAO */
char    out_buffer[80];             /* Buffer for $FAO output */

$DESCRIPTOR(out_desc, out_buffer);  /* Descriptor for out_buffer */
```

Each of the 10 examples also assumes the caller of each example will check the returned status, and write the output string produced by $FAO if no error occurred. The following code fragment shows how the example call may be made, and the resultant string output to the user's terminal.

```
#include <stdio.h>
#include <stsdef.h>
#include <lib$routines.h>
        .
        .
        .
    status = example();

    /* Immediately signal (and quit) if error occurred */
    if ((status & STS$M_SUCCESS) == 0)  lib$signal(status);

    /* FAO directive succeeded, output resultant string */
    out_buffer[outlen] = '\0';       /* add string terminator to buffer */
    puts(out_buffer);                /* output the result */
```

The final example (numbered 11) shows a segment of a VSI Fortran for OpenVMS program used to output an ASCII string.

1. ```
   /* SYS$FAO example - illustrating !AC, !AS, !AD, and !/ directives */
   #include <string.h>
   #include <descrip.h>
   #include <starlet.h>

   /* MACRO and typedef for counted ASCII strings... */
   typedef struct {char len, str[25];} ASCIC;
   #define ASCIC_STRING(name, string) ASCIC name = {sizeof(string) - 1, string}

   int example()
   {
       char    *nod = "Nod";               /* Normal "C" string */
       const int nodlen = strlen(nod); /* Length of "Nod" without '\0' */
       static  ASCIC_STRING(winken, "Winken");
       static  $DESCRIPTOR(blinken, "Blinken");
       static  $DESCRIPTOR(fao_desc, "!/Sailors: !AC !AS !AD");

       return (sys$fao(&fao_desc,  /* Control string for $FAO */
                       &outlen,    /* Pointer for length of output string */
                       &out_desc,  /* Descriptor for output buffer */
                       &winken,    /* P1 - Counted ASCII string */
                       &blinken,   /* P2 - ASCII string descriptor */
                       nodlen,     /* P3 - Length of ASCII string */
                       nod));      /* P4 - ASCII string */
   }
   ```

   FAO writes the following string into the output buffer:

   ```
   <CR><KEY>(LF\TEXT)Sailors: Winken Blinken Nod
   ```
   ```

The !/ directive provides a carriage-return/line-feed character (shown as <CR><KEY>(LF\TEXT)) for terminal output.

The !AC directive requires the address of a counted ASCII string (*p1* argument).

The !AS directive requires the address of a character string descriptor (*p2* argument).

The !AD directive requires two parameters: the length of the string to be substituted (*p3* argument) and its address (*p4* argument).

2.
```
/*
** SYS$FAO example - illustrating !! and !AS directives,
** repeat count, and output field length
*/
#include <descrip.h>
#include <starlet.h>

int example()
{
    static $DESCRIPTOR(jones, "Jones");
    static $DESCRIPTOR(harris, "Harris");
    static $DESCRIPTOR(wilson, "Wilson");
    static $DESCRIPTOR(fao_desc, "Unable to locate !3(8AS)!!");

    return(sys$fao(&fao_desc,   /* Control string for $FAO */
                   &outlen,     /* Pointer for length of output string */
                   &out_desc,   /* Descriptor for output buffer */
                   &jones,      /* P1 - ASCII string descriptor */
                   &harris,     /* P2 - ASCII string descriptor */
                   &wilson));   /* P3 - ASCII string descriptor */
}
```

$FAO writes the following string into the output buffer:

```
Unable to locate Jones___Harris__Wilson__!
```

The !3(8AS) directive contains a repeat count: three parameters (addresses of character string descriptors) are required. $FAO left-justifies each string into a field of eight characters (the output field length specified).

The double exclamation point directive (!!) supplies a literal exclamation point (!) in the output.

If the directive were specified without an output field length, that is, if the directive were specified as !3(AS), the three output fields would be concatenated, as follows:

```
Unable to locate JonesHarrisWilson!
```

3.
```
/* SYS$FAO example - illustrating !UL, !XL, and !SL directives */
#include <descrip.h>
#include <starlet.h>

int example()
{
    int val1 = 200,             /* Values */
        val2 = 300,             /*  for   */
        val3 = -400;            /*   $FAO */

    static $DESCRIPTOR(fao_desc,
                       "Values !UL (Decimal) !XL (Hex) !SL (Signed)");

    return(sys$fao(&fao_desc,   /* Control string for $FAO */
```

```
                    &outlen,     /* Pointer for length of output string */
                    &out_desc,   /* Descriptor for output buffer */
                    val1,        /* P1 - longword value */
                    val2,        /* P2 - longword value */
                    val3));      /* P3 - longword value */
}
```

$FAO writes the following string to the output buffer:

```
Values 200 (Decimal) 0000012C (Hex) -400 (Signed)
```

The longword value 200 is converted to decimal, the value 300 is converted to hexadecimal, and the value --400 is converted to signed decimal. The ASCII results of each conversion are placed in the appropriate position in the output string.

Note that the hexadecimal output string has eight characters and is zero-filled to the left. This is the default output length for hexadecimal longwords.

4.
```
/* SYS$FAOL example - illustrating !UL, !XL, and !SL directives */
#include <descrip.h>
#include <starlet.h>

int example()
{
    static int values[3] = {200, 300, -400};   /* Parameters for $FAOL */
    static $DESCRIPTOR(fao_desc,
                       "Values !UL (Decimal) !XL (Hex) !SL (Signed)");

    return(sys$faol(&fao_desc,  /* Control string for $FAO */
                    &outlen,    /* Pointer for length of output string */
                    &out_desc,  /* Descriptor for output buffer */
                    values));   /* Parameter list - longwords */
}
```

$FAOL writes the following string to the output buffer:

```
Values 200 (Decimal) 0000012C (Hex) -400 (Signed)
```

The results are the same as the results of Example 3; however, unlike the $FAO directive, which requires each parameter on the call to be specified, the $FAOL directive points to a list of consecutive longwords, which $FAO reads as parameters.

5.
```
/* SYS$FAOL example - illustrating !UB, !XB, and !SB directives */
#include <descrip.h>
#include <starlet.h>

int example()
{
    static int values[3] = {200, 300, -400};   /* Parameters for $FAOL */
    static $DESCRIPTOR(fao_desc,
                       "Values !UB (Decimal) !XB (Hex) !SB (Signed)");

    return(sys$faol(&fao_desc,  /* Control string for $FAO */
                    &outlen,    /* Pointer for length of output string */
                    &out_desc,  /* Descriptor for output buffer */
                    values));   /* Parameter list - longwords */
}
```

$FAO writes the following output string:

```
Values 200 (Decimal) 2C (Hex) 112 (Signed)
```

The input parameters are the same as those for Example 4; however, the control string (fao_desc) specifies that byte values are to be converted. $FAO uses the low-order byte of each longword parameter passed to it. The high-order three bytes are not evaluated. Compare these results with the results of Example 4.

6.
```
/*
** SYS$FAO example - illustrating !XW, !ZW, and !- directives,
** repeat count, and output field length
*/
#include <descrip.h>
#include <starlet.h>

int example()
{
    static $DESCRIPTOR(fao_desc,
                       "Hex: !2(6XW) Zero-filled Decimal: !2(-)!2(7ZW)");

    return(sys$fao(&fao_desc,   /* Control string for $FAO */
                   &outlen,     /* Pointer for length of output string */
                   &out_desc,   /* Descriptor for output buffer */
                   10000,       /* P1 - longword value */
                   9999));      /* P2 - longword value */
}
```

$FAO writes the following string to the output buffer:

```
Hex:___2710__270F Zero-filled Decimal: 00100000009999
```

Each of the directives !2(6XW) and !2(7ZW) contains repeat counts and output lengths. First, $FAO performs the !XW directive twice, using the low-order word of the numeric parameters passed. The output length specified is two characters longer than the default output field width of hexadecimal word conversion, so two spaces are placed between the resulting ASCII strings.

The !-- directive causes $FAO to back up over the parameter list. A repeat count is specified with the directive so that $FAO skips back over two parameters; then, it uses the same two parameters for the !ZW directive. The !ZW directive causes the output string to be zero-filled to the specified length (in this example, seven characters). Thus, there are no spaces between the output fields.

7.
```
/*
** SYS$FAOL example - illustrating !AS, !UB, !%S, and !- directives,
** and variable repeat count
*/
#include <descrip.h>
#include <starlet.h>

/* Layout of argument list for examples */
typedef struct {void    *desc;          /* ASCII string descriptor */
                int     arg[4];         /* Longword arguments */
                } LIST;

$DESCRIPTOR(fao_desc, "!AS received !UB argument!%S: !-!#(4UB)");

int example_a()
{
    static $DESCRIPTOR(orion, "ORION");
    static LIST
        list_a = {&orion,       /* Address of descriptor */
                  3,            /* Number of arguments */
                  10,           /* Argument 1 */
                  123,          /* Argument 2 */
                  210};         /* Argument 3 */
```

```
    return(sys$faol(&fao_desc,  /* Control string for $FAO */
                    &outlen,    /* Pointer for length of output string */
                    &out_desc,  /* Descriptor for output buffer */
                    &list_a));  /* Parameter list */
}

int example_b()
{
    static $DESCRIPTOR(lyra, "LYRA");
    static LIST
        list_b = {&lyra,        /* ASCII descriptor cast as an (int) */
                  1,            /* Number of arguments */
                  255};         /* Argument 1 */

    return(sys$faol(&fao_desc,  /* Control string for $FAO */
                    &outlen,    /* Pointer for length of output string */
                    &out_desc,  /* Descriptor for output buffer */
                    &list_b));  /* Parameter list */
}
```

In example A, $FAO writes the following string to the output buffer:

```
ORION received 3 arguments:___10 123 210
```

In example B, $FAO writes the following string to the output buffer:

```
LYRA received 1 argument:__255
```

In each of the examples, the parameter list argument points to a different parameter list; each list contains, in the first longword, the address of a character string descriptor. The second longword begins an argument list, with the number of arguments remaining in the list. The control string uses this second longword twice: first to output the value contained in the longword, and then to provide the repeat count to output the number of arguments in the list (the !- directive indicates that $FAO should reuse the parameter).

The !%S directive provides a conditional plural. When the last value converted has a value not equal to 1, $FAO outputs the character *s*; if the value is a 1 (as in Example B), $FAO does not output the character *s*. $FAO outputs the plural character in lowercase since the preceding character was in lowercase.

The output field length defines a width of four characters for each byte value converted, to provide spacing between the output fields.

8.
```
/*
** SYS$FAO example - illustrating !n*c (repeat character)
** and !%D (date/time) directives
*/
#include <descrip.h>
#include <starlet.h>

int example()
{
    static $DESCRIPTOR(fao_desc, "!5*> The time is now: !%D");

    return(sys$fao(&fao_desc,   /* Control string for $FAO */
                   &outlen,     /* Pointer for length of output string */
                   &out_desc,   /* Descriptor for output buffer */
                   0));         /* P1 - time value, 0 = current time */
}
```

$FAO writes the following string to the output buffer:

```
>>>>> The time is now: dd-mmm-yyyy hh:mm:ss.cc
```

where:

dd	is the day of the month
mmm	is the month
yyyy	is the year
hh:mm:ss.cc	is the time in hours, minutes, seconds, and hundredths of a second

The !5*> directive requests $FAO to write five greater-than (>) characters into the output string. Because there is a space after the directive, $FAO also writes a space after the greater-than characters on output.

The !%D directive requires the address of a quadword time value, which must be in the system time format; however, when the address of the time value is specified as 0, $FAO uses the current date and time. For a detailed description of the ASCII date and time string returned, see the discussion of the Convert Binary Time to ASCII String ($ASCTIM) system service.

9.
```
/*
** SYS$FAO example - illustrating !%D and !%T (with output field lengths),
** and !n directive with variable repeat count
*/
#include <descrip.h>
#include <starlet.h>

int example()
{
    static $DESCRIPTOR(fao_desc, "Date: !11%D!#*_Time: !5%T");

    return(sys$fao(&fao_desc,   /* Control string for $FAO */
                   &outlen,     /* Pointer for length of output string */
                   &out_desc,   /* Descriptor for output buffer */
                   0,           /* P1 - time value, 0 = current time */
                   5,           /* P2 - Number of underscores */
                   0));         /* P3 - time value, 0 = current time */
}
```

$FAO writes the following string to the output buffer:

```
Date: dd-mmm-yyyy_____Time: hh:mm
```

An output length of 11 bytes is specified with the !%D directive so that $FAO truncates the time from the date and time string, and outputs only the date.

The !#*_ directive requests that the underscore character (_) be repeated the number of times specified by the next parameter. Because *p2* is specified as 5, five underscores are written into the output string.

The !%T directive normally returns the full system time. The !5%T directive provides an output length for the time; only the hours and minutes fields of the time string are written into the output buffer.

10.
```
/*
** SYS$FAO example - illustrating !< and !> (define field width),
** !AC, and !UL directives
*/
#include <descrip.h>
```

```
#include <starlet.h>

/* MACRO and typedef for counted ASCII strings... */
typedef struct {char len, str[25];} ASCIC;
#define ASCIC_STRING(name, string) ASCIC name = {sizeof(string) - 1, string}

$DESCRIPTOR(fao_desc, "!32<Variable: !AC Value: !UL!>Total:!7UL");

int example_a()
{
    int     val_a = 334,        /* Current value for variable */
            tot_a = 6554;       /* Current total for variable */

    static ASCIC_STRING(var_a, "Inventory");    /* Counted ASCII string */

    return(sys$fao(&fao_desc,   /* Control string for $FAO */
                   &outlen,     /* Pointer for length of output string */
                   &out_desc,   /* Descriptor for output buffer */
                   &var_a,      /* P1 - Variable name */
                   val_a,       /* P2 - Value for variable */
                   tot_a));     /* P3 - Total for variable */
}

int example_b()
{
    int val_b = 280,            /* Current value for variable */
        tot_b = 10750;          /* Current total for variable */

    static ASCIC_STRING(var_b, "Sales");        /* Counted ASCII string */

    return(sys$fao(&fao_desc,   /* Control string for $FAO */
                   &outlen,     /* Pointer for length of output string */
                   &out_desc,   /* Descriptor for output buffer */
                   &var_b,      /* P1 - Variable name */
                   val_b,       /* P2 - Value for variable */
                   tot_b));     /* P3 - Total for variable */
}
```

In example A, $FAO writes the following string to the output buffer:

```
Variable: Inventory Value: 334__Total:___6554
```

n example B, $FAO writes the following string to the output buffer:

```
Variable: Sales Value: 280______Total:__10750
```

The !25< directive requests an output field width of 25 characters; the end of the field is delimited by the !> directive. Within the field defined are two directives, !AC and !UL. The strings substituted by these directives can vary in length, but the entire field always has 25 characters.

The !7UL directive formats the longword passed in each example (*p2* argument) and right-justifies the result in a 7-character output field.

11.
```
INTEGER STATUS,
2       SYS$FAO,
2       SYS$FAOL

! Resultant string
CHARACTER*80 OUTSTRING
INTEGER*2    LEN
! Array for directives in $FAOL
INTEGER*4    PARAMS(2)

! File name and error number
```

```
CHARACTER*80 FILE
INTEGER*4    FILE_LEN,
2            ERROR
! Descriptor for $FAOL
INTEGER*4    DESCR(2)

! These variables would generally be set following an error
FILE = '[BOELITZ]TESTING.DAT'
FILE_LEN = 18
ERROR = 25

! Call $FAO
STATUS = SYS$FAO ('File !AS aborted at error !SL',
2                 LEN,
2                 OUTSTRING,
2                 FILE(1:FILE_LEN),
2                 %VAL(ERROR))
IF (.NOT. STATUS) CALL LIB$SIGNAL (%VAL(STATUS))

TYPE *,'From SYS$FAO:'
TYPE *,OUTSTRING (1:LEN)

! Set up descriptor for filename
DESCR(1) = FILE_LEN    ! Length
DESCR(2) = %LOC(FILE)  ! Address
! Set up array for directives
PARAMS(1) = %LOC(DESCR) ! File name
PARAMS(2) = ERROR       ! Error number
! Call $FAOL
STATUS = SYS$FAOL ('File !AS aborted at error !SL',
2                  LEN,
2                  OUTSTRING,
2                  PARAMS)
IF (.NOT. STATUS) CALL LIB$SIGNAL (%VAL(STATUS))

TYPE *,'From SYS$FAOL:'
TYPE *,OUTSTRING (1:LEN)

END
```

This example shows a segment of a VSI Fortran for OpenVMS program used to output the following string:

```
FILE [BOELITZ]TESTING.DAT ABORTED AT ERROR 25
```

$FAOL_64 (Alpha and Integrity servers)

Formatted ASCI Output with List Parameter for 64-Bit Virtual Addresses — On Alpha and Integrity server systems, converts a binary value into an ASCII character string in decimal, hexadecimal, or octal notation; returns the character string in an output string; and inserts variable character-string data into an output string. $FAOL_64 interprets the parameter list as a list of quadwords rather than a list of longwords. In all other respects, $FAOL_64 is identical to $FAOL. For all other information about the $FAOL_64 service, refer to the description of $FAO/$FAOL in this manual. This service accepts 64-bit addresses.

Format

```
SYS$FAOL_64 ctrstr_64 [,outlen_64 [,outbuf_64 [,quad_prmlst_64]]]
```

C Prototype

```
int sys$faol_64
   (void *ctrstr_64, unsigned short int *outlen_64, void *outbuf_64,
    void *quad_prmlst_64);
```

Arguments

ctrstr_64

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

The 32- or 64-bit address of the control string (64-bit or 32-bit string descriptor).

outlen_64

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

The 32- or 64-bit address of the quadword that contains the output length, in bytes, of the fully formatted output string.

outbuf_64

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

The 32- or 64-bit address of a character string descriptor that points to the output buffer into which $FAOL_64 writes the fully formatted output string.

quad_prmlst_64

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

The 32- or 64-bit address of a quadword-aligned array of quadword FAO arguments.

$FILESCAN

Scan String for File Specification — Searches a string for a file specification and parses the components of that file specification.

Format

```
SYS$FILESCAN srcstr ,valuelst ,[fldflags] ,[auxout] ,[retlen]
```

C Prototype

```
int sys$filescan
   (void *srcstr, void *valuelst, unsigned  int *fldflags, void *auxout,
    unsigned short int *retlen);
```

Arguments

srcstr

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

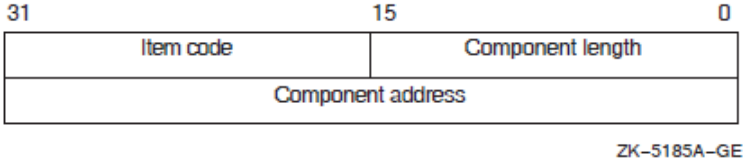
String to be searched for the file specification. The *srcstr* argument is the address of a descriptor pointing to this string.

valuelst

OpenVMS usage:	item_list_2
type:	longword (unsigned)
access:	modify
mechanism:	by reference

Item list specifying which components of the file specification are to be returned by $FILESCAN. The components are the full node specification, primary node name, primary node's access control, secondary node information, device, directory, file name, file type, and version number. The *itmlst* argument is the address of a list of item descriptors wherein each item descriptor specifies one component. The list of item descriptors is terminated by a longword of 0.

The following diagram depicts a single item descriptor.



The following table defines the item descriptor fields.

Descriptor Field	Definition
Component length	A word in which $FILESCAN writes the length (in characters) of the requested component. If $FILESCAN does not locate the component, it returns the value 0 in this field and in the component address field and returns the SS$_NORMAL condition value.
Item code	A user-supplied, word-length symbolic code that specifies the component desired. The $FSCNDEF macro defines the item codes.

Descriptor Field	Definition
Component address	A longword in which $FILESCAN writes the starting address of the component. This address points to a location in the input string itself. If $FILESCAN does not locate the component, it returns the value 0 in this field (see item code FSCN$_NAME for exception) and 0 in the component length field, and returns the SS$_NORMAL condition value.

fldflags

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

Longword flag mask in which $FILESCAN sets a bit for each file specification component found in the input string. The *fldflags* argument is the address of this longword flag mask.

The $FSCNDEF macro defines a symbolic name for each significant flag bit. The following table shows the file specification component that corresponds to the symbolic name of each flag bit.

Symbolic Name	Corresponding Component
FSCN$V_DEVICE	Device name
FSCN$V_DIRECTORY	Directory name
FSCN$V_NAME	File name
FSCN$V_NODE	Node name
FSCN$V_NODE_ACS	Access control string of primary node
FSCN$V_NODE_PRIMARY	Primary (first) node name
FSCN$V_NODE_SECONDARY	Secondary (additional) node information
FSCN$V_ROOT	Root directory name string
FSCN$V_TYPE	File type
FSCN$V_VERSION	Version number

The *fldflags* argument is optional. When you want to know which components of a file specification are present in a string but do not need to know the contents or length of these components, specify *fldflags* instead of *valuelst*.

auxout

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

Auxiliary output buffer. The *auxout* argument is the address of a character-string descriptor pointing to the auxiliary buffer.

When you specify an auxiliary output buffer, $FILESCAN copies the entire source string, with quotation information reduced and simplified for only the primary node, into the auxiliary output buffer.

When the auxiliary output buffer is provided, all addresses returned in the item list point to locations in the auxiliary output buffer.

retlen

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

Length of the auxiliary buffer. The *retlen* argument is the address of a word into which $FILESCAN writes the length of the auxiliary buffer name string.

Item Codes

FSCN$_DEVICE

When you specify FSCN$_DEVICE, $FILESCAN returns the length and starting address of the device name. The device name includes the single colon (:).

FSCN$_DIRECTORY

When you specify FSCN$_DIRECTORY, $FILESCAN returns the length and starting address of the directory name. The directory name includes the brackets ([ ]) or angle brackets (< >).

FSCN$_FILESPEC

When you specify FSCN$_FILESPEC, $FILESCAN returns the length and starting address of the full file specification. The full file specification contains the node, device, directory, name, type, and version.

FSCN$_NAME

When you specify FSCN$_NAME, $FILESCAN returns the length and starting address of the file name. The file name includes no syntactical elements.

$FILESCAN also returns the length and starting address of a quoted file specification following a node specification (as in the specification NODE:: “FILE-SPEC”). The beginning and ending quotation marks are included.

FSCN$_NODE

When you specify FSCN$_NODE, $FILESCAN returns the length and starting address of the full node specification. The full node specification includes the primary node name, the primary node's access control string, any secondary node information, and the final double colon (::).

FSCN$_NODE_ACS

When you specify FSCN$_NODE_ACS, $FILESCAN returns the length and starting address of the primary access control string. If multiple nodes are specified, the primary access control string represents the control information (if present) for the first node specified. The primary access control string does not contain the double colon (::), but does contain the double quotes.

FSCN$_NODE_PRIMARY

When you specify FSCN$_NODE_PRIMARY, $FILESCAN returns the length and starting address of the primary node name. If multiple nodes are specified, the primary node name represents the first node

specification. The node name does not include the double colon (::) or any access control information. If an auxiliary output buffer is specified, quotation information is reduced and simplified for only the primary node.

FSCN$_NODE_SECONDARY

When you specify FSCN$_NODE_SECONDARY, $FILESCAN returns the length and starting address of any secondary node information. The secondary node string contains any node information referring to additional nodes, including the final double colon (::), as well as any access control strings (if present) for the additional nodes.

FSCN$_ROOT

When you specify FSCN$_ROOT, $FILESCAN returns the length and starting address of the root directory string. The root directory name string includes the brackets ([ ]) or angle brackets (< >).

FSCN$_TYPE

When you specify FSCN$_TYPE, $FILESCAN returns the length and starting address of the file type. The file type includes the preceding period (.).

FSCN$_VERSION

When you specify FSCN$_VERSION, $FILESCAN returns the length and starting address of the file version number. The file version number includes the preceding period (.) or semicolon (;) delimiter.

Description

The Scan String for File Specification service searches a string for a file specification and parses the components of that file specification. When $FILESCAN locates a partial file specification (for example, DISK:[FOO]), it returns the length and starting address of those components that were requested in the item list and were found in the string. If a component was requested in the item list but not found in the string, $FILESCAN returns a length of 0 and an address of 0 to the component address field in the item descriptor for that component (see item code FSCB$_NAME for exception).

The information returned about all of the individual components describes the entire contiguous file specification string. For example, to extract only the file name and file type from a full file specification string, you can add the length of these two components and use the address of the first component (file name). However, the specific node name and node control strings extracted using the FSCN$_NODE_PRIMARY and FSCN$_NODE_ACS item codes cannot be recombined because the double colon (::) is not included in either string.

If an auxiliary output buffer is provided, $FILESCAN copies the entire source string, removing and reducing quotation marks from the primary node name.

The $FILESCAN service does not perform comprehensive syntax checking. Specifically, it does not check that a component has a valid length.

However, $FILESCAN does check for the following information:

- The component must have required syntactical elements; for example, a directory component must be enclosed in brackets ([]), and a node name must be followed by an unquoted double colon (::).
- The component must not contain invalid characters. Invalid characters are specific to each component. For example, a comma (,) is a valid character in a directory component but not in a file type component.

- Spaces, tabs, and carriage returns are permitted within quoted strings, but are invalid anywhere else.
- If a node name contains a space, tab, double quote ("), or double colon (::), then the node name must be quoted.

The node component of a file specification contains one or more node specifications. A node specification is a node name, followed by an optional access control string, followed by a double colon (::). A node name is either a standard name or a quoted name. If the node name contains quotation marks, the quotes must be doubled ("") and the entire name quoted. For example, the node abc "def" would be represented as "abc""def""". An access control string is a quoted string containing a user name, an optional password, and an optional account name. Note that the double quotes are only used to differentiate between a primary node name containing embedded quotes and the access control string. See the Example section for the proper use of this syntax.

Invalid characters are treated as terminators. For example, if $FILESCAN encounters a space within a file name component, it assumes that the space terminates the full file specification string.

For node names, a space, tab, double quote ("), and comma (,) are treated as terminators and must be quoted if they are part of the node name. In addition, the double colon (::) and the trailing colon (for example, NODE:) are treated as terminators and must also be quoted if they are part of the node name.

The $FILESCAN service recognizes the DEC Multinational alphabetical characters (such as à) as alphanumeric characters.

The $FILESCAN service accepts extended (ODS-5) file names. The parsing types and names are independent of the volume they reside on; therefore, if the device name points to an ODS-2 volume, you can specify an ODS-5 file name. Refer to the *VSI OpenVMS Guide to OpenVMS File Applications* for information on ODS-5 file specifications.

The $FILESCAN service does not (1) assume default values for unspecified file specification components, (2) perform logical name translation on components, (3) perform wildcard processing, or (4) perform directory lookups.

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

SS$_NORMAL

The service completed successfully.

SS$_ACCVIO

The service could not read the string pointed to by the *srcstr* argument; or it could not write to an item descriptor in the item list specified by the *valuelst* argument; or it could not write to the specified auxiliary output buffer; or the *retlen* argument could not be written

SS$_BADPARAM

The item list contains an invalid item code.

Example

```
! The following is a C example for using the $FILESCAN service.

#include <perror.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <descrip.h>
#include <iledef.h>
#include <fscndef.h>
#include <starlet.h>
#include <ssdef.h>

//
// Build table of names.  Verify positions in table.
//

#if FSCN$_FILESPEC != 1
#error "table assumption for FILESPEC is invalid."
#endif
#if FSCN$_NODE_SECONDARY != 11
#error "table assumption for NODE_SECONDARY is invalid."
#endif

char *name_field[] = {
                "",
                "FILESPEC",
                "NODE",
                "DEVICE",
                "ROOT",
                "DIRECTORY",
                "NAME",
                "TYPE",
                "VERSION",
                "NODE_PRIMARY",
                "NODE_ACS",
                "NODE_SECONDARY" } ;

typedef struct fldflags FLDFLAGS ;

main(int argc, char *argv[] )
{

    int status ;
    int field_flags ;
    char *filename;
    struct dsc$descriptor_s file_name = { 0 } ;
    struct dsc$descriptor_s output_string = { 0 } ;
    short retlen ;
    int i ;
    ile2 itmlst[] = {
                {0, FSCN$_FILESPEC, 0},
```

```
                {0, FSCN$_NODE, 0},
                {0, FSCN$_DEVICE, 0},
                {0, FSCN$_ROOT, 0},
                {0, FSCN$_DIRECTORY, 0},
                {0, FSCN$_NAME, 0},
                {0, FSCN$_TYPE, 0},
                {0, FSCN$_VERSION, 0},
                {0, FSCN$_NODE_PRIMARY, 0},
                {0, FSCN$_NODE_ACS, 0},
                {0, FSCN$_NODE_SECONDARY, 0},
                { 0, 0, 0 } } ;


    output_string.dsc$w_length = 4096 ;
    if ((output_string.dsc$a_pointer = malloc(4096)) == NULL) {
        perror("Malloc Failure");
        exit(0) ;
        }
    if ((file_name.dsc$a_pointer = filename = malloc(4096)) == NULL) {
        perror("Malloc Failure");
        exit(0) ;
        }

    for( ;; ) {

        printf("\nEnter file name: ") ;
        if (gets(filename) == NULL)
            break ;

        file_name.dsc$w_length = strlen(filename) ;

        status = sys$filescan( &file_name, itmlst, &field_flags,
                               &output_string, &retlen) ;

        if (!(status&1)) exit(status) ;

        if ((itmlst[0].ile2$ps_bufaddr != NULL) && (retlen != 0))
            printf("The file name returned is %.*s\n", retlen,
                (char*)itmlst[0].ile2$ps_bufaddr ) ;
        else
            printf("There is no file name returned for FSCN$_FILESPEC\n");

        for( i = 0; i < FSCN$_NODE_SECONDARY; i++) {
        if ((int)field_flags & 1<<i)
            printf("The component %s is present: %.*s\n", name_field[i+2],
                            itmlst[i+1].ile2$w_length,
                                (char*)itmlst[i+1].ile2$ps_bufaddr ) ;
        }
    }
    exit(status) ;
}
```

This C example demonstrates the use of the $FILESCAN service.

Following is an example of the program output:

```
$ RUN TEST_SCAN

Enter file name: abc"def"
The file name is abc"def"
The component NAME is present: abc

Enter file name: "abc""def"""
There is no file name returned for FSCN$_FILESPEC

Enter file name: "abc""def""::
```

```
The file name is abc"def"::
The component NODE is present: abc"def"::
The component NODE_PRIMARY is present: abc"def"

Enter file name: "abc""def""""user password"::
The file name is abc"def""user password"::
The component NODE is present: abc"def""user password"::
The component NODE_PRIMARY is present: abc"def"
The component NODE_ACS is present: "user password"

Enter file name: abc"def"::
The file name is abc"def"::
The component NODE is present: abc"def"::
The component NODE_PRIMARY is present: abc
The component NODE_ACS is present: "def"

Enter file name: "abc""def""""system password"::[dir.1]a^ ^".file;1
The file name returned is abc"def""system password"::[dir.1]a^ ^".file;1
The component NODE is present: abc"def""system password"::
The component DIRECTORY is present: [dir.1]
The component NAME is present: a^ ^"
The component TYPE is present: .file
The component VERSION is present: ;1
The component NODE_PRIMARY is present: abc"def"
The component NODE_ACS is present: "system password"
```

$FIND

Locates Record — The Find service locates a specified record in a file and returns its record file address in the RAB$W_RFA field of the RAB. The Find service can be used with all file organizations. For additional information about this service, see the *VSI OpenVMS Record Management Services Reference Manual*.

$FIND_HELD

Find Identifiers Held by User — Returns the identifiers held by a specified holder.

Format

```
SYS$FIND_HELD holder ,[id] ,[attrib] ,[contxt]
```

C Prototype

```
int sys$find_held
   (struct _generic_64 *holder,  unsigned int *id, unsigned int *attrib,
    unsigned int *contxt);
```

Arguments

holder

OpenVMS usage:	rights_holder
type:	quadword (unsigned)
access:	read only

mechanism:	by reference

Holder whose identifiers are to be found when $FIND_HELD completes execution. The *holder* argument is the address of a quadword data structure containing the holder identifier. This quadword data structure consists of a longword containing the holder UIC, followed by a longword containing the value 0.

id

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

Identifier value found when $FIND_HELD completes execution. The *id* argument is the address of a longword containing the identifier value with which the holder is associated.

attrib

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

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

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

Bit Position	Meaning When Set
KGB$V_DYNAMIC	Allows holders of the identifier to remove it from or add it to the process rights list by using the DCL command SET RIGHTS_LIST.
KGB$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.

contxt

OpenVMS usage:	context
type:	longword (unsigned)
access:	modify

mechanism: by reference

Context value used when repeatedly calling $FIND_HELD. The *contxt* argument is the address of a longword used while searching for all identifiers. The context value must be initialized to 0, and the resulting context of each call to $FIND_HELD must be presented to each subsequent call. After *contxt* is passed to $FIND_HELD, you must not modify its value.

Description

The Find Identifiers Held by User service returns a list of the identifiers that another identifier holds. Use the $FIND_HELD service to construct the process rights when a user logs in (unless that process has read access to the rights database). To determine all the identifiers held by the specified holder, call $FIND_HELD repeatedly until it returns the status code SS$_NOSUCHID. When SS$_NOSUCHID is returned, $FIND_HELD has returned all the identifiers, cleared the context value, and deallocated the record stream.

If you complete your calls to $FIND_HELD before SS$_NOSUCHID is returned, use $FINISH_RDB to clear the context value and deallocate the record stream.

Note that, when you use wildcards with this service, the records are returned in the order that they were originally written because the first record is located on the basis of the holder ID. Thus, all the target records have the same holder ID or, in other words, they have duplicate keys, which leads to retrieval in the order in which they were written.

Required Access or Privileges

Read access to the rights database is required to obtain information about identifiers held by other users.

Required Quota

None

Related Services

$ADD_HOLDER, $ADD_IDENT, $ASCTOID, $CREATE_RDB, $FIND_HOLDER, $FINISH_RDB, $GRANTID, $IDTOASC, $MOD_HOLDER, $MOD_IDENT, $REM_HOLDER, $REM_IDENT, $REVOKID

Condition Values Returned

SS$_NORMAL

The service completed successfully.

SS$_ACCVIO

The *id* argument cannot be written by the service, or the *holder*, *attrib*, or *contxt* argument cannot be read by the service.

SS$_IVCHAN

The contents of the *contxt* longword are not valid.

SS$_INSFMEM

The process dynamic memory is insufficient for opening the rights database.

SS$_IVIDENT

The format of the specified holder identifier is invalid.

SS$_NOIOCHAN

No more rights database context streams are available.

SS$_NOSUCHID

The specified holder identifier does not exist, or no further identifiers are held by the specified holder.

RMS$_PRV

You do not have read access to the rights database.

Because the rights database is an indexed file accessed with 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 *VSI OpenVMS Record Management Services Reference Manual*.

$FIND_HOLDER

Find Holder of Identifier — Returns the holder of a specified identifier.

Format

```
SYS$FIND_HOLDER id ,[holder] ,[attrib] ,[contxt]
```

C Prototype

```
int sys$find_holder
   (unsigned int id, struct _generic_64 *holder, unsigned int *attrib,
    unsigned int *contxt);
```

Arguments

id

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

Binary identifier value whose holders are found by $FIND_HOLDER. The *id* argument is a longword containing the binary identifier value.

holder

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

Holder identifier returned when $FIND_HOLDER completes execution. The *holder* argument is the address of a quadword containing the holder identifier. The first longword contains the UIC of the holder with the high-order word containing the group number and the low-order word containing the member number. The second longword contains the value 0.

attrib

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

Mask of attributes associated with the holder record specified by *holder*. The *attrib* argument is the address of a longword containing the attribute mask.

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

Bit Position	Meaning When Set
KGB$V_DYNAMIC	Allows holders of the identifier to remove it from or add it to the process rights list by using the DCL command SET RIGHTS_LIST. For more information on SET RIGHTS_LIST, see the *VSI OpenVMS DCL Dictionary*.
KGB$V_NOACCESS	Makes any rights of the identifier null and void. This attribute is intended as a modifier for a resource identifier or the Subsystem attribute.
KGB$V_RESOURCE	Allows the holder of an identifier to charge disk space to the identifier. It is used only for file objects.
KGB$V_SUBSYSTEM	Allows holders of an 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 while searching for all the holders of the specified identifier when executing $FIND_HOLDER. The *contxt* argument is the address of a longword containing the context value. When calling $FIND_HOLDER repeatedly, *contxt* must be set initially to 0 and the resulting context

of each call to $FIND_HOLDER must be presented to each subsequent call. After the argument is passed to $FIND_HOLDER, you must not modify its value.

Description

The Find Holder of Identifier service returns the holder of the specified identifier. To determine all the holders of the specified identifier, you call SYS$FIND_HOLDER repeatedly until it returns the status code SS$_NOSUCHID, which indicates that $FIND_HOLDER has returned all identifiers, cleared the context longword, and deallocated the record stream. If you complete your calls to $FIND_HOLDER before SS$_NOSUCHID is returned, you use the $FINISH_RDB service to clear the context value and deallocate the record stream.

Note that when you use wildcards with this service, the records are returned in the order in which they were originally written. (This action results from the fact that the first record is located on the basis of the identifier. Thus, all the target records have the same identifier or, in other words, they have duplicate keys, which leads to retrieval in the order in which they were written.)

Required Access or Privileges

Read access to the rights database is required to obtain information about identifiers marked HOLDER_HIDDEN.

Required Quota

None

Related Services

$ADD_HOLDER, $ADD_IDENT, $ASCTOID, $CREATE_RDB, $FIND_HELD, $FINISH_RDB, $GRANTID, $IDTOASC, $MOD_HOLDER, $MOD_IDENT, $REM_HOLDER, $REM_IDENT, $REVOKID

Condition Values Returned

SS$_NORMAL

The service completed successfully.

SS$_ACCVIO

The *id* argument cannot be read by the caller, or the *holder*, *attrib*, or *contxt* argument cannot be written by the caller.

SS$_IVCHAN

The contents of the *contxt* longword are not valid.

SS$_INSFMEM

The process dynamic memory is insufficient for opening the rights database.

SS$_IVIDENT

The specified identifier or holder identifier is of invalid format.

SS$_NOIOCHAN

No more rights database context streams are available.

SS$_NOSUCHID

The specified identifier does not exist in the rights database, or no further holders exist for the specified identifier.

RMS$_PRV

The user does not have read access to the rights database.

Because the rights database is an indexed file accessed with 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 *VSI OpenVMS Record Management Services Reference Manual*.

$FINISH_RDB

Terminate Rights Database Context — Deallocates the record stream and clears the context value used with $FIND_HELD, $FIND_HOLDER, or $IDTOASC. On Alpha and Integrity server systems, this service accepts 64-bit addresses.

Format

```
SYS$FINISH_RDB contxt
```

C Prototype

```
int sys$finish_rdb (unsigned int *contxt);
```

Argument

contxt

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

Context value to be cleared when $FINISH_RDB completes execution. The *contxt* argument is a longword containing the address of the context value.

Description

The Terminate Rights Database Context service clears the context longword and deallocates the record stream associated with a sequence of rights database lookups performed by the $IDTOASC, $FIND_HOLDER, and $FIND_HELD services.

If you repeatedly call $IDTOASC, $FIND_HOLDER, or $FIND_HELD until SS$_NOSUCHID is returned, you do not need to call $FINISH_RDB because the record stream has already been deallocated and the context longword has already been cleared.

Required Access or Privileges

None

Required Quota

None

Related Services

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

Condition Values Returned

SS$_NORMAL

The service completed successfully.

SS$_ACCVIO

The `contxt` argument cannot be written by the caller.

SS$_IVCHAN

The contents of the `contxt` longword are not valid.

Because the rights database is an indexed file accessed with 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 *VSI OpenVMS Record Management Services Reference Manual*.

$FLUSH

Writes Out Buffers — The Flush service writes out all modified I/O buffers and file attributes associated with the file. This ensures that all record activity up to the point at which the Flush service executes is actually reflected in the file. For additional information about this service, see the *VSI OpenVMS Record Management Services Reference Manual*.

$FORCEX

Force Exit — Causes an Exit ($EXIT) service call to be issued on behalf of a specified process.

Format

```
SYS$FORCEX [pidadr] ,[prcnam] ,[code]
```

C Prototype

```
int sys$forcex (unsigned int *pidadr, void *prcnam, unsigned int code);
```

Arguments

pidadr

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

Process identification (PID) of the process to be forced to exit. The *pidadr* argument is the address of a longword containing the PID. The *pidadr* argument can refer to a process running on the local node or a process running on another node in the OpenVMS Cluster system.

The *pidadr* argument is optional but must be specified if the process that is to be forced to exit is not in the same UIC group as the calling process.

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 that is to be forced to exit. The *prcnam* argument is the address of a character string descriptor pointing to the process name string. 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 *prcnam* argument can be used only on behalf of processes in the same UIC group as the calling process. To force processes in other groups to exit, you must specify the *pidadr* argument. This restriction exists because the operating system interprets the UIC group number of the calling process as part of the specified process name; the names of processes are unique to UIC groups.

code

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

Completion code value to be used as the exit parameter. The *code* argument is a longword containing this value. If you do not specify the *code* argument, the value 0 is passed as the completion code.

Description

The Force Exit service causes an Exit service call to be issued on behalf of a specified process.

If you specify neither the *pidadr* nor the *prcnam* argument, the caller is forced to exit and control is not returned.

If the longword at address *pidadr* is 0, the PID of the target process is returned.

The Force Exit system service requires system dynamic memory.

The image executing in the target process follows normal exit procedures. For example, if any exit handlers have been specified, they gain control before the actual exit occurs. Use the Delete Process ($DELPRC) service if you do not want a normal exit.

When a forced exit is requested for a process, a user-mode asynchronous system trap (AST) is queued for the target process. The AST routine causes the $EXIT service call to be issued by the target process. Because the AST mechanism is used, user mode ASTs must be enabled for the target process, or no exit occurs until ASTs are reenabled. Thus, for example, a suspended process cannot be stopped by $FORCEX. The process that calls $FORCEX receives no notification that the exit is not being performed.

If an exit handler resumes normal processing, the process will not exit. In particular, if the program is written in Ada and there is a task within the program that will not terminate, the program will not exit.

The $FORCEX service completes successfully if a force exit request is already in effect for the target process but the exit is not yet completed.

Required Access or Privileges

Depending on the operation, the calling process might need a certain privilege to use $FORCEX:

- You need GROUP privilege to force an exit for a process in the same group that does not have the same UIC as the calling process.
- You need WORLD privilege to force an exit for any process in the system.

Required Quota

None

Related Services

$CANEXH, $CREPRC, $DCLEXH, $DELPRC, $EXIT, $GETJPI, $GETJPIW, $HIBER, $PROCESS_SCAN, $RESUME, $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$_INSFMEM

The system dynamic memory is insufficient for the operation.

SS$_IVLOGNAM

The process name string has a length equal to 0 or greater than 15.

SS$_NONEXPR

The specified process does not exist, or an invalid process identification was specified.

SS$_NOPRIV

The process does not have the privilege to force an exit for the specified process.

SS$_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.)

$FORGET_RM

Forget Resource Manager — Deletes a Resource Manager instance (RMI) from the calling process.

Format

```
YS$FORGET_RM [efn] ,[flags] ,iosb ,[astadr] ,[astprm] ,rm_id
```

C Prototype

```
int sys$forget_rm
   (unsigned int efn, unsigned int flags, struct _iosb *iosb,
    void (*astadr)(__unknown_params), int astprm, unsigned int rm_id);
```

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 used.

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 the option flag listed in Table 37. All undefined bits must be 0. If this argument is omitted, no flags are used.

Table 37. $FORGET_RM Option Flag

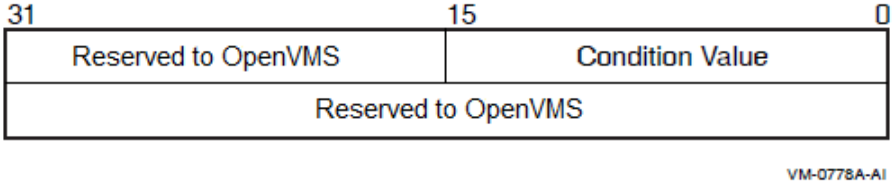
Flag Name	Description
DDTM$M_SYNC	Specifies successful synchronous completion by returning SS$_SYNCH. When SS$_SYNCH is returned, the AST routine is not called, the event flag is not set, and the I/O status block is not filled in.

iosb

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

The 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:



astadr

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

The AST routine executed when the service completes, if SS$_NORMAL is returned in R0. The *astadr* argument is the address of the entry mask of this routine. The routine is executed in the same access mode as that of the caller of the $FORGET_RM service.

astprm

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

The AST parameter that is passed to the AST routine specified by the `astadr` argument.

rm_id

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

The identifier of the RMI to be deleted from the calling process.

Description

The $FORGET_RM system service:

- Deletes the specified Resource Manager instance (RMI) from the calling process.
- Tries to abort all transactions that have not already committed and that have Resource Manager (RM) participants associated with that RMI.
- Removes all RM participants associated with the RMI from their transactions.
- Implicitly acknowledges all unacknowledged event reports delivered to that RMI or to its RM participants. The reply given in the implicit acknowledgment depends on the type of the event as follows:

Table 38. $FORGET_RM's Implicit Acknowledgments

Type of Event	Reply
Abort	SS$_NORMAL
Commit	SS$_REMEMBER
Prepare	SS$_VETO (with the DDTM$_SEG_FAIL reason code)
One-phase commit	SS$_VETO
Default transaction started	SS$_NORMAL
Nondefault transaction started	SS$_NORMAL

Preconditions for the successful completion of $FORGET_RM are:

- The calling process must contain the specified RMI.

- The access mode of the caller must be the same as or more privileged than that of the specified RMI.

Postconditions on successful completion of $FORGET_RM are described in Table 39:

Table 39. Postconditions When $FORGET_RM Completes Successfully

Postcondition	Meaning
The specified RMI is deleted from the calling process.	The result is that: • Its identifier is invalid. Any subsequent calls to $JOIN_RM or $FORGET_RM that pass its identifier will fail. • The DECdtm transaction manager will deliver no more event reports to that RMI.
There are no RM participants associated with the RMI.	Removes all RM participants associated with the specified RMI from their transactions. Thus the DECdtm transaction manager will deliver no more event reports to those RM participants. For an RM participant that had an unacknowledged event report, the postconditions are the same as those of the appropriate implicit acknowledgment (see Table 38) except that the RM participant is always removed from the transaction.
There are no unacknowledged event reports delivered to the RMI or its RM participants.	All unacknowledged event reports are implicitly acknowledged by this call to $FORGET_RM (see Table 38). Thus a subsequent call to $ACK_EVENT that acknowledges one of these event reports will fail.
Quotas are returned.	Returns the following quotas: • The BYTLM quota consumed by the call to $DECLARE_RM that created the RMI. • The ASTLM quotas consumed by all calls to $JOIN_RM or $ACK_EVENT that added RM participants associated with the RMI.

Note that when a process terminates (normally or abnormally), a $FORGET_RM is automatically performed for each RMI in that process. And when an image terminates (normally or abnormally), a $FORGET_RM is automatically performed for each user mode RMI in that process.

There is also a wait form of the service, $FORGET_RMW.

Required Privileges

None

Required Quotas

None

Related Services

$ABORT_TRANS, $ABORT_TRANSW, $ACK_EVENT, $ADD_BRANCH, $ADD_BRANCHW, $CREATE_UID, $DECLARE_RM, $DECLARE_RMW, $END_BRANCH, $END_BRANCHW, $END_TRANS, $END_TRANSW, $FORGET_RMW, $GETDTI, $GETDTIW, $GET_DEFAULT_TRANS, $JOIN_RM, $JOIN_RMW, $SETDTI,

$SETDTIW, $SET_DEFAULT_TRANS, $SET_DEFAULT_TRANSW, $START_BRANCH, $START_BRANCHW, $START_TRANS, $START_TRANSW, $TRANS_EVENT, $TRANS_EVENTW

Condition Values Returned

SS$_NORMAL

If returned in R0, the request was successfully queued. If 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 to the caller.

SS$_BADPARAM

An option flag was invalid.

SS$_EXASTLM

The process AST limit (ASTLM) was exceeded.

SS$_ILLEFC

The event flag number was invalid.

SS$_INSFARGS

A required argument was missing.

SS$_INSFMEM

There was insufficient system dynamic memory for the operation.

SS$_WRONGACMODE

The access mode of the caller was less privileged than that of the RMI.

SS$_NOSUCHRM

The calling process did not contain the specified RMI.

$FORGET_RMW

Forget Resource Manager and Wait — Deletes a Resource Manager instance (RMI) from the calling process. $FORGET_RMW always waits for the request to complete before returning to the caller. Other than this, it is identical to $FORGET_RM.

Format

```
SYS$FORGET_RMW [efn] ,[flags] ,iosb ,[astadr] ,[astprm] ,rm_id
```

C Prototype

```
int sys$forget_rmw
   (unsigned int efn, unsigned int flags, struct _iosb *iosb,
    void (*astadr)(__unknown_params), int astprm, unsigned int rm_id);
```

$FORMAT_ACL

Format Access Control List Entry — Formats the specified access control entry (ACE) into a text string.

Format

```
SYS$FORMAT_ACL
   aclent ,[acllen] ,aclstr ,[width] ,[trmdsc] ,[indent] ,[accnam]
   ,[nullarg]
```

C Prototype

```
int sys$format_acl
   (void *aclent, unsigned short int *acllen, void *aclstr,
    unsigned short int *width, void *trmdsc, unsigned short int *indent,
    unsigned int *accnam, int (*routin)(__unknown_params));
```

Arguments

aclent

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

Description of the ACE formatted when $FORMAT_ACL completes execution. The *aclent* argument is the address of a descriptor pointing to a buffer containing the description of the input ACE. The first byte of the buffer contains the length of the ACE; the second byte contains a value that identifies the type of ACE, which in turn determines the ACE format.

For more information about the ACE format, see the Description section.

acllen

OpenVMS usage:	word_unsigned
type:	word (unsigned)
access:	write only

mechanism:	by reference

Length of the output string resulting when $FORMAT_ACL completes execution. The *acllen* argument is the address of a word containing the number of characters written to *aclstr*.

aclstr

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

Formatted ACE resulting when $FORMAT_ACL completes its execution. The *aclstr* argument is the address of a string descriptor pointing to a buffer containing the output string.

width

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

Maximum width of the formatted ACE resulting when $FORMAT_ACL completes its execution. The *width* argument is the address of a word containing the maximum width of the formatted ACE. If this argument is omitted or contains the value 0, an infinite length display line is assumed. When the width is exceeded, the character specified by *trmdsc* is inserted.

trmdsc

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

Line termination characters used in the formatted ACE. The *trmdsc* argument is the address of a descriptor pointing to a character string containing the termination characters that are inserted for each formatted ACE when the width has been exceeded.

indent

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

Number of blank characters beginning each line of the formatted ACE. The *indent* argument is the address of a word containing the number of blank characters that you want inserted at the beginning of each formatted ACE.

accnam

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

Names of the bits in the access mask when executing the $FORMAT_ACL. The *accnam* argument is the address of an array of 32 quadword descriptors that define the names of the bits in the access mask. Each element points to the name of a bit. The first element names bit 0, the second element names bit 1, and so on.

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 OpenVMS.

Description

The Format Access Control List Entry service formats the specified access control entry (ACE) into text string representation. There are seven types of ACE:

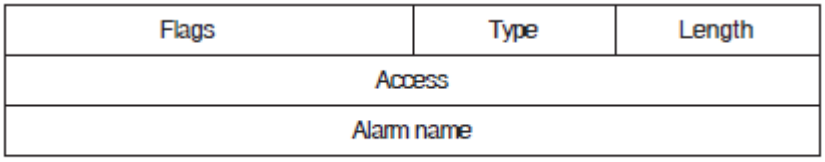
- Alarm ACE
- Application ACE
- Audit ACE

- Creator ACE
- Default Protection ACE
- Identifier ACE
- Subsystem ACE

The format for each of the ACE types is described in the following sections and the byte offsets and type values for each ACE type are defined in the $ACEDEF system macro library.

Alarm ACE

The access Alarm ACE generates a security alarm. Its format is as follows.



ZK–1710–GE

The following table describes the ACE fields and lists the symbol name for each.

Field	Symbol Name	Description
Length	ACE$B_SIZE	Byte containing the length in bytes of the ACE buffer
Type	ACE$B_TYPE	Byte containing the type value ACE$C_ALARM
Flags	ACE$W_FLAGS	Word containing Alarm ACE information and ACE type-independent information
Access	ACE$L_ACCESS	Longword containing a mask indicating the access modes to be watched
Alarm name	ACE$T_AUDITNAME	Character string containing the alarm name

The flag field contains information specific to Alarm ACEs and information applicable to all types of ACEs. The following symbols are bit offsets to the Alarm ACE information.

Bit Position	Meaning When Set
ACE$V_SUCCESS	Indicates that the alarm is raised when access is successful
ACE$V_FAILURE	Indicates that the alarm is raised when access fails

The following symbols are bit offsets to ACE information that is independent of ACE type.

Bit Position	Meaning When Set
ACE$V_DEFAULT	This ACE is added to the ACL of any file created in the directory whose ACL contains this ACE. This bit is applicable only for an ACE in a directory file's ACL.
ACE$V_HIDDEN	This ACE is application dependent. You cannot use the DCL ACL commands and the ACL editor to change the setting; the DCL command DIRECTORY/ACL does not display it.
ACE$V_NOPROPAGATE	This ACE is not propagated among versions of the same file.

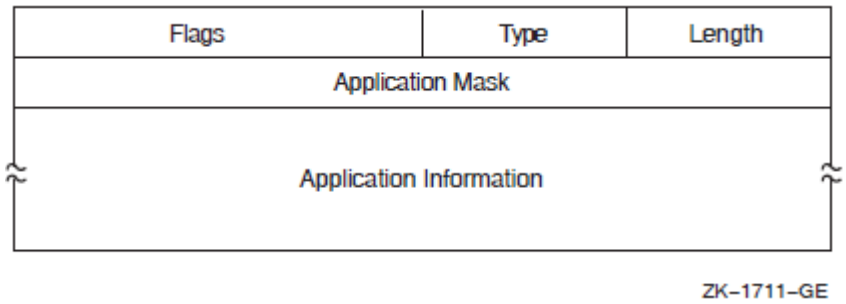
Bit Position	Meaning When Set
ACE$V_PROTECTED	This ACE is not deleted if the entire ACL is deleted; instead you must delete this ACE explicitly.

The following symbol values are offsets to bits within the access mask. You can also obtain the symbol values as masks with the appropriate bit set using the prefix ACE$M rather than ACE$V.

Bit	Meaning When Set
ACE$V_READ	Read access is monitored.
ACE$V_WRITE	Write access is monitored.
ACE$V_EXECUTE	Execute access is monitored.
ACE$V_DELETE	Delete access is monitored.
ACE$V_CONTROL	Modification of the access field is monitored.

Application ACE

The Application ACE contains application-dependent information. Its format is as follows.



The following table describes the ACE fields and lists the symbol name for each.

Field	Symbol Name	Description
Length	ACE$B_SIZE	Byte containing the length in bytes of the ACE buffer.
Type	ACE$B_TYPE	Byte containing the type value ACE$C_INFO.
Flags	ACE$W_FLAGS	Word containing Application ACE information and ACE type-independent information.
Application mask	ACE$L_INFO_FLAGS	Longword containing a mask defined and used by the application.
Application information	ACE$T_INFO_START	Variable-length data structure defined and used by the application. The length of this data is implied by the length field.

The flag field contains information specific to Application ACEs and information applicable to all types of ACEs. The following symbol is a bit offset to the Application ACE information.

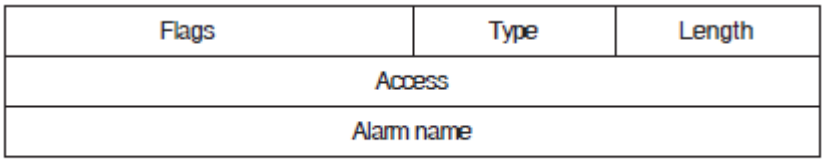
Bit	Meaning When Set
ACE$V_INFO_TYPE	Four-bit field containing a value indicating whether the application is a CSS application (ACE$C_CSS) or a customer application (ACE$C_CUST).

The following symbols are bit offsets to ACE information that is independent of ACE type.

Bit	Meaning When Set
ACE$V_DEFAULT	This ACE is added to the ACL of any file created in the directory whose ACL contains this ACE. This bit is applicable only for an ACE in a directory file's ACL.
ACE$V_HIDDEN	This bit is application dependent. You cannot use the DCL ACL commands and the ACL editor to change the setting; the DCL command DIRECTORY/ACL does not display it.
ACE$V_NOPROPAGATE	This ACE is not propagated among versions of the same file.
ACE$V_PROTECTED	This ACE is not deleted if the entire ACL is deleted; instead you must delete this ACE explicitly.

Audit ACE

The Audit ACE sets a security audit. Its format is as follows.



ZK–1710–GE

The following table describes the ACE fields and lists the symbol name for each.

Field	Symbol Name	Description
Length	ACE$B_SIZE	Byte containing the length in bytes of the ACE buffer
Type	ACE$B_TYPE	Byte containing the type value ACE$C_AUDIT
Flags	ACE$W_FLAGS	Word containing Audit ACE information and ACE type-independent information
Access	ACE$L_ACCESS	Longword containing a mask indicating the access modes to be watched
Alarm name	ACE$T_AUDITNAME	Character string containing the alarm name

The following symbols are bit offsets to ACE information that is independent of ACE type.

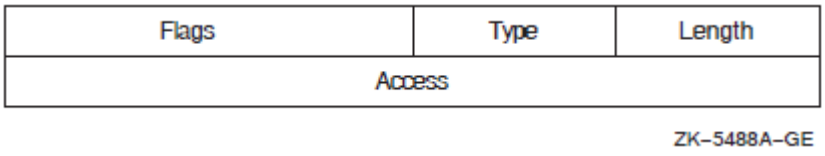
Bit Position	Meaning When Set
ACE$V_DEFAULT	This ACE is added to the ACL of any file created in the directory whose ACL contains this ACE. This bit is applicable only for an ACE in a directory file's ACL.
ACE$V_HIDDEN	This ACE is application dependent. You cannot use the DCL ACL commands and the ACL editor to change the setting; the DCL command DIRECTORY/ACL does not display it.
ACE$V_NOPROPAGATE	This ACE is not propagated among versions of the same file.
ACE$V_PROTECTED	This ACE is not deleted if the entire ACL is deleted; instead you must delete this ACE explicitly.

The following symbol values are offsets to bits within the access mask. You can also obtain the symbol values as masks with the appropriate bit set using the prefix ACE$M rather than ACE$V.

Bit	Meaning When Set
ACE$V_READ	Read access is monitored.
ACE$V_WRITE	Write access is monitored.
ACE$V_EXECUTE	Execute access is monitored.
ACE$V_DELETE	Delete access is monitored.
ACE$V_CONTROL	Modification of the access field is monitored.

Creator ACE

The Creator ACE controls access to an object based on creators. Its format is as follows.



The following table describes the ACE fields and lists the symbol name for each.

Field	Symbol Name	Description
Length	ACE$B_SIZE	Byte containing the length in bytes of the ACE buffer.
Type	ACE$B_TYPE	Byte containing the type value ACE$C_NEW_OWNER.
Flags	ACE$W_FLAGS	Word containing Creator ACE information and ACE type-independent information.
Access	ACE$L_ACCESS	Longword containing a mask indicating the access modes to be granted to the creator of the file.

The following symbols are bit offsets to ACE information that is independent of ACE type.

Bit	Meaning When Set
ACE$V_NOPROPAGATE	This ACE is not propagated among versions of the same file.
ACE$V_PROTECTED	This ACE is not deleted if the entire ACL is deleted; instead you must delete this ACE explicitly.

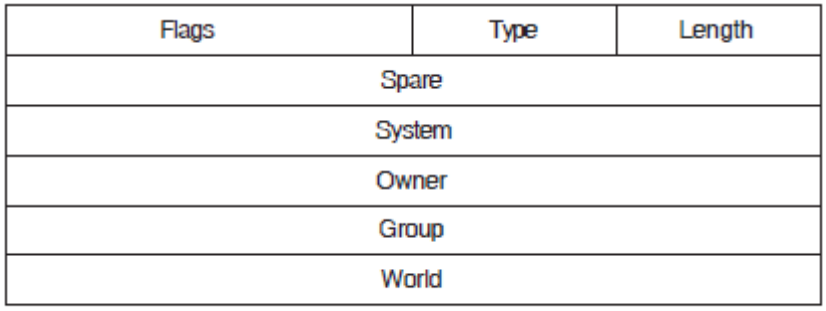
The following symbol values are offsets to bits within the mask indicating the access mode granted in the system, owner, group, and world fields.

Bit Position	Meaning When Set
ACE$V_READ	Read access is granted.
ACE$V_WRITE	Write access is granted.
ACE$V_EXECUTE	Execute access is granted.
ACE$V_DELETE	Delete access is granted.
ACE$V_CONTROL	Modification of the access field is granted.

You can also obtain the symbol values as masks with the appropriate bit set by using the prefix ACE$M rather than ACE$V.

Default Protection ACE

The Default Protection ACE specifies the UIC-based protection for all files created in the directory. You can use this type of ACE only in the ACL of a directory file. Its format is as follows.



The following table describes the ACE fields and lists the symbol name for each.

Field	Symbol Name	Description
Length	ACE$B_SIZE	Byte containing the length in bytes of the ACE buffer.
Type	ACE$B_TYPE	Byte containing the type value ACE$C_DIRDEF.
Flags	ACE$W_FLAGS	Word containing ACE type-independent information.
Spare	ACE$L_SPARE1	Longword that is reserved for future use and must be 0.
System	ACE$L_SYS_PROT	Longword containing a mask indicating the access mode granted to system users. Each bit represents one type of access.
Owner	ACE$L_OWN_PROT	Longword containing a mask indicating the access mode granted to the owner. Each bit represents one type of access.
Group	ACE$L_GRP_PROT	Longword containing a mask indicating the access mode granted to group users. Each bit represents one type of access.
World	ACE$L_WOR_PROT	Longword containing a mask indicating the access mode granted to the world. Each bit represents one type of access.

The flag field contains information applicable to all types of ACEs. The following symbols are bit offsets to ACE information that is independent of ACE type.

Bit Position	Meaning When Set
ACE$V_HIDDEN	This ACE is application dependent. You cannot use the DCL ACL commands and the ACL editor to change the setting; the DCL command DIRECTORY/ACL does not display it.
ACE$V_NOPROPAGATE	This ACE is not propagated among versions of the same file.
ACE$V_PROTECTED	This ACE is not deleted if the entire ACL is deleted; instead you must delete this ACE explicitly.

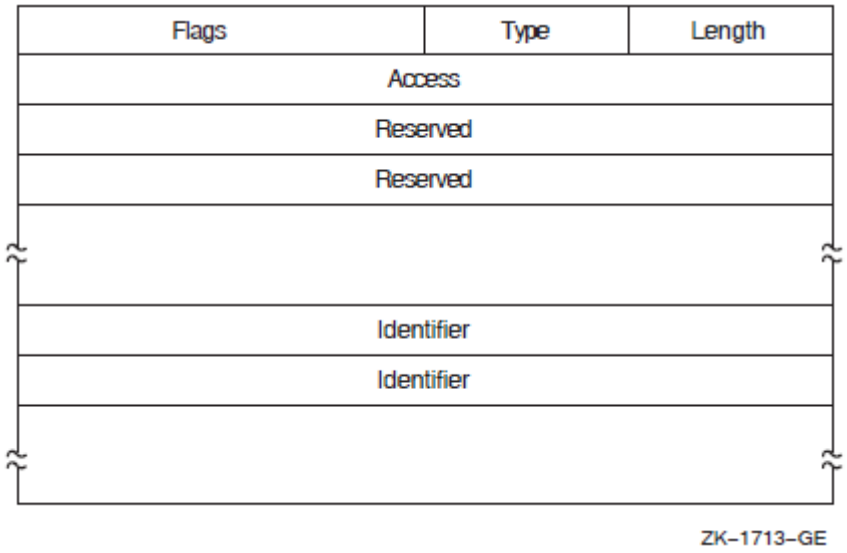
The system interprets the bits within the access mask as shown in the following table. The following symbol values are offsets to bits within the mask indicating the access mode granted in the system, owner, group, and world fields.

Bit Position	Meaning When Set
ACE$V_READ	Read access is granted.
ACE$V_WRITE	Write access is granted.
ACE$V_EXECUTE	Execute access is granted.
ACE$V_DELETE	Delete access is granted.

You can also obtain the symbol values as masks with the appropriate bit set by using the prefix ACE$M rather than ACE$V.

Identifier ACE

The Identifier ACE controls access to an object based on identifiers. Its format is as follows.



The following table describes the ACE fields and lists the symbol name for each.

Field	Symbol Name	Description
Length	ACE$B_SIZE	Byte containing the length in bytes of the ACE buffer.
Type	ACE$B_TYPE	Byte containing the type value ACE$C_KEYID.
Flags	ACE$W_FLAGS	Word containing Identifier ACE information and ACE type-independent information.
Access	ACE$L_ACCESS	Longword containing a mask indicating the access mode granted to the specified identifiers.
Reserved	ACE$V_RESERVED	Longwords containing application-specific information. The number of reserved longwords is specified in the flags field.
Identifier	ACE$L_KEY	Longwords containing identifiers. The number of longwords is implied by ACE$B_SIZE. If an accessor holds all of the listed identifiers, the ACE is said to match the accessor, and the access specified in ACE$L_ACCESS is granted.

The flags field contains information specific to Identifier ACEs and information applicable to all types of ACEs. The following symbol is a bit offset to Identifier ACE information.

Bit	Meaning When Set
ACE$V_RESERVED	Four-bit field containing the number of longwords to reserve for application-dependent data. The number must be between 0 and 15. The reserved longwords, if any, immediately precede the identifiers.

The following symbols are bit offsets to ACE information that is independent of ACE type.

Bit	Meaning When Set
ACE$V_DEFAULT	This ACE is added to the ACL of any file created in the directory whose ACL contains this ACE. This bit is applicable only for an ACE in a directory file's ACL.
ACE$V_HIDDEN	This bit is application dependent. You cannot use the DCL ACL commands and the ACL editor to change the setting; the DCL command DIRECTORY/ACL does not display it.
ACE$V_NOPROPAGATE	This ACE is not propagated among versions of the same file.
ACE$V_PROTECTED	This ACE is not deleted if the entire ACL is deleted; instead you must delete this ACE explicitly.

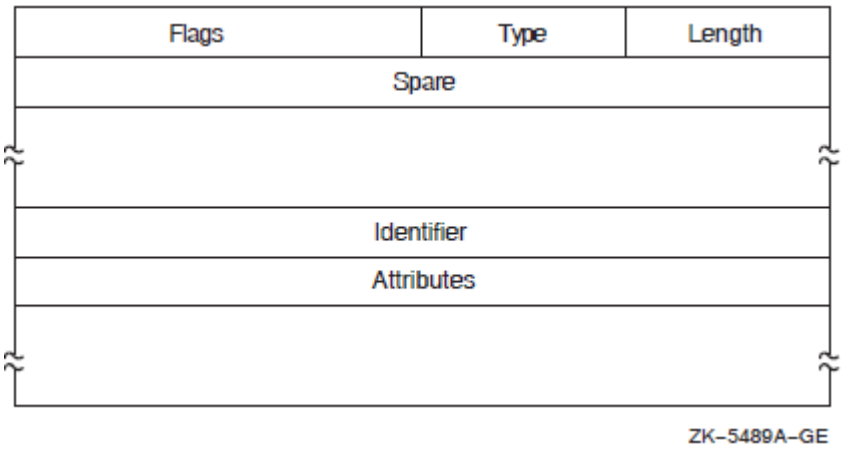
The following symbol values are offsets to bits within the mask indicating the access mode granted in the system, owner, group, and world fields.

Bit Position	Meaning When Set
ACE$V_READ	Read access is granted.
ACE$V_WRITE	Write access is granted.
ACE$V_EXECUTE	Execute access is granted.
ACE$V_DELETE	Delete access is granted.
ACE$V_CONTROL	Modification of the access field is granted.

You can also obtain the symbol values as masks with the appropriate bit set by using the prefix ACE$M rather than ACE$V.

Subsystem ACE

The Subsystem ACE maintains protected subsystems. Its format is as follows.



The following table describes the ACE fields and lists the symbol name for each.

Field	Symbol Name	Description
Length	ACE$B_SIZE	Byte containing the length in bytes of the ACE buffer.
Type	ACE$B_TYPE	Byte containing the type value ACE$C_SUBSYSTEM_IDS.
Flags	ACE$W_FLAGS	Word containing Subsystem ACE information and ACE type-independent information.
Spare	ACE$L_SPARE1	Longword that is reserved for future use and must be 0.
Identifier/Attributes	ACE$Q_IMAGE_IDS	Longword identifier value and its associated longword attributes.

A Subsystem ACE can contain multiple identifier/attribute pairs. In this case, the Subsystem ACE is an array of identifiers and attributes starting at ACE$Q_IMAGE_IDS. Beginning at this offset, KGB$L_IDENTIFIER and KGB$L_ATTRIBUTES are used to address each of the separate longwords.

The number of identifier/attribute pairs is computed by subtracting ACE$C_LENGTH from ACE$W_SIZE and dividing by KGB$S_IDENTIFIER.

The following symbols are bit offsets to ACE information that is independent of ACE type.

Bit	Meaning When Set
ACE$V_NOPROPAGATE	This ACE is not propagated among versions of the same file.
ACE$V_PROTECTED	This ACE is not deleted if the entire ACL is deleted; instead you must delete this ACE explicitly.

The following symbol values are offsets to bits within the mask indicating the access mode granted in the system, owner, group, and world fields.

Bit Position	Meaning When Set
ACE$V_READ	Read access is granted.
ACE$V_WRITE	Write access is granted.
ACE$V_EXECUTE	Execute access is granted.
ACE$V_DELETE	Delete access is granted.
ACE$V_CONTROL	Modification of the access field is granted.

You can also obtain the symbol values as masks with the appropriate bit set by using the prefix ACE$M rather than ACE$V.

Required Access or Privileges

None

Required Quota

None

Related Services

$ADD_HOLDER, $ADD_IDENT, $ASCTOID, $CREATE_RDB, $CREATE_USER_PROFILE, $FIND_HELD, $FIND_HOLDER, $FINISH_RDB, $FORMAT_AUDIT, $GET_SECURITY,

$GRANTID, $HASH_PASSWORD, $IDTOASC, $MOD_HOLDER, $MOD_IDENT, $REM_HOLDER, $REM_IDENT, $REVOKID, $SET_RESOURCE_DOMAIN, $SET_SECURITY

Condition Values Returned

SS$_BUFFEROVF

The service completed successfully. The output string has overflowed the buffer and has been truncated.

SS$_NORMAL

The service completed successfully.

SS$_ACCVIO

The ACL entry or its descriptor cannot be read by the caller, or the string descriptor cannot be read by the caller, or the length word or the string buffer cannot be written by the caller.

$FORMAT_AUDIT

Format Security Audit Event Message — Converts a security auditing event message from binary format to ASCII text.

Format

```
SYS$FORMAT_AUDIT
   fmttyp ,audmsg ,[outlen] ,outbuf ,[width] ,[trmdsc] ,[routin] ,[fmtflg]
```

C Prototype

```
int sys$format_audit
   (unsigned int fmttyp, void *audmsg, unsigned short int *outlen,
    void *outbuf, unsigned short int *width, void *trmdsc,
    int (*routin)(__unknown_params), unsigned int fmtflg);
```

Arguments

fmttyp

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

Format for the message. The *fmttyp* argument is a value indicating whether the security audit message should be in brief format, which is one line of information, or full format. The default is full format. See the *VSI OpenVMS System Manager's Manual* for examples of formatted output.

The following table defines the brief and full formats.

Value	Meaning
NSA$C_FORMAT_STYLE_BRIEF	Use a brief format for the message.
NSA$C_FORMAT_STYLE_FULL	Use a full format for the message.

audmsg

OpenVMS usage:	char_string
type:	byte stream (unsigned)
access:	read only
mechanism:	by reference

Security auditing message to format. The *audmsg* argument is the address of a buffer containing the message that requires formatting.

outlen

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

Length of the formatted security audit message. The *outlen* argument is the address of the word receiving the final length of the ASCII message.

outbuf

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

Buffer holding the formatted message. The *outbuf* argument is the address of a descriptor pointing to the buffer receiving the message.

width

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

Maximum width of the formatted message. The *width* argument is the address of a word containing the line width value. The default is 80 columns.

The *width* argument does not work consistently. In most cases, if you specify both the *width* argument and the full format style (NSA$C_FORMAT_STYLE_FULL), $FORMAT_AUDIT ignores the *width* argument. The minimum width is 80 columns; lower values do not limit the width to less than 80. If you specify a width greater than 80 columns, most lines are not joined to use the full width.

In most cases, you should avoid using the *width* argument.

trmdsc

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

Line termination characters used in a full format message. The *trmdsc* argument is the address of a descriptor pointing to the line termination characters to insert within a line segment whenever the *width* is reached.

routin

OpenVMS usage:	procedure
type:	procedure value
access:	read only
mechanism:	by reference

Routine that writes a formatted line to the output buffer. The *routin* argument is the address of a routine called each time a line segment is formatted. The argument passed to the routine is the address of a character string descriptor for the line segment.

When an application wants event messages in the brief format, $FORMAT_AUDIT calls the routine twice to format the first event message. The first time it is called, the routine passes a string containing the column titles for the message. The second and subsequent calls to the routine pass the formatted event message. By using this routine argument, a caller can gain control at various points in the processing of an audit event message.

fmtflg

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

Determines the formatting of certain kinds of audit messages. The *fmtflg* argument is a mask specifying whether sensitive information should be displayed or column titles built for messages in brief format. For example, the operating system uses bit 0 to suppress plain-text passwords from security alarm messages. The following table describes the significant bits.

Bit	Value	Description
0	1	Do not format sensitive information.
	0	Format sensitive information.
1	1	Build a column title for messages in brief format. (You must specify a *fmttyp* of brief and a *routin* argument.)
	0	Do not build column titles.

Description

The Format Audit service converts a security auditing event message from binary format to ASCII text and can filter sensitive information. $FORMAT_AUDIT allows the caller to format a message in a

multiple-line format or a single-line format and tailor the information for a display device of a specific width.

$FORMAT_AUDIT is intended for utilities that need to format the security auditing event messages received from the audit server listener mailbox or the system security audit log file.

Required Access or Privileges

None

Required Quota

$FORMAT_AUDIT can cause a process to exceed its page-file quota (PGFLQUOTA) if it has to format a long auditing event message. The caller of $FORMAT_AUDIT can also receive quota violations from services that $FORMAT_AUDIT uses, such as $IDTOASC, $FAO, and $GETMSG.

Related Services

$AUDIT_EVENT

Condition Values Returned

SS$_NORMAL

The service completed successfully.

SS$_MSGNOTFND

The service completed successfully; however, the message code cannot be found and a default message has been returned.

SS$_ACCVIO

The item list cannot be read by the caller, or the buffer length or buffer cannot be written by the caller.

SS$_BADPARAM

The item list contains an invalid identifier.

SS$_BUFFEROVF

The service completed successfully; however, the formatted output string overflowed the output buffer and has been truncated.

SS$_INSFMEM

The process dynamic memory is insufficient for opening the rights database.

SS$_IVCHAN

The format of the specified identifier is not valid. This condition value returned is not directly returned by $FORMAT_AUDIT. It is indirectly returned when $FORMAT_AUDIT in turn calls another service, such as an identifier translation or binary time translation service.

SS$_IVIDENT

The format of the specified identifier is invalid.

SS$_NOSUCHID

The specified identifier name does not exist in the rights database. This condition value returned is not directly returned by $FORMAT_AUDIT. It is indirectly returned when $FORMAT_AUDIT in turn calls another service, such as an identifier translation or binary time translation service.

$FREE

Unlocks Records — The Free service unlocks all records that were previously locked for the record stream. Refer to the *VSI OpenVMS Record Management Services Reference Manual* for additional information about this service.

$FREE_USER_CAPABILITY (Alpha and Integrity servers)

Release a Reserved User Capability — On Alpha and Integrity server systems, releases a user capability, indicating to other processes that the resource is now available. This service accepts 64-bit addresses.

Format

```
SYS$GET_USER_CAPABILITY cap_num [,prev_num] [,flags]
```

C Prototype

```
int sys$free_user_capability
   (int *cap_num, struct  _generic_64 *prev_mask,
    struct _generic_64 *flags);
```

Arguments

cap_num

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

Capability number to be released by the calling Kernel thread. This number can range from 1 to 16. The *cap_num* argument is the 32- or 64-bit address of the longword containing the user capability number.

prev_mask

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

The previous user capability reservation mask before execution of this service call. The *prev_mask* argument is the 32- or 64-bit address of a quadword into which the service writes a quadword bit mask specifying the previously reserved user capabilities.

flags

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

Options selected for the user capability reservation. The *flags* argument is a quadword bit vector wherein a bit corresponds to an option.

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.

At this time, all bits are reserved to OpenVMS and must be 0.

Description

The Release a Reserved User Capability service releases a user capability back to the global pool, making it available for subsequent calls to $GET_USER CAPABILITY. The state of all user capabilities in the system are kept in SCH$GQ_RESERVED_USER_CAPS; this service clears the bit position in that cell reflecting the capability number specified in *cap_num*.

This service can also return the state of the global reservation bit mask prior to a release operation.

Required Privileges

The caller must have both ALTPRI and WORLD privileges to call $FREE_USER_CAPABILITY to release a user capability. No privileges are required if $FREE_USER_CAPABILITY is called only to retrieve the current user capability reservation mask.

Required Quota

None.

Related Services

$GET_USER_CAPABILITY, $CPU_CAPABILITIES, $PROCESS_CAPABILITIES

Condition Values Returned

SS$_NORMAL

The service completed successfully.

SS$_ACCVIO

The service cannot access the locations specified by one or more arguments.

SS$_INSFARG

Fewer than the required number of arguments were specified, or no operation was specified.

SS$_NOPRIV

Insufficient privilege for the attempted operation.

SS$_TOO_MANY_ARGS

Too many arguments were presented to the system service.

SS$_WASCLR

The requested user capability was already released.

$GET

Retrieves Record — The Get service retrieves a record from a file. Refer to the *VSI OpenVMS Record Management Services Reference Manual* for additional information about this service.

$GETDTI

Get Distributed Transaction Information — Returns information about the state of transactions.

Format

```
SYS$GETDTI
   [efn] ,[flags] ,iosb ,[astadr] ,[astprm] ,[log_id] ,[contxt]
   ,search ,itmlst
```

C Prototype

```
int sys$getdti
   (unsigned int efn, unsigned int flags, struct _iosb *iosb,
    void (*astadr)(__unknown_params), int astprm, unsigned int log_id [4],
    unsigned int *contxt, void *search, void *itmlst);
```

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 used.

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 described in Table 40. All undefined bits must be 0. If this argument is omitted, no flags are used.

Table 40. $GETDTI Option Flags

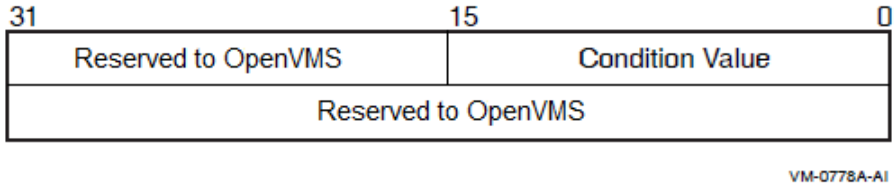
Flag Name	Description
DDTM$M_FULL_STATE	Indicates that the $GETDTI call for a specified TID is not to complete until the ABORTED or COMMITTED state can be returned. Thus, if another node or coordinating resource manager must be contacted and it is currently unreachable, this service does not return until the node can be contacted. Indicates on a wildcard search that only transactions known to be in the ABORTED or COMMITTED states are to be returned.
DDTM$M_SYNC	Specifies successful synchronous completion 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

The 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:



astadr

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

The AST routine executed when the service completes, if SS$_NORMAL is returned in R0. The *astadr* argument is the address of the entry mask of this routine. The routine is executed in the same access mode as that of the caller of the $GETDTI service.

astprm

OpenVMS usage:	user_arg
type:	longword (unsigned)

access:	read only
mechanism:	by value

The AST parameter passed to the AST routine specified by the *astadr* argument.

log_id

OpenVMS usage:	uid
type:	octaword (unsigned)
access:	read only
mechanism:	by reference

The log id of the transaction manager that is coordinating the transaction, returned as the *log_id* argument on a $DECLARE_RM operation. The *log_id* argument verifies that the recovery log returning transaction information is the same one used to record transaction state information.

If you do not specify the same log id used by the transaction manager to write transaction information, then $GETDTI will return an error.

If the *log_id* argument is specified as a zero UID, then $GETDTI will use the current active log for this node. If the specified transaction cannot be found in this log, then the returned state will be aborted. The log identifier can only be specified as zero when the DTI$_SEARCH_AS_NODE item is absent or results in a search that specifies the local node. Note that the *log_id* argument cannot be specified as a zero address.

To ensure smooth operation in a mixed-network environment, refer to the chapter entitled Managing DECdtm Services in the *VSI OpenVMS System Manager's Manual*, for information on defining node names.

contxt

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

The address of a longword used to maintain a context between sequential calls to $GETDTI. A call to $GETDTI will start a new search if the context value is zero or continue the existing search if the context is valid.

The search context is valid only after a successful call to $GETDTI and is invalidated by a subsequent call to $GETDTI.

The context is also used as input to $SETDTI after a successful call to $GETDTI. Calls to $SETDTI do not modify or invalidate the context.

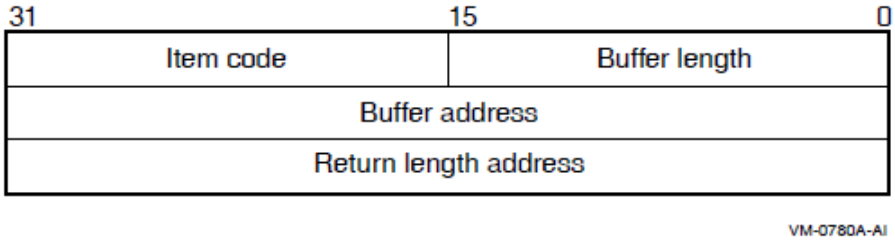
search

OpenVMS usage:	item_list_3
type:	descriptor list
access:	read only
mechanism:	by reference

Item list specifying how the search for transaction information is to be bounded.

The *search* argument is the address of a list of item descriptors, each of which describes a search item. The list of item descriptors is terminated by a longword of 0. Each item descriptor in the search item list acts as an input argument to $GETDTI and as such is only required to be read only.

The following diagram shows the format of a single item descriptor:



The following table describes the **search** item descriptor fields:

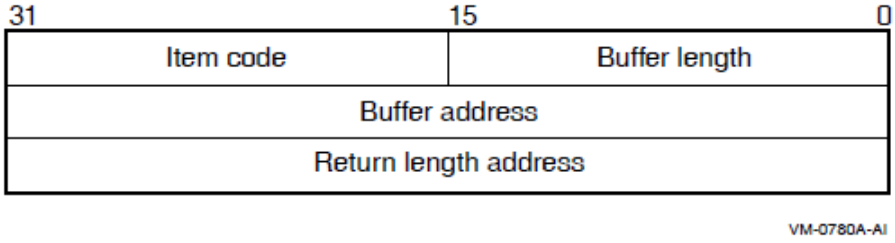
Field	Description
Buffer length	A word containing a user-supplied integer specifying the length (in bytes) of a buffer from which $GETDTI is to read the information. The length of the buffer needed depends on the item code field of the search item descriptor. If the value of buffer length is too small, $GETDTI will return an error status.
Item code	A word containing a user-supplied symbolic code specifying the search item that $GETDTI is to use. The $DTIDEF macro defines these codes. Each item code is described in the Search Item Codes section below.
Buffer address	A longword containing the user-supplied address of the buffer from which $GETDTI reads the search information.
Return length address	This longword is not used in the search item list, because all search items are read-only.

itmlst

OpenVMS usage:	item_list_3
type:	descriptor list
access:	read only
mechanism:	by reference

Item list specifying the transaction information that $GETDTI is to return. The *itmlst* argument is the address of a list of item descriptors, containing a single entry that describes an item of information. The list of item descriptors is terminated by a longword of 0. The item descriptor in the item list acts as an output argument to $GETDTI and as such is required to be writeable in caller's mode.

The following diagram shows the format of a single item descriptor:



The following table describes the *itmlst* item descriptor fields:

Field	Description
Buffer length	A word containing a user-supplied integer specifying the length (in bytes) of the buffer where $GETDTI is to write the information. The length of the buffer needed depends on the item code field of the search item descriptor. If the value of buffer length is too small, $GETDTI truncates the data and returns the condition code value SS$_BUFFEROVF.
Item code	A word containing a user-supplied symbolic code specifying the search item that $GETDTI is to use. The $DTIDEF macro defines these codes. Each item code is described in the Itmlst Item Codes section.
Buffer address	A longword containing the user-supplied address of the buffer where $GETDTI is to write the information.
return length address	A longword containing the user-supplied address of a word where $GETDTI writes return length information.

Search Item Codes

DTI$_SEARCH_AS_NODE

When you specify DTI$_SEARCH_AS_NODE, $GETDTI limits the get request to the specified node name. This can be used during cluster failover recovery processing to allow another node in the cluster to act on behalf of the failed node. The DTI$_SEARCH_AS_NODE item descriptor should point to an ASCII string containing a valid node name string.

To ensure smooth operation in a mixed-network environment, refer to the chapter entitled Managing DECdtm Services in the *VSI OpenVMS System Manager's Manual*, for information on defining node names.

DTI$_SEARCH_RESOLVED_STATE

When you specify DTI$_SEARCH_RESOLVED_STATE, the buffer address points to a transaction record that describes the search conditions for this $GETDTI call. The following fields are used from this transaction record and must be specified before $GETDTI can proceed. The $DTIDEF macro defines these fields.

Item	Description
DTI$B_PART_NAME_LEN	A byte containing the length of the participant name field DTI$T_PART_NAME.
DTI$T_PART_NAME	A character field containing DTI$B_PART_NAME_LEN characters that specifies a resource manager name. When the resource manager name string is supplied, a wildcard search can be specified. The left-most characters supplied in this string will be matched against all resource managers with the same leftmost characters. If the string entered has a length of 0, all resource managers will be selected.To ensure smooth operation in a mixed-network environment, refer to the chapter entitled Managing DECdtm Services in the *VSI OpenVMS System Manager's Manual*, for information on defining node names.
DTI$T_PART_LOG_ID	Reserved to OpenVMS.
DTI$T_TID	A 16-byte field containing the transaction identifier.

DTI$_SEARCH_RESOLVED_STATE uses the DTI$_TRANSACTION_INFORMATION item in the `itmlst` argument to write the following information:

Item	Description
DTI$B_STATE	A byte containing the state of the transaction identified by DTI$T_TID. Table 41 lists the possible values returned in the state field.
DTI$B_PART_NAME_LEN	A byte containing the length of the participant name field DTI$T_PART_NAME.
DTI$T_PART_NAME	A character field containing DTI$B_PART_NAME_LEN characters that specifies a resource manager name. When the resource manager name string is supplied, a wildcard search can be specified. The left-most characters supplied in this string will be matched against all resource managers with the same leftmost characters. If the string entered has a length of 0, all resource managers will be selected.
DTI$T_PART_LOG_ID	Reserved to OpenVMS.
DTI$T_TID	A 16-byte field containing the transaction identifier.

Table 41. $GETDTI Transaction States

Item	Description
DTI$K_STARTING	The transaction is in the starting state.
DTI$K_ACTIVE	The transaction is in the active state.
DTI$K_ONE_P_COMMITTING	The transaction is committing to phase one.
DTI$K_PREPARING	The transaction is in the preparing state.
DTI$K_PREPARED	The transaction has prepared.
DTI$K_COMMITTING	The transaction is in the committing state.
DTI$K_COMMITTED[1]	The transaction has committed.
DTI$K_ONE_P_COMMITTED	The transaction has committed to phase one.
DTI$K_ABORTING	The transaction is in the aborting state.
DTI$K_ABORTED[1]	The transaction has been aborted or forgotten. Note that, if the transaction was aborted before the call to $GETDTI, then the alternate status SS$_NOSUCHTID is returned; if the transaction was aborted during the call to $GETDTI, then the DTI$K_ABORTED state is returned in the transaction record.

[1]The DTI$K_COMMITTED and DTI$K_ABORTED transaction states are the only values that can be returned when the DDTM$M_FULL_STATE flag is specified.

Itmlst Item Codes

DTI$_TRANSACTION_INFORMATION

When you specify DTI$_TRANSACTION_INFORMATION, $GETDTI returns the following fields dependent on the search criteria established by the *search* argument.

Each record may be composed of some of the following items:

Item	Description
DTI$B_STATE	A byte containing the state of the transaction identified by DTI$T_TID. Table 41 lists the possible values returned in the state field.

Item	Description
DTI$B_PART_NAME_LEN	A byte containing the length of the participant name field DTI$T_PART_NAME.
DTI$T_PART_NAME	A character field containing DTI$B_PART_NAME_LEN characters that specifies a resource manager name.
DTI$T_PART_LOG_ID	Reserved to OpenVMS.
DTI$T_TID	A 16-byte field containing the transaction identifier.

The DTI$_TRANSACTION_INFORMATION item buffer size should be at least equal to the symbolic value DTI$S_TRANSACTION_INFORMATION defined by $DTIDEF.

Description

During recovery from a failure, a resource manager calls $GETDTI to request the state of certain transactions with which it had been involved. As part of the recovery, the resource manager identifies any unresolved transactions, so that they can be processed to completion.

The $GETDTI service returns the resolved state of in-doubt transactions to recovering resource managers. The DDTM$M_FULL_STATE flag instructs $GETDTI to return only the transactions in the DTI$K_COMMITTED or DTI$K_ABORTED states (or an error status of SS$_NOSUCHTID). The action taken depends on whether a transaction identifier is specified:

- If a transaction identifier is specified in DTI$T_TID, $GETDTI does not return until the resolved state of the transaction is available.
- If DTI$T_TID is 0, $GETDTI ignores transactions that are not in the DTI$K_COMMITTED or DTI$K_ABORTED states.

  This can mean that the $GETDTI call may have to wait for other DECdtm nodes or a coordinating CRM to become available. This is the recommended method of obtaining an in-doubt transaction outcome for recovering resource managers.

  Alternatively, $GETDTI can be used to return the current known state of transactions. Intermediate states may be returned. In particular, DTI$K_PREPARED indicates that a DECdtm node is unavailable and DTI$K_PREPARING indicates that a coordinating CRM is unavailable.

  If $GETDTI is called during normal system operation to resolve the state of transactions that have not failed, then the returned transaction state can be any of the states listed in Table 41.

  A $GETDTI call may also be used in an OpenVMS Cluster environment to perform recovery on behalf of a resource manager on a failed node. To perform this action, the DTI$_SEARCH_AS_NODE search item descriptor is used to inform $GETDTI of the node for which recovery is being performed. This action is equivalent to performing the $GETDTI request on the failed node. The use of DTI$_SEARCH_AS_NODE will perform correctly when the target node is either available or unavailable. If the target node is available when the $GETDTI call is executed, the request is re-routed to the target node for execution.

  To obtain information about transactions, $GETDTI must be given a set of search criteria. The criteria are specified as parameters and as part of an itemlist for the *search* argument using the DTI$_SEARCH_RESOLVED_STATE item descriptor. The transaction information required by $GETDTI to resolve a transaction includes:

  - Transaction Identifier (TID).

- Resource Manager name.
- Transaction Manager log identifier.
- Resource Manager log identifier.
- Optionally, a remote node name.

If you specify a TID as 0, $GETDTI assumes a wildcard operation and returns the requested information for each TID in the log that it has privilege to access, one TID per call. To perform a wildcard operation, you must call $GETDTI in a loop, testing for the condition value SS$_NOSUCHTID after each call and exiting the loop when SS$_NOSUCHTID is returned.

A resource manager is identified by its name in DTI$T_PART_NAME. A wildcard search can be specified for the resource manager with this item. The leftmost characters supplied in this string are matched against all resource managers with the same leftmost characters in their names.

If a resource manager name entered as the item has a length of 0, all resource managers are selected.

Transaction managers and resource managers maintain log files to keep a record of transactions and their related states. During recovery, it is important that the transaction manager and resource manager use matching log files. The transaction manger log identifier is returned by the $DECLARE_RM service call. It should be recorded in the resource manager's log records and supplied to $GETDTI as the value of the *log-id* argument. If the wrong resource manager log, or the wrong transaction manager log, is used, the discrepancy will result in an error from $GETDTI or $SETDTI.

The *contxt* argument is used by $GETDTI to establish a search context when it is returning the resolved state of a transaction. The search context indicates the node and transaction manager log identifier for use in a subsequent $SETDTI operation to delete the resource manager from the transaction. The search context is created when the *contxt* argument is invalid, or re-used if the *contxt* argument is valid. The search context is deleted when a call is made to $GETDTI that returns SS$_NOSUCHTID. The search context is maintained exclusively by $GETDTI and $SETDTI and should not be modified by the caller, with an exception of initially zeroing the context. SYSPRV privilege is required to retrieve or modify information about transactions with which the process is not currently associated.

Required Privileges

SYSPRV is required to retrieve information about transactions with which the process is not currently associated.

Required Quotas

BYTLM, ASTLM

Related Services

$ABORT_TRANS, $ABORT_TRANSW, $ACK_EVENT, $ADD_BRANCH, $ADD_BRANCHW, $CREATE_UID, $DECLARE_RM, $DECLARE_RMW, $END_BRANCH, $END_BRANCHW, $END_TRANS, $END_TRANSW, $FORGET_RM, $FORGET_RMW, $GETDTIW, $GET_DEFAULT_TRANS, $JOIN_RM, $JOIN_RMW, $SETDTI, $SETDTIW, $SET_DEFAULT_TRANS, $SET_DEFAULT_TRANSW, $START_BRANCH,

$START_BRANCHW, $START_TRANS, $START_TRANSW, $TRANS_EVENT, $TRANS_EVENTW

Condition Values Returned

SS$_NORMAL

If returned in R0, the request was successfully queued. If 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 to the caller.

SS$_BADLOGVER

There is an invalid or unsupported log version.

SS$_BADPARAM

The option flags, SEARCH or ITMLST are invalid.

SS$_BUGCHECK

A failure has occurred during the processing of the request.

SS$_EXASTLM

The process AST limit (ASTLM) was exceeded.

SS$_ILLEFC

The event flag number was invalid.

SS$_INSFARGS

A required argument was missing.

SS$_INSFMEM

There was insufficient system dynamic memory for the operation.

SS$_INVLOG

The log format is invalid.

SS$_NOSUCHFILE

The transaction manager log cannot be found.

SS$_NOSUCHNODE

The subordinate DECnet node is unknown.

SS$_NOSUCHPART

The participant is not part of the transaction.

SS$_NOSUCHTID

The designated TID is unknown.

SS$_NOSYSPRV

The caller is not authorized to examine the specified transaction.

SS$_PROTOCOL

There is a message protocol error.

SS$_REMOTE_PROC

There was an attempt to use a node when it is not part of the OpenVMS Cluster.

SS$_REMRSRC

There are insufficient resources at the remote node.

SS$_UNREACHABLE

A superior node is unreachable.

$GETDTIW

Get Distributed Transaction Information and Wait — Returns information about the resolved state of transactions and the process default transaction identifier. $GETDTIW always waits for the request to complete before returning to the caller. Other than this, it is identical to $GETDTI.

Format

```
SYS$GETDTIW
   [efn] ,[flags] ,iosb ,[astadr] ,[astprm] ,[contxt] ,[log_id]
   ,search ,itmlst
```

C Prototype

```
int sys$getdtiw
   (unsigned int efn, unsigned int flags, struct _iosb *iosb,
    void (*astadr)(__unknown_params), int astprm, unsigned int log_id [4],
    unsigned int *contxt, void *search, void *itmlst);
```

$GETDVI

Get Device/Volume Information — Returns information related to the primary and secondary device characteristics of an I/O device. All **pathname**-related information pertains only to Alpha and Integrity server systems. For synchronous completion, use the Get Device/Volume Information and Wait ($GETDVIW) service. The $GETDVIW service is identical to the $GETDVI service in every way except that $GETDVIW returns to the caller with the requested information. For additional information

about system service completion, see the Synchronize ($SYNCH) service. Note About Item Codes: For item codes that return a string data type, failure to pass in a buffer that is large enough to hold the returned data results in silent data truncation. When $GETDVI completes, VSI recommends that you check the returned length field of an item list descriptor for each item code that can return a string. If the returned length is equal to the size of the buffer allocated to hold the returned data, the data might have been truncated. In that case, call $GETDVI iteratively with a larger buffer until the length of the returned data is less than the size of the buffer allocated. Unless the description of an item code specifies otherwise, VSI recommends that you use a buffer of 32 bytes to hold the returned string. $GETDVI pads the unused portion of the buffer with null characters.

Format

```
SYS$GETDVI
   [efn] ,[chan] ,[devnam] ,itmlst [,iosb] [,astadr] [,astprm] [,nullarg]
   [,pathname]
```

C Prototype

```
int sys$getdvi(
        unsigned int efn,
        unsigned short int chan,
        void *devnam,
        void *itmlst,
        struct _iosb *iosb,
        void (*astadr)(__unknown_params),
        int astprm,
        struct _generic_64 *nullarg,
        __optional_params);
```

Arguments

efn

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

Number of the event flag to be set when $GETDVI returns the requested information. The *efn* argument is a longword containing this number; however, $GETDVI uses only the low-order byte.

Upon request initiation, $GETDVI clears the specified event flag (or event flag 0 if *efn* was not specified). Then, when $GETDVI returns the requested information, it sets the specified event flag (or event flag 0).

VSI strongly recommends the use of the EFN$C_ENF "no event flag" value as the event flag if you are not using an event flag to externally synchronize with the completion of this system service call. The $EFNDEF macro defines EFN$C_ENF. For more information, see the *VSI OpenVMS Programming Concepts Manual*.

chan

OpenVMS usage:	channel

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

Number of the I/O channel assigned to the device about which information is desired. The *chan* argument is a word containing this number.

To identify a device to $GETDVI, you can specify either the *chan* or *devnam* argument, but you should not specify both. If you specify both arguments, the *chan* argument is used.

If you specify neither *chan* nor *devnam*, $GETDVI uses a default value of 0 for *chan*.

devnam

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

The name of the device about which $GETDVI is to return information. The *devnam* argument is the address of a character string descriptor pointing to this name string.

The device name string can be either a physical device name or a logical name. If the first character in the string is an underscore (_), the string is considered a physical device name; otherwise, the string is considered a logical name and logical name translation is performed until either a physical device name is found or the system default number of translations has been performed.

If the device name string contains a colon (:), the colon and the characters that follow it are ignored.

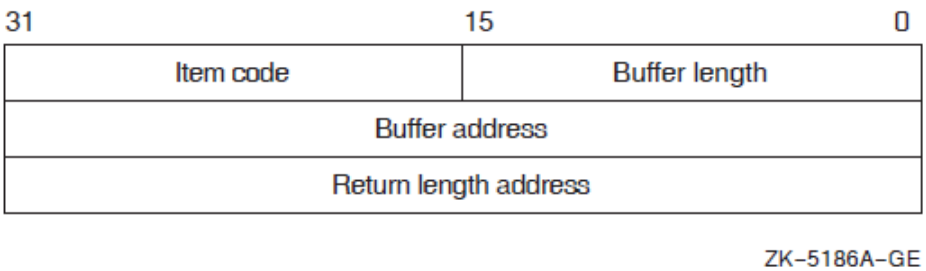
To identify a device to $GETDVI, you can specify either the *chan* or *devnam* argument, but you should not specify both. If both arguments are specified, the *chan* argument is used.

If you specify neither *chan* nor *devnam*, $GETDVI uses a default value of 0 for *chan*.

itmlst

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

Item list specifying which information about the device is to be returned. The *itmlst* argument is the address of a list of item descriptors, each of which describes an item of information. The list of item descriptors is terminated by a longword of 0. The following diagram depicts the format of a single item descriptor.



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 $GETDVI is to write the information. The length of the buffer needed depends upon the item code specified in the item code field of the item descriptor. If the value of buffer length is too long, $GETDVI truncates the data.
Item code	A word containing a user-supplied symbolic code specifying the item of information that $GETDVI is to return. The $DVIDEF macro defines these codes. Each item code is described in the Item Codes section.
Buffer address	A longword containing the user-supplied address of the buffer in which $GETDVI is to write the information.
Return length address	A longword containing the user-supplied address of a word in which $GETDVI is to write the information.

iosb

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

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

When you specify the *iosb* argument, $GETDVI sets the quadword to 0 upon request initiation. Upon request completion, a condition value is returned to the first longword; the second longword is reserved to OpenVMS.

Though this argument is optional, VSI strongly recommends that you specify it, for the following reasons:

- If you are using an event flag to signal the completion of the service, you can test the I/O status block for a condition value to be sure that the event flag was not set by an event other than service completion.
- If you are using the $SYNCH service to synchronize completion of the service, the I/O status block is a required argument for $SYNCH.
- The condition value returned in R0 and the condition value returned in the I/O status block provide information about different aspects of the call to the $GETDVI service. The condition value returned in R0 gives you information about the success or failure of the service call itself; the condition value returned in the I/O status block gives you information about the success or failure of the service operation. Therefore, to accurately assess the success or failure of the call to $GETDVI, you must check the condition values returned in both R0 and the I/O status block.

astadr

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

mechanism:	by reference

AST service routine to be executed when $GETDVI 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 $GETDVI 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.

nullarg

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

Placeholding argument reserved to OpenVMS.

pathname

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

On Alpha and Integrity server systems, the name of the path about which $GETDVI is to return information. The *pathname* argument is the address of a character string descriptor pointing to this name string. The *pathname* can be used with either the *chan* or the *devnam* argument.

Check the definitions of the item codes to see if the *pathname* argument is used. In general, item codes that return information that can vary by path make use of the *pathname* argument. Use the SHOW DEVICE /FULL command, the SYS$DEVICE_PATH_SCAN system service, or the F$MULTIPATH DCL lexical function to see the paths for a multipath device.

If the *pathname* argument is used, it is validated against the existing paths for the device specified. If the path does not exist, the error SS$_NOSUCHPATH is returned, even if the item code or codes used do not make use of the *pathname* argument.

Item Codes

DVI$_ACCESSTIMES_RECORDED

On Alpha and Integrity server systems, returns an unsigned longword, which is interpreted as Boolean. A value of 1 indicates the volume supports the recording of access times.

DVI$_ACPPID

Returns the ACP process ID as an unsigned integer longword.

DVI$_ACPTYPE

Returns the ACP type code as an unsigned integer longword. The following symbols define each of the ACP type codes that $GETDVI can return:

Symbol	Description
DVI$C_ACP_F11V1	Files-11 Level 1
DVI$C_ACP_F11V2	Files-11 Level 2
DVI$C_ACP_F11V3	Files-11 presentation of ISO 9660
DVI$C_ACP_F11V4	Files-11 presentation of High Sierra
DVI$C_ACP_F11V5	Files-11 Level 5
DVI$C_ACP_F11V6	Files-11 Level 6
DVI$C_ACP_F64	Files 64 support for Spiralog
DVI$C_ACP_HBS	Not used
DVI$C_ACP_HBVS	Host Based Volume Shadowing
DVI$C_ACP_MTA	Magnetic tape
DVI$C_ACP_NET	Networks
DVI$C_ACP_REM	Remote I/O
DVI$C_ACP_UCX	ACP for TCP/IP Services for OpenVMS

DVI$_ALL

Returns an unsigned longword, interpreted as Boolean. A value of 1 indicates that the device is allocated.

DVI$_ALLDEVNAM

Returns the allocatable device name as a string.

The allocatable device name, sometimes called the allocation-class device name, uniquely identifies each device that is currently accessible to any given node in an OpenVMS cluster or to a single-node OpenVMS system. This item code generates the same name on every system for a device that is accessible on multiple cluster nodes through multiple paths or servers.

The device name string returned by the DVI$_ALLDEVNAM item code is recommended for use with the $ASSIGN service, regardless of whether the system is clustered or whether the device is cluster accessible, because it returns the same string value for all conditions.

Another possible use for the DVI$_ALLDEVNAM item code might be an application wherein processes need to coordinate their access to devices (not volumes) using the lock manager. In this case, the program would make the device a resource to be locked by the lock manager, specifying as the resource name the following concatenated components: (1) a user facility prefix followed by an underscore character and (2) the allocatable device name of the device.

Depending on the device type and characteristics, the format of the returned name may include an allocation class or node name prefix.

Use the name returned by the DVI$_DEVLOCKNAM item code to coordinate access to volumes.

DVI$_ALLOCLASS

Returns the allocation class of the host as a longword integer between 0 and 32767. An allocation class is a unique number between 0 and 32767 that the system manager assigns to a pair of hosts and the dual-pathed devices that the hosts make available to other nodes in the cluster.

The allocation class provides a way for you to access dual-pathed devices through either of the hosts that act as servers to the cluster. In this way, if one host of an allocation class set is not available, you can gain access to a device specified by that allocation class through the other host of the allocation class. You do not have to be concerned about which host of the allocation class provides access to the device. Specifically, the device name string has the following format:

```
$allocation_class$device_name
```

For a detailed discussion of allocation classes, see *VSI OpenVMS Cluster Systems*.

DVI$_ALT_HOST_AVAIL

Returns a longword that is interpreted as Boolean. A value of 1 indicates that the host serving the alternate path is available; a value of 0 indicates that it is not available.

The host is the node that makes the device available to other nodes in the OpenVMS Cluster system. A host node can be a VMS system with an MSCP server or an HSC50 controller.

A dual-pathed device is one that is made available to the cluster by two hosts. Each of the hosts provides access (serves a path) to the device for users. One host serves the primary path; the other host serves the alternate path. The primary path is the path that the system creates through the first available host.

Do not be concerned with which host provides access to the device. When accessing a device, you specify the allocation class of the desired device, not the name of the host that serves it.

If the host serving the primary path fails, the system automatically creates a path to the device through the alternate host.

DVI$_ALT_HOST_NAME

Returns the name of the host serving the alternate path as a string.

For more information about hosts, dual-pathed devices, and primary and alternate paths, see the description of the DVI$_ALT_HOST_AVAIL item code.

DVI$_ALT_HOST_TYPE

Returns a four-byte string containing an arbitrary text description of the OpenVMS host or cluster storage controller that is serving this storage device. The particular text string displayed was provided to the local OpenVMS system by the remote System Communications Services (SCS) server.

Storage devices are served within an OpenVMS Cluster utilizing the SCS protocol, with the server running on a remote OpenVMS system or on an SCS-capable storage controller. The alternate host type value displayed is determined from the local cluster storage hardware configuration, and indicates the type of devices serving the storage for the available SCS path or paths to the particular storage device.

Item codes in this group include: DVI$_HOST_TYPE, DVI$_ALT_HOST_TYPE, DVI$_HOST_AVAIL, and DVI$_ALT_HOST_AVAIL. Also related is DVI$_HOST_NAME.

VSI suggests this item code be used solely for display purposes, and do not use it during an attempt to determine the particular cluster hardware configuration.

DVI$_AVAILABLE_PATH_COUNT

On Alpha and Integrity server systems, returns the number of available, working paths for a multipath-capable device as an unsigned longword.

DVI$_CLUSTER

Returns the volume cluster size as an unsigned integer longword. This item code is applicable only to disks.

DVI$_CYLINDERS

Returns the number of cylinders on the volume as an unsigned integer longword. This item code is applicable only to disks.

DVI$_DEVBUFSIZ

Returns the device buffer size (for example, the width of a terminal or the block size of a tape) as an unsigned integer longword.

DVI$_DEVCHAR

Returns device-independent characteristics as a 4-byte bit vector. Each characteristic is represented by a bit. When $GETDVI sets a bit, the device has the corresponding characteristic. Each bit in the vector has a symbolic name. The $DEVDEF macro defines the following symbolic names:

Symbol	Description
DEV$V_REC	Device is record oriented.
DEV$V_CCL	Device is a carriage control device.
DEV$V_TRM	Device is a terminal.
DEV$V_DIR	Device is directory structured.
DEV$V_SDI	Device is single-directory structured.
DEV$V_SQD	Device is sequential and block oriented.
DEV$V_SPL	Device is being spooled.
DEV$V_OPR	Device is an operator.
DEV$V_RCT	Disk contains Revector Cache Table (RCT). This bit is set for every DSA disk.
DEV$V_NET	Device is a network device.
DEV$V_FOD	Device is files oriented.
DEV$V_DUA	Device is dual ported.
DEV$V_SHR	Device is shareable.
DEV$V_GEN	Device is a generic device.
DEV$V_AVL	Device is available for use.

Symbol	Description
DEV$V_MNT	Device is mounted.
DEV$V_MBX	Device is a mailbox.
DEV$V_DMT	Device is marked for dismount.
DEV$V_ELG	Device has error logging enabled.
DEV$V_ALL	Device is allocated.
DEV$V_FOR	Device is mounted foreign.
DEV$V_SWL	Device is software write locked.
DEV$V_IDV	Device can provide input.
DEV$V_ODV	Device can provide output.
DEV$V_RND	Device allows random access.
DEV$V_RTM	Device is a real-time device.
DEV$V_RCK	Device has read-checking enabled.
DEV$V_WCK	Device has write-checking enabled.

Note that each device characteristic has its own individual $GETDVI item code with the format DVI$_xxxx, where `xxxx` are the characters following the underscore character in the symbolic name for that device characteristic.

For example, when you specify the item code DVI$_REC, $GETDVI returns a longword value that is interpreted as Boolean. If the value is 0, the device is not record oriented; if the value is 1, it is record oriented. This information is identical to that returned in the DEV$V_REC bit of the longword vector specified by the DVI$_DEVCHAR item code.

The buffer must specify a longword for all of these device-characteristic item codes.

DVI$_DEVCHAR2

Returns additional device-independent characteristics as a 4-byte bit vector. Each bit in the vector, when set, corresponds to a symbolic name. The $DEVDEF macro defines the following symbolic names:

Symbol	Description
DEV$V_CLU	Device is available clusterwide.
DEV$V_DET	Device is detached terminal.
DEV$V_RTT	Device has remote terminal UCB extension.
DEV$V_CDP	Dual-pathed device with two UCBs.
DEV$V_2P	Two paths are known to this device.
DEV$V_MSCP	Device accessed using MSCP (disk or tape). Before using this bit to differentiate between types of disk and tape devices, be sure that no other more appropriate differentiation mechanism exists.
DEV$V_SSM	Device is a shadow set member.
DEV$V_SRV	Device is served by the MSCP server.
DEV$V_RED	Device is redirected terminal.
DEV$V_NNM	Device has node$ prefix.
DEV$V_WBC	Device supports write-back caching.

Symbol	Description
DEV$V_WTC	Device supports write-through caching.
DEV$V_HOC	Device supports host caching.
DEV$V_LOC	Device accessible by local (non-emulated) controller.
DEV$V_DFS	Device is DFS-served.
DEV$V_DAP	Device is DAP accessed.
DEV$V_NLT	Device is not-last-track; that is, it has no bad block. Information is on its last track.
DEV$V_SEX	Device (tape) supports serious exception handling.
DEV$V_SHD	Device is a member of a host-based shadow set.
DEV$V_VRT	Device is a shadow set virtual unit.
DEV$V_LDR	Loader present (tapes).
DEV$V_NOLB	Device ignores server load balancing requests.
DEV$V_NOCLU	Device will never be available clusterwide.
DEV$V_VMEM	Virtual member of a constituent set.
DEV$V_SCSI	Device is an SCSI device.
DEV$V_WLG	Device has write-logging capability.
DEV$V_NOFE	Device does not support forced error.

DVI$_DEVCLASS

Returns the device class as an unsigned integer longword. Each class has a corresponding symbol. The $DCDEF macro defines these symbols. The following table describes each device class symbol:

Symbol	Description
DC$_DISK	Disk device
DC$_TAPE	Tape device
DC$_SCOM	Synchronous communications device
DC$_CARD	Card reader
DC$_TERM	Terminal
DC$_LP	Line printer
DC$_REALTIME	Real-time
DC$_MAILBOX	Mailbox
DC$_MISC	Miscellaneous device
DC$_BUS	Peripheral adapter

DVI$_DEVDEPEND

Returns device-dependent characteristics as a 4-byte bit vector. To determine what information is returned for a particular device, see the *VSI OpenVMS I/O User's Reference Manual*.

Note that, for terminals only, individual $GETDVI item codes are provided for most of the informational items returned in the DVI$_DEVDEPEND longword bit vector. The names of these item codes have the format DVI$_TT_xxxx, where `xxxx` is the characteristic name. The same characteristic name

follows the underscore character in the symbolic name for each bit (defined by the $TTDEF macro) in the DVI$_DEVDEPEND longword. For example, the DVI$_TT_NOECHO item code returns the same information as that returned in the DVI$_DEVDEPEND bit whose symbolic name is TT$V_NOECHO.

Each such item code requires that the buffer specify a longword value, which is interpreted as Boolean. A value of 0 indicates that the terminal does not have that characteristic; a value of 1 indicates that it does.

The list of these terminal-specific item codes follows this list of item codes.

DVI$_DEVDEPEND2

Returns additional device-dependent characteristics as a 4-byte bit vector. To determine what information is returned for a particular device, see the *VSI OpenVMS I/O User's Reference Manual*.

Note that, for terminals only, individual $GETDVI item codes are provided for most of the informational items returned in the DVI$_DEVDEPEND2 longword bit vector. As with DVI$_DEVDEPEND, the same characteristic name appears in the item code as appears in the symbolic name defined for each bit in the DVI$_DEVDEPEND2 longword, except that in the case of DVI$_DEVDEPEND2, the symbolic names for bits are defined by the $TT2DEF macro.

The list of these terminal-specific item codes follows this list of item codes.

DVI$_DEVICE_MAX_IO_SIZE

On Alpha and Integrity server systems, returns the maximum unsegmented transfer size supported by the device's device driver as an unsigned integer longword. Note that although this value is the absolute maximum size supported by the device driver, other software layers (RMS and XFC, for example) might impose lower maximum values limiting the maximum unsegmented transfer size.

DVI$_DEVICE_TYPE_NAME

Returns a string identifying the type of device about which information was requested. VSI recommends a buffer size of 64 bytes to return the device type name.

DVI$_DEVLOCKNAM

Returns the device lock name, which is a 64-byte string. The device lock name uniquely identifies each volume or volume set in an OpenVMS Cluster system or in a single-node system. This item code is applicable only to disks.

The item code is applicable to all disk volumes and volume sets: mounted, not mounted, mounted shared, mounted private, or mounted foreign.

The device lock name is assigned to a volume when it is first mounted, and you cannot change this name, even if the volume name itself is changed. This allows any process on any node in an OpenVMS Cluster system to access a uniquely identified volume.

One use for the device lock name might be in an application wherein processes need to coordinate their access to files using the lock manager. In this case, the program would make the file a resource to be locked by the lock manager, specifying as the resource name the following concatenated components: (1) a user facility prefix followed by an underscore character, (2) the device lock name of the volume on which the file resides, and (3) the file ID of the file.

DVI$_DEVNAM

Returns a displayable device name as a string.

Although DVI$_DEVNAM returns a displayable device name, it might not return a unique name for cluster-accessible devices. For this reason, the DVI$_DEVNAM item code is not recommended for use with the $ASSIGN service. Use the DVI$_ALLDEVNAM item code with the $ASSIGN service.

Depending on the device type and characteristics, the format of the returned name may include an allocation class or node name prefix.

The DVI$_DEVNAM item code is provided solely for compatibility with existing programs from previous OpenVMS releases; therefore, VSI recommends that you use the DVI$_ALLDEVNAM, DVI$_FULLDEVNAM, or DVI$_DISPLAY_DEVNAM item codes, depending on the function required.

DVI$_DEVSTS

Returns device-dependent status information as a 4-byte bit vector. The $UCBDEF macro defines symbols for the status bits. For this device-dependent information, see the *VSI OpenVMS I/O User's Reference Manual*.

DVI$_DEVTYPE

Returns the device type as an unsigned integer longword. The $DCDEF macro defines symbols for the device types.

DVI$_DFS_ACCESS

Returns a Boolean value indicating whether a device is a DFS served disk. A value of 0 indicates that the device is a DFS served disk; a value of 1 indicates that the device is not.

This information allows you to determine if a function works on remote disk devices with DFS. Access control lists (ACLs), for example, cannot be set or displayed on local disk devices with DFS.

DVI$_DISPLAY_DEVNAM

Returns a displayable device name as a string. VSI recommends a buffer size of 64 bytes to return the displayable device name.

The format of the returned name includes the allocatable device name for the device (see DVI$_ALLDEVNAM) followed by any current primary or secondary path information for the device.

Depending on the device type and characteristics, the format of the returned name may include an allocation class or node name prefix.

The device name string returned by the DVI$_DISPLAY_DEVNAM item code cannot be used for the $ASSIGN service. Use the DVI$_ALLDEVNAM item code to form the device name for the $ASSIGN service.

DVI$_ERASE_ON_DELETE

On Alpha and Integrity server systems, returns an unsigned longword, which is interpreted as Boolean. A value of 1 indicates that disk blocks are zeroed upon file deletion on the volume.

DVI$_ERRCNT

Returns the error count of the device as an unsigned longword. The error count might have been reset with the SET DEVICE/RESET=ERRCNT command. If the error count has been reset, use the SHOW DEVICE/FULL command to display the date and time that the error count was reset.

On Alpha and Integrity server systems, this item code can be used with the PATHNAME parameter. If the PATHNAME parameter is omitted, the summation of the error counts for all paths in a multipath device is returned (which was the behavior prior to the introduction of the PATHNAME parameter). If the PATHNAME parameter is specified, only the error count for that path is returned.

DVI$_ERROR_RESET_TIME

On Alpha and Integrity server systems, returns the time at which the error count was reset as a 64-bit VMS date/time format.

DVI$_EXPSIZE

On Alpha and Integrity server systems, returns the current expansion limit on the volume as an unsigned integer longword.

DVI$_FC_HBA_FIRMWARE_REV

On Alpha and Integrity server systems, returns the firmware revision information of a fibre channel host bus adapter as a string. A null string is returned for all other devices.

DVI$_FC_NODE_NAME, **DVI$_FC_PORT_NAME**

On Alpha and Integrity server systems, returns the Fibre Channel Node or Port name for the Fibre Channel Host Bus Adapter as a string for each of these values.

DVI$_FREEBLOCKS

Returns the number of free blocks on a disk as an unsigned integer longword. This item code is applicable only to disks.

DVI$_FULLDEVNAM

Returns a displayable device name as a string.

DVI$_FULLDEVNAM returns a displayable device name, which includes appropriate identification to distinguish cluster-accessible devices by including the allocation class or node name in the device string.

One use for the DVI$_FULLDEVNAM item code might be to retrieve the name of a device in order to have that name displayed on a terminal. However, do not use this name as a resource name in input to the lock manager; instead, use the name returned by the DVI$_DEVLOCKNAM item code for locking volumes and the name returned by DVI$_ALLDEVNAM for locking devices.

Depending on the device type and characteristics, the format of the returned name may include an allocation class or node name prefix.

The name returned by the DVI$_FULLDEVNAM item code can be used for the $ASSIGN service. VSI recommends that you use the DVI$_ALLDEVNAM item code to form the name for the $ASSIGN service, so that the name appears the same regardless of which OpenVMS Cluster node is accessing the device.

DVI$_HARDLINKS_SUPPORTED

On Alpha and Integrity server systems, returns an unsigned longword, which is interpreted as Boolean. A value of 1 indicates that POSIX hardlinks, rather than aliases, are supported.

DVI$_HOST_AVAIL

Returns a longword, which is interpreted as Boolean. A value of 1 indicates that the host serving the primary path is available; a value of 0 indicates that it is not available.

For more information about hosts, dual-pathed devices, and primary and alternate paths, see the description of the DVI$_ALT_HOST_AVAIL item code.

DVI$_HOST_COUNT

Returns the number of hosts that make the device available to other nodes in the OpenVMS Cluster system as a longword integer. One or two hosts, but no more, can make a device available to other nodes in the cluster.

For more information about hosts, dual-pathed devices, and primary and alternate paths, see the description of the DVI$_ALT_HOST_AVAIL item code.

DVI$_HOST_NAME

Returns the name of the host serving the primary path as a string.

For more information about hosts, dual-pathed devices, and primary and alternate paths, see the description of the DVI$_ALT_HOST_AVAIL item code.

DVI$_HOST_TYPE

Returns a four-byte string containing an arbitrary text description of the OpenVMS host or cluster storage controller that is serving this storage device. The particular text string displayed was provided to the local OpenVMS system by the remote System Communications Services (SCS) server.

Storage devices are served within an OpenVMS Cluster utilizing the SCS protocol, with the server running on a remote OpenVMS system or on an SCS-capable storage controller. The host type value displayed is determined from the local cluster storage hardware configuration, and indicates the type of devices serving the storage for the available SCS path or paths to the particular storage device.

Item codes in this group include: DVI$_HOST_TYPE, DVI$_ALT_HOST_TYPE, DVI$_HOST_AVAIL, and DVI$_ALT_HOST_AVAIL. Also related is DVI$_HOST_NAME.

VSI suggests this item code be used solely for display purposes; do not use it during an attempt to determine the particular cluster hardware configuration.

DVI$_LAN_ALL_MULTICAST_MODE

On Alpha and Integrity server systems, returns an unsigned longword, which is interpreted as Boolean. A value of 1 indicates that the device is enabled to receive all multicast packets, rather than only packets addressed to enabled multicast addresses.

DVI$_LAN_AUTONEG_ENABLED

On Alpha and Integrity server systems, returns an unsigned longword, which is interpreted as Boolean. A value of 1 indicates that the device is set to autonegotiate the speed and duplex settings.

DVI$_LAN_DEFAULT_MAC_ADDRESS

On Alpha and Integrity server systems, returns the default media access control (MAC) address of the device as a string.

VI$_LAN_FULL_DUPLEX

On Alpha and Integrity server systems, returns an unsigned longword, which is interpreted as Boolean. A value of 1 indicates that the device is operating in full-duplex mode.

DVI$_LAN_JUMBO_FRAMES_ENABLED

On Alpha and Integrity server systems, returns an unsigned longword, which is interpreted as Boolean. A value of 1 indicates that jumbo frames are enabled on the device.

DVI$_LAN_LINK_STATE_VALID

On Alpha and Integrity server systems, returns an unsigned longword, which is interpreted as Boolean. A value of 1 indicates that the device driver for the LAN device correctly maintains the link status.

The device drivers for the following adapters do not maintain the link status: DEMNA, any TURBOchannel adapter, any PCMCIA adapter, any Token Ring adapter, Galaxy shared memory, TGEC, DE205, DE422, DE425, DE434, DE435, DE500-XA. (The -AA and -BA variants *are* supported.)

DVI$_LAN_LINK_UP

On Alpha and Integrity server systems, returns an unsigned longword, which is interpreted as Boolean. A value of 1 indicates that the link is up. This item code is valid only for the template device (that is, unit number 0); this item returns 0 if used against a nontemplate LAN device.

This item is supported only on newer adapters. To determine whether or not a particular LAN device supports DVI$_LAN_LINK_UP, you must first call $GETDVI using the item code DVI$_LAN_LINK_STATE_VALID. For more information, see that item code.

If DVI$_LAN_LINK_UP is used on an adapter that does not maintain the link status, the returned status from $GETDVI will be SS$_UNSUPPORTED. For more information, see DVI$_LAN_LINK_VALID.

DVI$_LAN_MAC_ADDRESS

On Alpha and Integrity server systems, returns the current MAC address of the device as a string. Note that certain protocols, such as DECnet Phase IV, change the default MAC address to one based on the DECnet address. For DECnet Phase IV, the MAC address is in the form AA-00-04-00-nn-mm, where nn-mm is calculated by multiplying the DECnet Phase IV area number by 1024, adding the node number (that is, the number following the dot), taking the hexadecimal representation of that sum, and swapping the two bytes.

For example, a node with a DECnet Phase IV address of 3.31 gives a decimal representation of 3103 (the value corresponding to the system parameter SCSSYSTEMID) and a hexadecimal representation of 0C1F. If DECnet Phase IV is started on this system, the MAC address of any LAN adapter on this system running DECnet is AA-00-04-00-1F-0C. < PARAMITEM>(DVI$_LAN_PROMISCUOUS_MODE) On Alpha and Integrity server systems, returns an unsigned longword, which is interpreted as Boolean. A value of 1 indicates that the device is enabled to receive all packets, not just packets addressed to the MAC address and enabled multicast addresses.

DVI$_LAN_PROTOCOL_NAME

On Alpha and Integrity server systems, returns the name of the LAN protocol running on the device as a string.

DVI$_LAN_PROTOCOL_TYPE

On Alpha and Integrity server systems, returns the LAN protocol type running on the device as a string.

DVI$_LAN_SPEED

On Alpha and Integrity server systems, returns the speed of the LAN device (in Mb/s) as an unsigned longword. Valid values are 4, 10, 16, 100, 1000, and 10000.

DVI$_LOCKID

Returns the lock ID of the lock on a disk as an unsigned longword. The lock manager locks a disk if it is available to all nodes in an OpenVMS Cluster system and it is either allocated or mounted. A disk is available to all nodes in an OpenVMS Cluster system if, for example, it is served by an HSC controller or MSCP server or if it is a dual-ported MASSBUS disk.

DVI$_LOGVOLNAM

Returns the logical name of the volume or volume set as a string.

DVI$_MAILBOX_BUFFER_QUOTA

On Alpha and Integrity server systems, returns the current mailbox quota as an unsigned integer longword.

DVI$_MAILBOX_INITIAL_QUOTA

On Alpha and Integrity server systems, returns the initial mailbox quota as an unsigned integer longword.

DVI$_MAXBLOCK

Returns the maximum number of blocks on the volume as an unsigned integer longword. This item code is applicable only to disks.

DVI$_MAXFILES

Returns the maximum number of files on the volume as an unsigned integer longword. This item code is applicable only to disks.

DVI$_MEDIA_ID

Returns the nondecoded media ID as a longword. This item code is applicable only to disks and tapes.

DVI$_MEDIA_NAME

Returns the name of the volume type (for example, RK07 or TA78) as a string. This item code is applicable only to disks and tapes.

DVI$_MEDIA_TYPE

Returns the device name prefix of the volume (for example, DM for an RK07 device or MU for a TA78 device) as a string. This item code is applicable only to disks and tapes.

DVI$_MOUNT_TIME

On Alpha and Integrity server systems, returns the time at which the volume was mounted. Because the returned time is in the standard 64-bit absolute time format, specify 8 (bytes) for the buffer length field in the item descriptor.

Note that for volumes mounted in a cluster, only the time of the initial mount is recorded; the time of any subsequent mount is not recorded.

DVI$_MOUNTCNT

Returns the mount count for the volume as an unsigned integer longword and displays the number of times the volume has been mounted on the local system.

The value of MOUNTCNT displayed by the SHOW DEVICE command is the total of all mounts of the volume across all members of the cluster.

DVI$_MOUNTVER_ELIGIBLE

On Alpha and Integrity server systems, returns an unsigned longword, which is interpreted as Boolean. A value of 1 indicates the volume is eligible to undergo mount verification. A volume mounted with either the /FOREIGN or /NOMOUNT_VERIFICATION qualifier is not subject to mount verification.

DVI$_MPDEV_AUTO_PATH_SW_CNT

On Alpha and Integrity server systems, returns as an unsigned longword the number of times a multipath device has automatically switched paths due to an I/O error, or as the result of automatically "failing back" to a local path from a remote path after the local path became available.

DVI$_MPDEV_CURRENT_PATH

On Alpha and Integrity server systems, returns the current path name for multipath devices as a string.

If the device is not part of a multipath set, $GETDVI will return the name of the device path if the class driver for this device supports path names. SYS$DKDRIVER, SYS$DUDRIVER, SYS$MKDRIVER, and SYS$GKDRIVER support path names.

If the class driver for the device does not support path names, $GETDVI returns a null string.

DVI$_MPDEV_MAN_PATH_SW_CNT

On Alpha and Integrity server systems, returns as an unsigned longword the number of times a multipath device has manually switched paths due to a $SET DEVICE /PATH /SWITCH command or the use of the $SET_DEVICE system service.

DVI$_MSCP_UNIT_NUMBER

Returns the internal coded value for MSCP unit numbers as a longword integer. This item code is reserved to OpenVMS.

DVI$_MT3_DENSITY

Returns the density of the device. Valid for tapes only. This code is an unsigned longword integer.

DVI$_MT3_SUPPORTED

The return value of 1 indicates that the device supports tape density codes defined by MT3DEF. Valid for tapes only. This code is an unsigned longword integer.

DVI$_MULTIPATH

On Alpha and Integrity server systems, returns a longword, interpreted as Boolean. A value of 1 indicates the device is a member of a multipath set.

DVI$_MVSUPMSG

On Alpha and Integrity server systems, returns an unsigned longword, interpreted as Boolean. A value of 1 indicates that mount verification OPCOM messages are currently being suppressed on this device. See the MVSUPMSG_INTVL AND MVSUPMSG_NUM system parameters for more information on the suppression of mount verification messages.

DVI$_NEXTDEVNAM

Returns the device name of the next volume in the volume set as a string. The node name is also returned. This item code is applicable only to disks.

DVI$_NOCACHE_ON_VOLUME

On Alpha and Integrity server systems, returns an unsigned longword, which is interpreted as Boolean. A value of 1 indicates the volume is mounted with all caching disabled.

DVI$_NOHIGHWATER

On Alpha and Integrity server systems, returns an unsigned longword, which is interpreted as Boolean. A value of 1 indicates that high-water marking is disabled on the volume.

DVI$_NOSHARE_MOUNTED

On Alpha and Integrity server systems, returns an unsigned longword, which is interpreted as Boolean. A value of 1 indicates the volume is mounted /NOSHARE.

DVI$_NOXFCCACHE_ON_VOLUME

On Alpha and Integrity server systems, returns a Boolean value indicating the XFC caching status of the volume. A value of 1 indicates that the XFC caching is disabled on the volume; a value of 0 indicates that the XFC caching is enabled on the volume.

DVI$_ODS2_SUBSET0

On Alpha and Integrity server systems, returns an unsigned longword, which is interpreted as Boolean. A value of 1 indicates the volume supports only a subset of the ODS-2 file structure.

DVI$_ODS5

On Alpha and Integrity server systems, returns an unsigned longword, which is interpreted as Boolean. A value of 1 indicates the volume is mounted ODS-5.

DVI$_OPCNT

Returns the operation count of the device as an unsigned longword. Note that the operation count might have been reset with the SET DEVICE/RESET=OPCNT command.

On Alpha and Integrity server systems, this item code can be used with the PATHNAME parameter. If the PATHNAME parameter is omitted, the summation of the operation counts for all paths in a multipath device is returned (which was the behavior prior to the introduction of the PATHNAME parameter). If the PATHNAME parameter is specified, only the operation count for that path is returned.

DVI$_OWNUIC

Returns the user identification code (UIC) of the owner of the device as a standard 4-byte UIC.

DVI$_PATH_AVAILABLE

On Alpha and Integrity server systems, returns an unsigned longword, interpreted as Boolean. A value of 1 indicates the specified path is available.

This item code is usually used with the PATHNAME parameter. If the PATHNAME parameter is omitted, information about the current path of the multipath device is returned.

DVI$_PATH_NOT_RESPONDING

On Alpha and Integrity server systems, returns an unsigned longword, interpreted as Boolean. A value of 1 indicates the specified path is marked as not responding.

This item code is usually used with the PATHNAME parameter. If the PATHNAME parameter is omitted, information about the current path of the multipath device is returned.

DVI$_PATH_POLL_ENABLED

On Alpha and Integrity server systems, returns an unsigned longword, interpreted as Boolean. A value of 1 indicates that the specified path is enabled for multipath polling.

This item code is usually used with the PATHNAME parameter. If the PATHNAME parameter is omitted, information about the current path of the multipath device is returned.

DVI$_PATH_SWITCH_FROM_TIME

On Alpha and Integrity server systems, returns the time from which this path was switched, either manually or automatically. Because the returned time is in the standard 64-bit absolute time format, specify 8 (bytes) for the buffer length field in the item descriptor.

This item code is usually used with the PATHNAME parameter. If the PATHNAME parameter is omitted, information about the current path of the multipath device is returned.

DVI$_PATH_SWITCH_TO_TIME

On Alpha and Integrity server systems, returns the time to which this path was switched, either manually or automatically. Because the returned time is in the standard 64-bit absolute time format, specify 8 (bytes) for the buffer length field in the item descriptor.

This item code is usually used with the PATHNAME parameter. If the PATHNAME parameter is omitted, information about the current path of the multipath device is returned.

DVI$_PATH_USER_DISABLED

On Alpha and Integrity server systems, returns an unsigned longword, interpreted as Boolean. A value of 1 indicates the specified path has been disabled using the $SET DEVICE /PATH /NOENABLE command.

This item code is usually used with the PATHNAME parameter. If the PATHNAME parameter is omitted, information about the current path of the multipath device is returned.

DVI$_PID

Returns the process identification (PID) of the owner of the device as an unsigned integer longword.

DVI$_PREFERRED_CPU

Do not use this item code. The service continues to return the correct data for systems with up to 64 CPUs but fails for systems with more than 64 CPUs. Use item code DVI$_PREFERRED_CPU_BITMAP instead.

DVI$_PREFERRED_CPU_BITMAP

On Alpha and Integrity server systems, the return argument is a bitmap with a bit indicating the preferred CPU. A return argument containing a bit mask of zero indicates that no preferred CPU exists, either because Fast Path is disabled or the device is not a Fast Path capable device. The return argument serves as a CPU bitmap input argument to the $PROCESS_AFFINITY system service. The argument can be used to assign an application process to the optimal preferred CPU.

The size of the returned bitmap is determined by the number of supported CPUs on the system. You can compute the number of bytes needed for the bitmap as follows: Use the $GETSYI system service with an item code of SYI$_MAX_CPUS to find the minimum number of bits needed, round this number up to a multiple of 64, and divide the result by 8.

DVI$_PROT_SUBSYSTEM_ENABLED

On Alpha and Integrity server systems, returns an unsigned longword, which is interpreted as Boolean. A value of 1 indicates the volume is mounted with protected subsystems enabled.

DVI$_RECSIZ

Returns the blocked record size as an unsigned integer longword.

DVI$_REFCNT

Returns the number of channels assigned to the device as an unsigned integer longword.

DVI$_REMOTE_DEVICE

Returns a longword, which is interpreted as Boolean. A value of 1 indicates that the device is a remote device; a value of 0 indicates that it is not a remote device. A remote device is a device that is not directly connected to the local node, but instead is visible through the OpenVMS Cluster system.

DVI$_ROOTDEVNAM

Returns the device name of the root volume in the volume set as a string. This item code is applicable only to disks.

DVI$_SCSI_DEVICE_FIRMWARE_REV

On Alpha and Integrity server systems, returns the firmware revision of a SCSI disk or SCSI tape as a four-character string. This item code is valid only for SCSI disks and SCSI tapes; a null string is returned for any other device.

DVI$_SECTORS

Returns the number of sectors per track as an unsigned integer longword. This item code is applicable only to disks.

DVI$_SERIALNUM

Returns the serial number of the volume as an unsigned integer longword. This item code is applicable only to disks.

DVI$_SERVED_DEVICE

Returns a longword, which is interpreted as Boolean. A value of 1 indicates that the device is a served device; a value of 0 indicates that it is not a served device. A served device is one whose local node makes it available to other nodes in the OpenVMS Cluster system.

DVI$_SHDW_CATCHUP_COPYING

Returns a longword, which is interpreted as Boolean. The value 1 indicates that the device is the target of a full copy operation.

DVI$_SHDW_COPIER_NODE

On Alpha and Integrity server systems, returns the name of the node that is actively performing either the copy or the merge operation, as a string.

DVI$_SHDW_DEVICE_COUNT

On Alpha and Integrity server systems, returns the total number of devices in the virtual unit, including devices being added as copy targets, as a longword.

DVI$_SHDW_GENERATION

On Alpha and Integrity server systems, returns the current, internal revision number of the virtual unit, as a quadword.

DVI$_SHDW_MASTER

Returns a longword, which is interpreted as Boolean. The value 1 indicates that the device is a virtual unit named DSAnnnn:.

DVI$_SHDW_MASTER_MBR

On Alpha and Integrity server systems, returns the name of the master member device (as a string) that is used for merge and copy repair operations and for shadow set recovery operations.

DVI$_SHDW_MASTER_NAME

When the specified device is a shadow set member, $GETDVI returns the device name of the virtual unit (DSAnnnn:) that represents the shadow set of which the specified device is a member. $GETDVI returns a null string if the specified device is not a member or is itself a virtual unit (DSAnnnn:).

Note

Shadow set members must have a nonzero allocation class to operate in an OpenVMS Cluster system. For more information, see *VSI Volume Shadowing for OpenVMS*.

DVI$_SHDW_MBR_COPY_DONE

On Alpha and Integrity server systems, returns the percentage of the copy operation that is complete on the current member unit, as a longword.

DVI$_SHDW_MBR_COUNT

On Alpha and Integrity server systems, returns the number of full source members in the virtual unit, as a longword. Devices added as copy targets are not full source members.

DVI$_SHDW_MBR_MERGE_DONE

On Alpha and Integrity server systems, returns the percentage of the merge operation that has been completed on the member, as a longword.

DVI$_SHDW_MBR_READ_COST

On Alpha and Integrity server systems, returns the current value set for the member unit, as a longword. This value can be modified to use a customer-specified value.

DVI$_SHDW_MEMBER

Returns a longword, which is interpreted as Boolean. The value 1 indicates that the device is a shadow set member.

DVI$_SHDW_MERGE_COPYING

On Alpha and Integrity server systems, returns a longword, which is interpreted as Boolean. The value 1 indicates that the device is a merge member of the shadow set.

DVI$_SHDW_MINIMERGE_ENABLE

On Alpha and Integrity server systems, returns a longword, which is interpreted as Boolean. The value 1 indicates that, if a system in the cluster that has this device mounted crashes, the virtual unit will undergo a minimerge and not a full merge.

DVI$_SHDW_NEXT_MBR_NAME

Returns the device name of the next member in the shadow set. If you specify a virtual unit with the *chan* or *devnam* argument, DVI$_SHDW_NEXT_MBR_NAME returns the device name of a member of a shadow set. If you specify the name of a shadow set member unit with the *chan* or *devnam* argument, DVI$_SHDW_NEXT_MBR_NAME returns the name of the next member unit or a null string if there are no more members.

To determine all the members of a shadow set, first specify the virtual unit (DSAnnnn:) to $GETDVI. Then, on subsequent calls, specify the member name returned by the previous $GETDVI call until it returns a null member name.

When the shadow set members have a nonzero allocation class, the device name returned by $GETDVI contains the allocation class; the name has the form $allocation-class$device. For example, if a shadow set has an allocation class of 255 and the device name is DUA42, $GETDVI returns the string 255DUA42.

Note

Shadow set members must have a nonzero allocation class to operate in an OpenVMS Cluster system. For more information, see *VSI Volume Shadowing for OpenVMS*.

DVI$_SHDW_READ_SOURCE

On Alpha and Integrity server systems, returns the name of the member device that is used for reads, at this point in time, as a string. DVI$_SHDW_READ_SOURCE uses the unit that has the lowest value of the sum of its queue length and read cost for reads. This is a dynamic value.

DVI$_SHDW_TIMEOUT

On Alpha and Integrity server systems, returns the customer-specified timeout value set for the device, as a longword.

If you do not set a value using the SETSHOWSHADOW utility, the SYSGEN parameter SHADOW_MBR_TWO is used for member units and MVTIMEOUT is used for virtual units.

DVI$_SPECIAL_FILES

If the item code DVI$_SPECIAL_FILES is set to 1, special files are enabled. When it is set to 0, special files are disabled.

DVI$_STS

Returns the device unit status as a 4-byte bit vector. Each bit in the vector, when set, corresponds to a symbolic name that is defined by the $UCBDEF macro. The following table describes each name:

Symbol	Description
UCB$V_ALTBSY	Unit is busy via alternate startio path.
UCB$V_BSY	Unit is busy.
UCB$V_CANCEL	I/O on unit is canceled.
UCB$V_CLUTRAN	OpenVMS Cluster state transition in progress.
UCB$V_DEADMO	Deallocate at dismount.
UCB$V_DELETEUCB	Delete this UCB when reference count equals 0.
UCB$V_DISMOUNT	Dismount in progress.
UCB$V_ERLOGIP	Error log is in progress on unit.
UCB$V_EXFUNC_SUPP	Unit supports the EXFUNC bit.
UCB$V_FAST_PATH	Unit supports FAST PATH Affinity.
UCB$V_FP_HWINT	Unit supports FAST PATH hardware interrupt CPU Affinity.
UCB$V_INT	Interrupt is expected.
UCB$V_INTTYPE	Receiver interrupt.
UCB$V_IOPOST_LOCAL	Unit supports I/O post processing on the current CPU.
UCB$V_LCL_VALID	Volume is valid on the local node.
UCB$V_MNTVERIP	Mount verification is in progress.
UCB$V_MOUNTING	Device is being mounted.
UCB$V_MNTVERPND	Mount verification is pending on busy device.
UCB$V_NO_ASSIGN	Unit cannot have channels assigned to it.
UCB$V_ONLINE	Unit is on line.
UCB$V_PATHVERIP	Path verification is in progress for this device.
UCB$V_POWER	Power failed while unit busy.
UCB$V_SNAPSHOT	Restart validation is in progress.
UCB$V_SUPMVMSG	If set, suppress success type mount version messages.
UCB$V_SVPN_END	Last byte used from page mapped by system virtual page number (SVPN).
UCB$V_TEMPLATE	Template UCB from which other UCBs for this device type are made.

Symbol	Description
UCB$V_TIM	Timeout is enabled.
UCB$V_TIMOUT	Unit timed out.
UCB$V_UNLOAD	Unload volume at dismount.
UCB$V_VALID	Volume is software valid.
UCB$V_WRONGVOL	Wrong volume detected during mount verification.
UCB$V_WRTLOCKMV	Write-locked mount verification in progress.

DVI$_TOTAL_PATH_COUNT

On Alpha and Integrity server systems, the number of paths for a multipath-capable device returns as an unsigned longword.

DVI$_TRACKS

Returns the number of tracks per cylinder as an unsigned integer longword. This item code is applicable only to disks.

DVI$_TRANSCNT

Returns the transaction count for the volume as an unsigned integer longword.

DVI$_QLEN

On Alpha and Integrity server systems, returns the queue length for the device as an unsigned integer longword. Note that this value is the number of I/O requests already in the driver and not the depth of the I/O pending queue.

DVI$_TT_ACCPORNAM

Returns the name of the remote access port associated with a channel number or with a physical or virtual terminal device number. If you specify a device that is not a remote terminal or a remote type that does not support this feature, $GETDVI returns a null string. The $GETDVI service returns the access port name as a string. VSI recommends a buffer size of 64 bytes to return the name of the remote access port.

The $GETDVI service returns the name in the format of the remote system. If the remote system is a LAT terminal server, $GETDVI returns the name as *server_name/port_name.* The names are separated by the slash (/) character. If the remote system is an X.29 terminal, the name is returned as *network.remote_DTE.*For devices using TCP/IP, the name is returned in the format *Host: 192.168.1.100 Port: 1*.

When writing applications, use the string returned by DVI$_ACCPORNAM (instead of the physical device name) to identify remote terminals.

DVI$_TT_CHARSET

Returns, as a 4-byte bit vector, the character sets supported by the terminal. Each bit in the vector, when set, corresponds to the name of a coded character set. The $TTCDEF macro defines the following coded character sets:

Symbol	Description
TTC$V_HANGUL	DEC Korean

Symbol	Description
TTC$V_HANYU	DEC Hanyu
TTC$V_HANZI	DEC Hanzi
TTC$V_KANA	DEC Kana
TTC$V_KANJI	DEC Kanji
TTC$V_THAI	DEC Thai

DVI$_TT_CS_HANGUL

Returns a longword, which is interpreted as Boolean. A value of 1 indicates that the device supports the DEC Korean coded character set; a value of 0 indicates that the device does not support the DEC Korean coded character set.

DVI$_TT_CS_HANYU

Returns a longword, which is interpreted as Boolean. A value of 1 indicates that the device supports the DEC Hanyu coded character set; a value of 0 indicates that the device does not support the DEC Hanyu coded character set.

DVI$_TT_CS_HANZI

Returns a longword, which is interpreted as Boolean. A value of 1 indicates that the device supports the DEC Hanzi coded character set; a value of 0 indicates that the device does not support the DEC Hanzi coded character set.

DVI$_TT_CS_KANA

Returns a longword, which is interpreted as Boolean. A value of 1 indicates that the device supports the DEC Kana coded character set; a value of 0 indicates that the device does not support the DEC Kana coded character set.

DVI$_TT_CS_KANJI

Returns a longword, which is interpreted as Boolean. A value of 1 indicates that the device supports the DEC Kanji coded character set; a value of 0 indicates that the device does not support the DEC Kanji coded character set.

DVI$_TT_CS_THAI

Returns a longword, which is interpreted as Boolean. A value of 1 indicates that the device supports the DEC Thai coded character set; a value of 0 indicates that the device does not support the DEC Thai coded character set.

DVI$_TT_PHYDEVNAM

Returns a string containing the physical device name of a terminal. If the caller specifies a disconnected virtual terminal or a device that is not a terminal, $GETDVI returns a null string. $GETDVI returns the physical device name as a string.

DVI$_UNIT

Returns the unit number as an unsigned longword.

DVI$_VOLCHAR

On Alpha and Integrity server systems, returns a 128-bit string (16 bytes) that represents the volume characteristics or capabilities of the mounted device. If a bit is set, the volume is capable of performing the function.

DVI$_VOLCOUNT

Returns the number of volumes in the volume set as an unsigned longword. This item code is applicable only to disks.

DVI$_VOLNAM

Returns the volume name as a string.

DVI$_VOLNUMBER

Returns the volume number of this volume in the volume set as an unsigned integer longword. This item code is applicable only to disks.

DVI$_VOLSETMEM

Returns a longword, which is interpreted as Boolean. A value of 1 indicates that the device is part of a volume set; a value of 0 indicates that it is not. This item code is applicable only to disks.

DVI$_VOLSIZE

On Alpha and Integrity server systems, returns the current logical volume size of the volume as an unsigned integer longword.

DVI$V_VOL_SPECIAL_FILE

If the item code DVI$V_VOL_SPECIAL_FILE is set to 1, special files are enabled. When the item code is set to 0, special files are disabled.

The DVI$_VOLCHAR item code supports DVI$V_VOL_SPECIAL_FILE when the latter is set to 1.

DVI$_VOLUME_EXTEND_QUANTITY

On Alpha and Integrity server systems, returns the number of blocks to be used as the default extension size for all files on the volume as an unsigned longword.

DVI$_VOLUME_MOUNT_GROUP

On Alpha and Integrity server systems, returns an unsigned longword, interpreted as Boolean. A value of 1 indicates the volume is mounted /GROUP.

DVI$_VOLUME_MOUNT_SYS

On Alpha and Integrity server systems, returns an unsigned longword, interpreted as Boolean. A value of 1 indicates the volume is mounted /SYSTEM.

DVI$_VOLUME_PENDING_WRITE_ERR

On Alpha and Integrity server systems, returns the count of pending write errors on the volume as an unsigned longword.

DVI$_VOLUME_RETAIN_MAX

On Alpha and Integrity server systems, returns the maximum retention time for the volume as specified with the DCL command SET VOLUME/RETENTION. Because the returned time is in the standard 64-bit absolute time format, specify 8 bytes for the buffer length field in the item descriptor.

DVI$_VOLUME_RETAIN_MIN

On Alpha and Integrity server systems, returns the minimum retention time for the volume, as specified with the DCL command bit SET VOLUME/RETENTION. Because the returned time is in the standard 64-bit absolute time format, specify 8 bytes for the buffer length field in the item descriptor.

DVI$_VOLUME_SPOOLED_DEV_CNT

On Alpha and Integrity server systems, returns the number of devices spooled to the volume as an unsigned longword.

DVI$_VOLUME_WINDOW

On Alpha and Integrity server systems, returns the volume default window size as an unsigned longword.

DVI$_VPROT

Returns the volume protection mask as a standard 4-byte protection mask.

DVI$_TT_xxxx

DVI$_TT_xxxx is the format for a series of item codes that return information about terminals. This information consists of terminal characteristics. The `xxxx` portion of the item code name specifies a single terminal characteristic.

Each of these item codes requires that the buffer specify a longword into which $GETDVI will write a 0 or 1: 0 if the terminal does not have the specified characteristic, and 1 if the terminal does have it. The one exception is the DVI$_TT_PAGE item code, which when specified causes $GETDVI to return a decimal longword value that is the page size of the terminal.

You can also obtain this terminal-specific information by using the DVI$_DEVDEPEND and DVI$_DEVDEPEND2 item codes. Each of these two item codes specifies a longword bit vector wherein each bit corresponds to a terminal characteristic; $GETDVI sets the corresponding bit for each characteristic possessed by the terminal.

Following is a list of the item codes that return information about terminal characteristics. For information about these characteristics, refer to the description of the F$GETDVI lexical function in the *VSI OpenVMS DCL Dictionary*.

DVI$_TT_NOECHO	DVI$_TT_NOTYPEAHD
DVI$_TT_HOSTSYNC	DVI$_TT_TTSYNC
DVI$_TT_ESCAPE	DVI$_TT_LOWER
DVI$_TT_MECHTAB	DVI$_TT_WRAP
DVI$_TT_LFFILL	DVI$_TT_SCOPE
DVI$_TT_CRFILL	DVI$_TT_SETSPEED
DVI$_TT_EIGHTBIT	DVI$_TT_MBXDSABL
DVI$_TT_READSYNC	DVI$_TT_MECHFORM
DVI$_TT_NOBRDCST	DVI$_TT_HALFDUP
DVI$_TT_MODEM	DVI$_TT_OPER

DVI$_TT_LOCALECHO	DVI$_TT_AUTOBAUD
DVI$_TT_PAGE	DVI$_TT_HANGUP
DVI$_TT_MODHANGUP	DVI$_TT_BRDCSTMBX
DVI$_TT_DMA	DVI$_TT_ALTYPEAHD
DVI$_TT_ANSICRT	DVI$_TT_REGIS
DVI$_TT_AVO	DVI$_TT_EDIT
DVI$_TT_BLOCK	DVI$_TT_DECCRT
DVI$_TT_EDITING	DVI$_TT_INSERT
DVI$_TT_DIALUP	DVI$_TT_SECURE
DVI$_TT_FALLBACK	DVI$_TT_DISCONNECT
DVI$_TT_PASTHRU	DVI$_TT_SIXEL
DVI$_TT_PRINTER	DVI$_TT_APP_KEYPAD
DVI$_TT_DRCS	DVI$_TT_SYSPWD
DVI$_TT_DECCRT2	
DVI$_TT_DECCRT3	
DVI$_TT_DECCRT4	

DVI$_WRITETHRU_CACHE_ENABLED

On Alpha and Integrity server systems, returns an unsigned longword, which is interpreted as Boolean. A value of 1 indicates the volume is mounted with write-through caching enabled.

DVI$_WWID

On Alpha and Integrity server systems, returns the World Wide Identifier (WWID) of Fibre Channel Disk and Tape devices as a string. The maximum length of this string might change with new devices; therefore, VSI recommends that a 380-byte buffer be passed to this function.

DVI$_XFC_DEPOSING

On Alpha and Integrity server systems, returns a Boolean value indicating whether the XFC volume depose operation is in progress or not. A value of 1 indicates that the XFC volume depose operation is in progress; a value of 0 indicates that the XFC volume depose operation is not in progress.

DVI$_yyyy

DVI$_yyyy is the format for a series of item codes that return device-independent characteristics of a device. There is an item code for each device characteristic returned in the longword bit vector specified by the DVI$_DEVCHAR item code.

In the description of the DVI$_DEVCHAR item code is a list of symbol names in which each symbol represents a device characteristic. To construct the $GETDVI item code for each device characteristic, substitute for `yyyy` that portion of the symbol name that follows the underscore character. For example, the DVI$_REC item code returns the same information as the DEV$V_REC bit in the DVI$_DEVCHAR longword bit vector.

The buffer for each of these item codes must specify a longword value, which is interpreted as Boolean. The $GETDVI service writes the value 1 into the longword if the device has the specified characteristic and the value 0 if it does not.

New Item Code DVI$_ADAPTER_IDENT

On Alpha and I64 systems, this returns (as a string) the description of an adapter as defined in either SYS$SYSTEM:SYS$CONFIG.DAT or SYS$SYSTEM:SYS$USER_CONFIG.DAT. Note that this service does not read either of those files; those files are read into memory in response to the SYSMAN IO REBUILD command, which typically happens while a system is booting.

VSI recommends a buffer size of 255 bytes to hold the identification string.

New item code DVI$_MOUNTCNT_CLUSTER

On Alpha and I64 systems, this item code returns (as a longword) the number of systems in a cluster that have a device mounted. Note that this new item code is not a direct replacement for the existing item code DVI$_MOUNTCNT. That item code returns the number of mounters for any device on the local system. The /SHARE qualifier to the MOUNT command can allow for more than one mounter.

Description

The Get Device/Volume Information service returns primary and secondary device characteristics information about an I/O device. You can use the *chan* argument only if (1) the channel has already been assigned, and (2) the caller's access mode is equal to or more privileged than the access mode from which the original channel assignment was made.

The caller of $GETDVI does not need to have a channel assigned to the device about which information is desired.

The $GETDVI service returns information about both primary device characteristics and secondary device characteristics. By default, $GETDVI returns information about the primary device characteristics only.

To obtain information about secondary device characteristics, you must perform a logical OR operation on the item code specifying the information desired with the code DVI$C_SECONDARY.

You can obtain information about primary and secondary devices in a single call to $GETDVI.

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

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

Unless otherwise stated in the description of the item code, $GETDVI returns information about the local node only.

Required Access or Privileges

None

Required Quota

Sufficient AST quota.

Related Services

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

Condition Values Returned

SS$_NORMAL

The service completed successfully.

SS$_ACCVIO

The device name string descriptor, device name string, or *itmlst* argument cannot be read; or the buffer or return length longword cannot be written by the caller.

SS$_BADPARAM

The item list contains an invalid item code, or the buffer address field in an item descriptor specifies less than four bytes for the return length information.

SS$_EXASTLM

The process has exceeded its AST limit quota.

SS$_IVCHAN

You specified an invalid channel number, that is, a channel number larger than the number of channels.

SS$_IVDEVNAM

The device name string contains invalid characters, or neither the *devnam* nor *chan* argument was specified.

SS$_IVLOGNAM

The device name string has a length of 0 or has more than 63 characters.

SS$_NONLOCAL

The device is on a remote system.

SS$_NOPRIV

The specified channel is not assigned or was assigned from a more privileged access mode.

SS$_NOSUCHDEV

The specified device does not exist on the host system.

SS$_NOSUCHPATH

On Alpha and Integrity server systems, the specified path does not exist for the device, even though the device itself does exist.

SS$_UNSUPPORTED

One or more of the item codes are not supported on the device specified.

Condition Values Returned in the I/O Status Block

The condition values returned are the same as those returned in R0.

$GETDVIW

Get Device/Volume Information and Wait — Returns information about an I/O device; this information consists of primary and secondary device characteristics.

Format

```
SYS$GETDVIW
   [efn] ,[chan] ,[devnam] ,itmlst [,iosb] [,astadr] [,astprm]
   [,nullarg,][pathname]
```

C Prototype

```
int sys$getdviw
        unsigned int efn,
        unsigned short int chan,
        void *devnam,
        void *itmlst,
        struct _iosb *iosb,
        void (*astadr)(__unknown_params),
        int astprm,
        struct _generic_64 *nullarg,
        __optional_params);
```

Description

The $GETDVIW service completes synchronously; that is, it returns to the caller with the requested information. VSI recommends that you use an IOSB with this service. An IOSB prevents the service from completing prematurely. In addition, the IOSB contains additional status information.

For asynchronous completion, use the Get Device/Volume Information ($GETDVI) service; $GETDVI returns to the caller after queuing the information request, without waiting for the information to be returned. In all other respects, $GETDVIW is identical to $GETDVI. For all other information about the $GETDVIW service, see the description of $GETDVI.

Note

All **pathname**-related information pertains only to Alpha and Integrity server systems.

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

$GETENV (Alpha Only)

Get Environment — Returns the value(s) of the specified console environment variable(s).

Format

```
SYS$GETENV itmlst
```

C Prototype

```
int sys$getenv (void *itmlst);
```

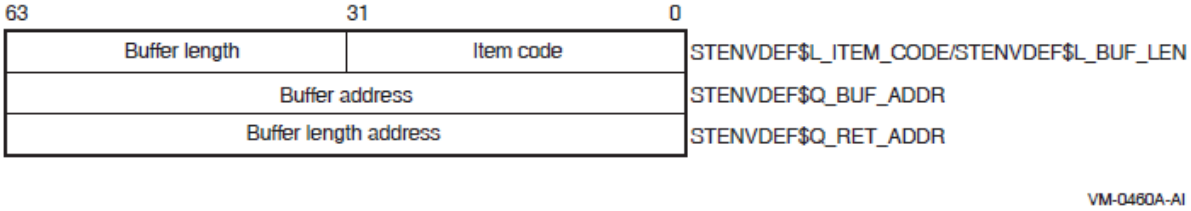
Arguments

itmlst

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

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 service takes one argument as input, an item list. This item list has the following format for a single item descriptor:



The following table defines the item descriptor fields:

Descriptor Field	Definition
Item code	A longword indicating which environment variable you want to retrieve. These codes are defined in $STENVDEF.
Buffer length	A longword specifying the length of the buffer in which GETENV is to write the environment variable's value.
Buffer address	A quadword indicating the address of the buffer in which GETENV is to write the environment variable's value.
Return length address	A quadword indicating the return address in which to put the length of the value that GETENV retrieved.

Description

This system service will return the value(s) of the specified console environment variable(s).

Required Access or Privileges
None

Required Quota
None

Related Services
None

Condition Values Returned

SS$_NORMAL

Operation was successful; requested data was returned to caller.

SS$_ACCVIO

This status is returned if the caller does not have write access to the two input buffers or if the probe for read access to the item list fails.

SS$_BADPARAM

This status is returned if an empty item list is specified, or if the console callback to read the environment variable fails for any reason.

$GET_GALAXY_LOCK_INFO (Alpha Only)

Get OpenVMS Galaxy Lock Information — Returns "interesting" fields from the specified lock.
Note that this system service is supported only in an OpenVMS Alpha Galaxy environment. For more information about programming with OpenVMS Galaxy system services, refer to the *VSI OpenVMS Alpha Partitioning and Galaxy Guide*.

Format

```
SYS$GET_GALAXY_LOCK_INFO
   handle ,name ,timeout ,size ,ipl ,rank ,flags [,name_length]
```

C Prototype

```
int sys$get_galaxy_lock_info
   (unsigned __int64 lock_handle, char *name, unsigned int *timeout,
    unsigned int *size, unsigned int *ipl, unsigned int *rank,
    unsigned short int *flags unsigned short int *name_length);
```

Arguments

handle

OpenVMS usage:	handle for the galaxy lock
type:	quadword (unsigned)
access:	read

mechanism:	input by value

The 64-bit lock handle that identifies the lock on which to return information. This value is returned by SYS$CREATE_GALAXY_LOCK.

name

OpenVMS usage:	address
type:	zero-terminated string
access:	write
mechanism:	output by reference

Pointer to a buffer. This buffer must be large enough to receive the name of the lock. Locks names are zero-terminated strings with a maximum size of 16 bytes.

timeout

OpenVMS usage:	address
type:	longword (unsigned)
access:	write
mechanism:	output by reference

Pointer to a longword. The value returned is the timeout value of the lock.

size

OpenVMS usage:	address
type:	longword (unsigned)
access:	write
mechanism:	output by reference

Pointer to a longword. The value returned is the size of the lock in bytes.

ipl

OpenVMS usage:	address
type:	longword (unsigned)
access:	write
mechanism:	output by reference

Pointer to a longword. The value returned is the IPL of the lock.

rank

OpenVMS usage:	address
type:	longword (unsigned)
access:	write
mechanism:	output by reference

Pointer to a longword. The value returned is the rank of the lock.

flags

OpenVMS usage:	address
type:	word (unsigned)
access:	write
mechanism:	output by reference

Pointer to a word. The value returned is the word mask of lock flags.

name_length

OpenVMS usage:	address
type:	word (unsigned)
access:	write
mechanism:	output by reference

Length of the string returned in the *name* argument.

Description

This service returns all "interesting" fields from the specified lock. See the $CREATE_GALAXY_LOCK service for detailed information regarding these values.

Required Access or Privileges

Read access to lock.

Required Quota

None

Related Services

$ACQUIRE_GALAXY_LOCK, $CREATE_GALAXY_LOCK, $CREATE_GALAXY_LOCK_TABLE, $DELETE_GALAXY_LOCK, $DELETE_GALAXY_LOCK_TABLE, $GET_GALAXY_LOCK_SIZE, $RELEASE_GALAXY_LOCK

Condition Values Returned

SS$_NORMAL

Normal completion.

SS$_IVLOCKID

Invalid lock id.

SS$_IVLOCKTBL

Invalid lock table.

$GET_GALAXY_LOCK_SIZE (Alpha Only)

— Returns the minimum and maximum size of an OpenVMS Galaxy lock. Note that this system service is supported only in an OpenVMS Alpha Galaxy environment. For more information about programming with OpenVMS Galaxy system services, refer to the *VSI OpenVMS Alpha Partitioning and Galaxy Guide*.

Format

```
SYS$GET_GALAXY_LOCK_SIZE min_size ,max_size
```

C Prototype

```
int sys$get_galaxy_lock_size
   (unsigned int *min_size, unsigned int *max_size);
```

Arguments

min_size

OpenVMS usage:	address
type:	longword (unsigned)
access:	write
mechanism:	output by reference

Pointer to a longword. The value returned is minimum legal size of a galaxy lock structure.

max_size

OpenVMS usage:	address
type:	longword (unsigned)
access:	write
mechanism:	output by reference

Pointer to a longword. The value returned is maximum legal size of a galaxy lock structure.

Description

This service returns the minimum and maximum size of an OpenVMS Galaxy lock. If a lock is created with the maximum size, the locking services will record acquire and release information in the lock.

The lock sizes can be used to determine the value of the *section_size* parameter to the $CREATE_GALAXY_LOCK_TABLE service.

Required Access or Privileges

Read access to lock.

Required Quota

None

Related Services

$ACQUIRE_GALAXY_LOCK, $CREATE_GALAXY_LOCK, $CREATE_GALAXY_LOCK_TABLE, $DELETE_GALAXY_LOCK, $DELETE_GALAXY_LOCK_TABLE, $GET_GALAXY_LOCK_INFO, $RELEASE_GALAXY_LOCK

Condition Values Returned

SS$_NORMAL

Normal completion.

$GETJPI

Get Job/Process Information — Returns information about one or more processes on the system or across the OpenVMS Cluster system. The $GETJPI service completes asynchronously. For synchronous completion, use the Get Job/Process Information and Wait ($GETJPIW) service. On Alpha and Integrity server systems, this service accepts 64-bit addresses.

Format

```
SYS$GETJPI
   [efn] ,[pidadr] ,[prcnam] ,itmlst ,[iosb] ,[astadr] ,[astprm]
```

C Prototype

```
int sys$getjpi
   (unsigned int efn, unsigned int *pidadr, void *prcnam, void *itmlst,
    struct _iosb *iosb, void (*astadr)(__unknown_params),
    unsigned __int64 astprm);
```

Arguments

efn

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

Number of the event flag to be set when $GETJPI returns the requested information. The *efn* argument is a quadword containing this number; however, $GETJPI uses only the low-order byte.

Upon request initiation, $GETJPI clears the specified event flag (or event flag 0 if *efn* was not specified). Then, when $GETJPI returns the requested information, it sets the specified event flag (or event flag 0).

VSI strongly recommends the use of the EFN$C_ENF “no event flag” value as the event flag if you are not using an event flag to externally synchronize with the completion of this system service call. The $EFNDEF macro defines EFN$C_ENF. For more information, see the *VSI OpenVMS Programming Concepts Manual*.

pidadr

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

Process identification (PID) of the process about which $GETJPI is to return information. The *pidadr* argument is the 32- or 64-bit 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 cluster.

If you give *pidadr* the value –1, $GETJPI assumes a wildcard operation and returns the requested information for each process on the system that it has the privilege to access, one process per call. To perform a wildcard operation, you must call $GETJPI in a loop, testing for the condition value SS$_NOMOREPROC after each call and exiting from the loop when SS$_NOMOREPROC is returned.

If you use $GETJPI with $PROCESS_SCAN, you can perform wildcard searches across the cluster. In addition, with $PROCESS_SCAN you can search for specific processes based on many different selection criteria.

You cannot abbreviate a PID. All significant digits of a PID must be specified; only leading zeros can be omitted.

prcnam

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

Name of the process about which $GETJPI is to return information. The *prcnam* argument is the 32- or 64-bit address of a character string descriptor pointing to this name string.

A process running on the local node can be identified with a 1- to 15-character string. To identify a process on a cluster, you must 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 local process name can look like a remote process name; therefore, if you specify ATHENS::SMITH, the system checks for a process named ATHENS::SMITH on the local node before checking node ATHENS for a process named SMITH.

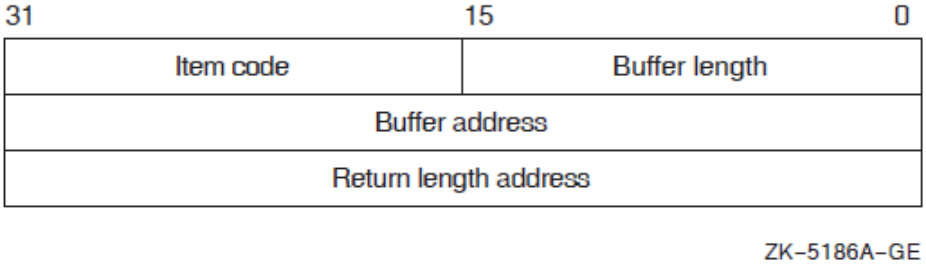
You can use the *prcnam* argument only if the process identified by *prcnam* has the same UIC group number as the calling process. If the process has a different group number, $GETJPI returns no information. To obtain information about processes in other groups, you must use the *pidadr* argument.

itmlst

OpenVMS usage:	32-bit item_list_3 or 64-bit item_list_64b
type:	longword (unsigned) for 32-bit; quadword (unsigned) for 64-bit
access:	read only
mechanism:	by 32- or 64-bit reference

Item list specifying which information about the process or processes is to be returned. The *itmlst* argument is the 32- or 64-bit address of a list of item descriptors, each of which describes an item of information. An item list in 32-bit format is terminated by a longword of 0; an item list in 64-bit format is terminated by a quadword of 0. All items in an item list must be of the same format—either 32-bit or 64-bit.

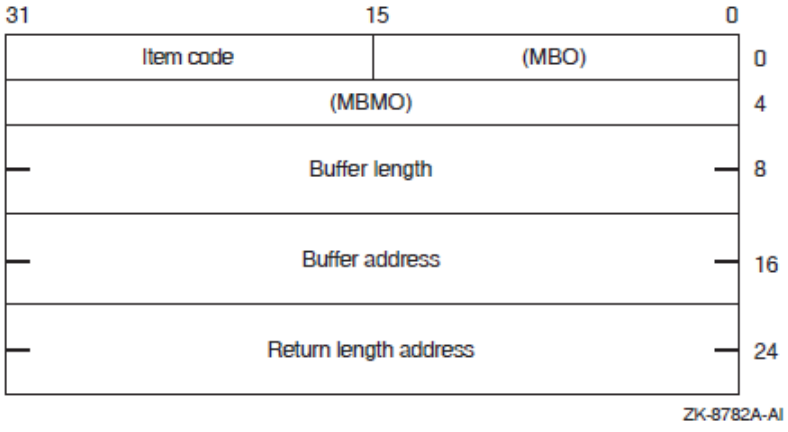
The following diagram depicts the 32-bit format of a single item descriptor:



The following table defines the item descriptor fields for 32-bit item list entries:

Descriptor Field	Definition
Buffer length	A word containing a user-supplied integer specifying the length (in bytes) of the buffer in which $GETJPI is to write the information. The length of the buffer needed depends on the item code specified in the item code field of the item descriptor. If the value of buffer length is too small, $GETJPI truncates the data.
Item code	A word containing a user-supplied symbolic code specifying the item of information that $GETJPI is to return. The $JPIDEF macro defines these codes. Each item code is described in the Item Codes section.
Buffer address	A longword containing the user-supplied 32-bit address of the buffer in which $GETJPI is to write the information.
Return length address	A longword containing the user-supplied 32-bit address of a word in which $GETJPI writes the length (in bytes) of the information it actually returned.

The following diagram depicts the 64-bit format of a single item descriptor:



The following table defines the item descriptor fields for 64-bit item list entries:

Descriptor Field	Definition
MBO	The field must contain a 1. The MBO and MBMO fields are used to distinguish 32-bit and 64-bit item list entries.

Descriptor Field	Definition
Item code	A word containing a symbolic code that describes the information in the buffer or the information to be returned to the buffer, pointed to by the buffer address field. The item codes are listed in the Item Codes section.
MBMO	The field must contain a –1. The MBMO and MBO fields are used to distinguish 32-bit and 64-bit item list entries.
Buffer length	A quadword containing a user-supplied integer specifying the length (in bytes) of the buffer in which $GETJPI is to write the information. The length of the buffer needed depends on the item code specified in the item code field of the item descriptor. If the value of buffer length is too small, $GETJPI truncates the data.
Buffer address	A quadword containing the user-supplied 64-bit address of the buffer in which $GETJPI is to write the information.
Return length address	A quadword containing the user-supplied 64-bit address of a word in which $GETJPI writes the length (in bytes) of the information it actually returned.

iosb

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

I/O status block that is to receive the final completion status. The *iosb* argument is the 32- or 64-bit address of the quadword I/O status block.

When you specify the *iosb* argument, $GETJPI 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, VSI strongly recommends that you specify it, for the following reasons:

- If you are using an event flag to signal the completion of the service, you can test the I/O status block for a condition value to be sure that the event flag was not set by an event other than service completion.
- If you are using the $SYNCH service to synchronize completion of the service, the I/O status block is a required argument for $SYNCH.
- The condition value returned in R0 and the condition value returned in the I/O status block provide information about different aspects of the call to the $GETJPI service. The condition value returned in R0 gives you information about the success or failure of the service call itself; the condition value returned in the I/O status block gives you information about the success or failure of the service operation. Therefore, to accurately assess the success or failure of the call to $GETJPI, you must check the condition values returned in both R0 and the I/O status block.

astadr

OpenVMS usage:	ast_procedure

type:	procedure value
access:	call without stack unwinding
mechanism:	by 32- or 64-bit reference

AST service routine to be executed when $GETJPI completes. The *astadr* argument is the 32- or 64-bit address of this routine.

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

JPI$_ACCOUNT

Returns the account name of the process, which is an 8-byte string, filled with trailing blanks if necessary.

JPI$_APTCNT

Returns, in pagelets, the active page table count of the process, which is a longword integer value.

JPI$_ASTACT

Returns the names of the access modes having active ASTs. This information is returned in a longword bit vector. When bit 0 is set, an active kernel mode AST exists; bit 1, an executive mode AST; bit 2, a supervisor mode AST; and bit 3, a user mode AST.

JPI$_ASTCNT

Returns a count of the remaining AST quota, which is a longword integer value.

JPI$_ASTEN

Returns a longword bit vector that indicates for each access mode whether ASTs are enabled for that mode. When bit 0 is set, Kernel mode has ASTs enabled; bit 1, Executive mode; bit 2, Supervisor mode; and bit 3, User mode.

JPI$_ASTLM

Returns the AST limit quota of the process, which is a longword integer value.

JPI$_AUTHPRI

Returns the authorized base priority of the process, which is a longword integer value. The authorized base priority is the highest priority a process without ALTPRI privilege can attain by means of the $SETPRI service.

JPI$_AUTHPRIV

Returns the privileges that the process is authorized to enable. These privileges are returned in a quadword privilege mask and are defined by the $PRVDEF macro.

JPI$_BIOCNT

Returns a count of the remaining buffered I/O quota, which is a longword integer value.

JPI$_BIOLM

Returns the buffered I/O limit quota of the process, which is a longword integer value.

JPI$_BUFIO

Returns a count of the buffered I/O operations of the process, which is a longword integer value.

JPI$_BYTCNT

Returns the remaining buffered I/O byte count quota of the process, which is a longword integer value.

JPI$_BYTLM

Returns the buffered I/O byte count limit quota of the process, which is a longword integer value.

JPI$_CASE_LOOKUP_PERM

On Alpha and Integrity server systems, returns information about the file name lookup case sensitivity of a specified process. This value is set for the life of the process unless the style is set again. Values are 0 (PPROP$K_CASE_BLIND) and 1 (PPROP$K_CASE_SENSITIVE).

For additional information, see the *VSI OpenVMS Guide to OpenVMS File Applications*.

JPI$_CASE_LOOKUP_TEMP

On Alpha and Integrity server systems, returns information about the file name lookup case sensitivity of a specified process. This value is set only for the life of the image. Values are 0 (PPROP$K_CASE_BLIND) and 1 (PPROP$K_CASE_SENSITIVE).

For additional information, see the *VSI OpenVMS Guide to OpenVMS File Applications*.

JPI$_CHAIN

Processes another item list immediately after processing the current one. The buffer address field in the item descriptor specifies the address of the next item list to be processed. You must specify the JPI$_CHAIN item code last in the item list.

You can chain together 32-bit and 64-bit item lists.

JPI$_CLASSIFICATION

On Alpha and Integrity server systems, returns, as a 20-byte padded string, the current MAC classification stored in the PSB.

JPI$_CLINAME

Returns the name of the command language interpreter that the process is currently using. Because the CLI name can include up to 39 characters, the buffer length field in the item descriptor should specify 39 bytes.

JPI$_CPU_ID

Returns, as a longword integer, the ID of the CPU on which the process is running or on which it last ran. This value is returned as –1 if the system is not a multiprocessor.

JPI$_CPULIM

Returns the CPU time limit of the process, which is a longword integer value.

JPI$_CPUTIM

Returns the process's accumulated CPU time in 10-millisecond ticks, which is a longword integer value.

JPI$_CREPRC_FLAGS

Returns the flags specified by the *stsflg* argument in the $CREPRC call that created the process. The flags are returned as a longword bit vector.

JPI$_CURPRIV

Returns the current privileges of the process. These privileges are returned in a quadword privilege mask and are defined by the $PRVDEF macro.

JPI$_CURRENT_AFFINITY_MASK

On Alpha and Integrity server systems, returns the current explicit affinity mask for the associated kernel thread.

JPI$_CURRENT_USERCAP_MASK

On Alpha and Integrity server systems, returns the current user capability mask for the associated kernel thread.

JPI$_DEADLOCK_WAIT

Returns the per-process deadlock wait ticks. This value is in 100-nsec units.

JPI$_DFMBC

Returns the default multibuffer count for a process as a longword integer value.

JPI$_DFPFC

Returns the default page fault cluster size of the process, which is a longword integer value measured in pagelets.

JPI$_DFWSCNT

Returns, in pagelets , the default working set size of the process, which is a longword integer value.

JPI$_DIOCNT

Returns the remaining direct I/O quota of the process, which is a longword integer value.

JPI$_DIOLM

Returns the direct I/O quota limit of the process, which is a longword integer value.

JPI$_DIRIO

Returns a count of the direct I/O operations of the process, which is a longword integer value.

JPI$_EFCS

Returns the state of the process's local event flags 0 through 31 as a longword bit vector.

JPI$_EFCU

Returns the state of the process's local event flags 32 through 63 as a longword bit vector.

JPI$_EFWM

Returns the event flag wait mask of the process, which is a longword bit vector.

JPI$_ENQCNT

Returns the remaining lock request quota of the process, which is a longword integer value.

JPI$_ENQLM

Returns the lock request quota of the process, which is a longword integer value.

JPI$_EXCVEC

Returns the address of a list of exception vectors for the process. Each exception vector in the list is a longword. There are eight vectors in the list: these are, in order, a primary and a secondary vector for kernel mode access, for executive mode access, for supervisor mode access, and for user mode access.

The $GETJPI service cannot return this information for any process other than the calling process; if you specify this item code and the process is not the calling process, $GETJPI returns the value 0 in the buffer.

JPI$_FILCNT

Returns the remaining open file quota of the process, which is a longword integer value.

JPI$_FILLM

Returns the open file limit quota of the process, which is a longword value.

JPI$_FINALEXC

Returns the address of a list of final exception vectors for the process. Each exception vector in the list is a longword. There are four vectors in the list, one for each access mode, in this order: kernel, executive, supervisor, and user.

The $GETJPI service cannot return this information for any process other than the calling process; if you specify this item code and the process is not the calling process, $GETJPI returns the value 0 in the buffer.

JPI$_FREP0VA

Returns the address of the first free page at the end of the program region (P0 space) of the process.

JPI$_FREP1VA

Returns the address of the first free page at the end of the control region (P1 space) of the process.

JPI$_FREPTECNT

Returns the number of pagelets that the process has available for virtual memory expansion.

On Alpha and Integrity server systems, the value returned requires a quadword of storage. If the buffer size supplied is not equal to 8 bytes, and the number of free pagelets exceeds the maximum value that can be represented in a longword, $GETJPI returns the largest positive 32-bit integer: 2147483647.

JPI$_GETJPI_CONTROL_FLAGS

The JPI$_GETJPI_CONTROL_FLAGS item code, which is specified in the $GETJPI item list, provides additional control over $GETJPI; therefore, $GETJPI might be unable to retrieve all the data requested in an item list because JPI$_GETJPI_CONTROL_FLAGS requests that $GETJPI not perform certain actions that might be necessary to collect the data. For example, a $GETJPI control flag might instruct the calling program not to retrieve a process that has been swapped out of the balance set.

If $GETJPI is unable to retrieve any data item because of the restrictions imposed by the control flags, it returns the data length as 0. To verify that $GETJPI received a data item, examine the data length to be sure that it is not 0. To ensure the verification, be sure to specify the return length for each item in the $GETJPI item list when any of the JPI$_GETJPI_CONTROL_FLAGS flags is used.

Unlike other $GETJPI item codes, the JPI$_GETJPI_CONTROL_FLAGS item is an input item. The item list entry should specify a longword buffer. The desired control flags should be set in this buffer.

Because the JPI$_GETJPI_CONTROL_FLAGS item code tells $GETJPI how to interpret the item list, it must be the first entry in the $GETJPI item list. The error code SS$_BADPARAM is returned if it is not the first item in the list.

The JPI$_GETJPI_CONTROL_FLAGS item code includes the following flags:

Flag	Description
JPI$M_NO_TARGET_INSWAP	Does not retrieve a process that has been swapped out of the balance set. This control flag is used to avoid adding the load of swapping processes into a system. By using this control flag and requesting information from a process that has been swapped out, the following occurs: • Any data stored in the virtual address space of the process is not accessible. • Any data stored in the process header (PHD) might not be accessible. • Any data stored in resident data structures, such as the process control block (PCB) or the job information block (JIB), is accessible.

Flag	Description
	You must examine the return length of an item to verify that the item was retrieved.
JPI$M_NO_TARGET_AST	Does not deliver a kernel mode AST to the target process. This control flag is used to avoid executing a target process to retrieve information. By using this control flag and not delivering an AST to a target process, the following occurs: • Any data stored in the virtual address space of the process is not accessible. • Any data stored in system data structures, such as the process header (PHD), the process control block (PCB), or the job information block (JIB), is accessible. You must examine the return length of an item to verify that the item was retrieved. The use of this control flag also implies that $GETJPI does not swap in a process, because $GETJPI would only bring a process into memory to deliver an AST to that process.
JPI$M_IGNORE_TARGET_STATUS	Attempts to retrieve as much information as possible, even though the process might be suspended or is being deleted. This control flag is used to retrieve all possible information from a process.
JPI$M_THREAD	Sets the wildcard mode to return information on all of the process's kernel threads beginning with the initial kernel thread.

JPI$_GPGCNT

Returns, in pagelets, the process's global page count in the working set, which is a longword integer value.

JPI$_GRP

Returns, as a longword integer value, the group number of the process's UIC.

JPI$_HOME_RAD

Returns the home RAD. RAD is supported on AlphaServer GS series systems and starting from OpenVMS Version 8.4, support is extended to NUMA capable Integrity servers.

JPI$_IMAGECOUNT

Returns, as a longword integer value, the number of images that have been run down for the process.

JPI$_IMAGE_AUTHPRIV

On Alpha and Integrity server systems, returns the authorized privilege mask of the installed image.

These privileges are returned in a quadword privilege mask and are defined by the $PRVDEF macro.

JPI$_IMAGE_PERMPRIV

On Alpha and Integrity server systems, returns the permanent (default) privilege mask of the installed image.

These privileges are returned in a quadword privilege mask and are defined by the $PRVDEF macro.

JPI$_IMAGE_RIGHTS

Returns the binary content of the image rights list as an array of quadword identifiers. Each entry consists of a longword identifier value and longword identifier attributes, as shown in Table 42. The image rights list is a set of identifiers associated with a protected subsystem image. When a process runs a protected subsystem, the subsystem rights are automatically added to the process's image rights list. These identifiers are subsequently removed during image rundown. Allocate a buffer that is sufficient to hold the image rights list, because $GETJPI returns only as much of the list as will fit in the buffer.

Table 42. Attributes of an Identifier

Symbolic Name	Description
KGB$M_DYNAMIC	Identifier can be enabled or disabled.
KGB$M_RESOURCE	Resources can be charged to the identifier.

JPI$_IMAGE_WORKPRIV

On Alpha and Integrity server systems, returns the working (active) privilege mask of the installed image.

These privileges are returned in a quadword privilege mask and are defined by the $PRVDEF macro.

JPI$_IMAGNAME

Returns, as a character string, the directory specification and the image file name.

JPI$_IMAGPRIV

Returns a quadword mask of the privileges with which the current image was installed. If the current image was not installed, $GETJPI returns the value 0 in the buffer.

JPI$_INITIAL_THREAD_PID

On Alpha and Integrity server systems, returns the PID of the initial thread for the target process. The PID is a longword hexadecimal value.

JPI$_INSTALL_RIGHTS

On Alpha and Integrity server systems, returns the binary content of the install rights list as an array of quadword identifiers. Each entry consists of a longword identifier value and longword identifier attributes, as shown in Table 42. The install rights list is a set of identifiers associated with an installed image.

JPI$_INSTALL_RIGHTS_SIZE

On Alpha and Integrity server systems, returns a longword integer containing the number of bytes needed to store the install rights.

JPI$_JOBPRCCNT

Returns the total number of subprocesses owned by the job, which is a longword integer value.

JPI$_JOBTYPE

Returns the execution mode of the process at the root of the job tree, which is a longword integer value. The symbolic name and value for each execution mode are listed in the following table. The $JPIDEF macro defines the symbolic names.

Mode Name	Value
JPI$K_DETACHED	0
JPI$K_NETWORK	1
JPI$K_BATCH	2
JPI$K_LOCAL	3
JPI$K_DIALUP	4
JPI$K_REMOTE	5

JPI$_KT_COUNT

On Alpha and Integrity server systems, returns the current count of kernel threads for the target process, which is a longword integer value.

JPI$_KT_LIMIT

On Alpha and Integrity server systems, returns the maximum number of kernel threads that can be created in the process. A value of 0 indicates that the process does not have a process-specific limit set, and the systemwide limit specified by the SYSGEN parameter MULTITHREAD is used.

JPI$_LAST_LOGIN_I

Returns, as a quadword absolute time value, the date of the last successful interactive login prior to the current session. It returns a quadword of 0 when processes have not executed the LOGINOUT image.

JPI$_LAST_LOGIN_N

Returns, as a quadword absolute time value, the date of the last successful noninteractive login prior to the current session. It returns a quadword of 0 when processes have not executed the LOGINOUT image.

JPI$_LOGIN_FAILURES

Returns the number of login failures that occurred prior to the current session. It returns a longword of 0 when processes have not executed the LOGINOUT image.

JPI$_LOGIN_FLAGS

Returns a longword bit mask containing information related to the login sequence. It returns a longword of 0 when processes have not executed the LOGINOUT image. The following bits are defined:

Symbolic Name	Description
JPI$M_NEW_MAIL_AT_LOGIN	User had new mail messages waiting at login.
JPI$M_PASSWORD_CHANGED	User changed the primary password during login.

Symbolic Name	Description
JPI$M_PASSWORD_EXPIRED	User's primary password expired during login.
JPI$M_PASSWORD_WARNING	System gave the user a warning at login that the account's primary password would expire within 5 days.
JPI$M_PASSWORD2_CHANGED	Account's secondary password was changed during login.
JPI$M_PASSWORD2_EXPIRED	Account's secondary password expired during login.
JPI$M_PASSWORD2_WARNING	System gave the user a warning at login that the account's secondary password would expire within 5 days.

JPI$_LOGINTIM

Returns the time at which the process was created, which is a standard 64-bit absolute time.

JPI$_MASTER_PID

Returns the process identification (PID) of the master process in the job. The PID is a longword hexadecimal value.

JPI$_MAXDETACH

Returns the maximum number of detached processes allowed for the user who owns the process specified in the call to $GETJPI. This limit is set in the UAF record of the user. The number is returned as a word decimal value. A value of 0 means that there is no limit on the number of detached processes for that user name.

JPI$_MAXJOBS

Returns the maximum number of active processes allowed for the user who owns the process specified in the call to $GETJPI. This limit is set in the UAF record of the user. The number is returned as a word decimal value. A value of 0 means that there is no limit on the number of active processes for that user name.

JPI$_MEM

Returns the member number of the process's UIC, which is a longword integer value.

JPI$_MODE

Returns the mode of the process, which is a longword integer value. The symbolic name and value for each mode are listed in the following table. The $JPIDEF macro defines the symbolic names.

Mode Name	Value
JPI$K_OTHER	0
JPI$K_NETWORK	1
JPI$K_BATCH	2
JPI$K_INTERACTIVE	3

JPI$_MSGMASK

Returns the default message mask of the process, which is a longword bit mask.

JPI$_MULTITHREAD

On Alpha and Integrity server systems, returns the maximum kernel thread count allowed for the target process, which is a longword integer value.

JPI$_NODENAME

Returns, as a character string, the name of the OpenVMS Cluster node on which the process is running.

JPI$_NODE_CSID

Returns, as a longword hexadecimal integer, the cluster ID of the OpenVMS Cluster node on which the process is running.

JPI$_NODE_VERSION

Returns, as a character string, the operating system version number of the OpenVMS Cluster node on which the process is running.

JPI$_OWNER

Returns the process identification (PID) of the process that created the specified process. The PID is a longword hexadecimal value.

JPI$_PAGEFLTS

Returns the total number of page faults incurred by the process. This is a longword integer value.

JPI$_PAGFILCNT

Returns the remaining paging file quota of the process, which is a longword integer value, measured in pages pagelets.

JPI$_PAGFILLOC

Returns the current paging file assignment of the process. The fourth byte of the returned longword value is the index of the system page file to which the process is currently assigned.

JPI$_PARSE_STYLE_IMAGE

On Alpha and Integrity server systems, set by $SET_PROCESS_PROPERTIESW, and can be either PARSE_STYLE$C_TRADITIONAL or PARSE_STYLE$C_EXTENDED (located in PPROPDEF). The return length is one byte.

JPI$_PARSE_STYLE_PERM

On Alpha and Integrity server systems, set by $SET_PROCESS_PROPERTIESW, and can be either PARSE_STYLE$C_TRADITIONAL or PARSE_STYLE$C_EXTENDED (located in PPROPDEF). The return length is one byte.

JPI$_PERMANENT_AFFINITY_MASK

On Alpha and Integrity server systems, returns the permanent explicit affinity mask for the associated kernel thread.

JPI$_PERMANENT_USERCAP_MASK

On Alpha and Integrity server systems, returns the permanent explicit affinity mask for the associated kernel thread.

JPI$_PERSONA_AUTHPRIV

On Alpha and Integrity server systems, returns the authorized privilege mask of the persona.

These privileges are returned in a quadword privilege mask and are defined by the $PRVDEF macro.

JPI$_PERSONA_ID

On Alpha and Integrity server systems, returns, as a longword integer, the ID of the persona.

JPI$_PERSONA_PERMPRIV

On Alpha and Integrity server systems, returns the permanent (default) privilege mask of the persona. These privileges are returned in a quadword privilege mask and are defined by the $PRVDEF macro.

JPI$_PERSONA_RIGHTS

On Alpha and Integrity server systems, returns the binary content of the persona rights list as an array of quadword identifiers. Each entry consists of a longword identifier value and longword identifier attributes, as listed in Table 42. The persona rights list is a set of identifiers associated with the process.

JPI$_PERSONA_RIGHTS_SIZE

On Alpha and Integrity server systems, returns a longword integer containing the number of bytes needed to store the persona rights.

JPI$_PERSONA_WORKPRIV

On Alpha and Integrity server systems, returns the privilege mask of the working (active) persona. These privileges are returned in a quadword privilege mask and are defined by the $PRVDEF macro.

JPI$_PGFLQUOTA

Returns the paging file quota (maximum virtual page count) of the process, which is a longword integer value, measured in pagelets .

JPI$_PHDFLAGS

Returns the process header flags as a longword bit vector.

JPI$_PID

Returns the process identification (PID) of the process. The PID is a longword hexadecimal value.

JPI$_P0_FIRST_FREE_VA_64

On Alpha and Integrity server systems, this item code returns the 64-bit virtual address of the first free page at the end of the program region (P0 space) of the process.

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

JPI$_P1_FIRST_FREE_VA_64

On Alpha and Integrity server systems, this item code returns the 64-bit virtual address of the first free page at the end of the control region (P1 space) of the process.

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

JPI$_P2_FIRST_FREE_VA_64

On Alpha and Integrity server systems, this item code returns the 64-bit virtual address of the first free page at the end of P2 space of the process.

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

JPI$_PPGCNT

Returns the number of pagelets the process has in the working set. This is a longword integer value.

JPI$_PRCCNT

Returns, as a longword integer value, the number of subprocesses created by the process. The number returned by JPI$_PRCCNT does not include any subprocesses created by subprocesses of the process named in the *procnam* argument.

JPI$_PRCLM

Returns the subprocess quota of the process, which is a longword integer value.

JPI$_PRCNAM

Returns, as a character string, the name of the process. Because the process name can include up to 15 characters, the buffer length field of the item descriptor should specify at least 15 bytes.

JPI$_PRI

Returns the current priority of the process, which is a longword integer value.

JPI$_PRIB

Returns the base priority of the process, which is a longword integer value.

JPI$_PROCESS_RIGHTS

Returns the binary content of the process rights list as an array of quadword identifiers. Each entry consists of a longword identifier value and longword identifier attributes, as shown in Table 42. Allocate a buffer that is sufficient to hold the process rights list because $GETJPI returns only as much of the list as will fit in the buffer.

JPI$_PROC_INDEX

Returns, as a longword integer value, the process index number of the process. The process index number is a number between 1 and the system parameter MAXPROCESSCNT, which identifies the process. Although process index numbers are reassigned to different processes over time, at any one instant, each process in the system has a unique process index number.

You can use the process index number as an index into system global sections. Because the process index number is unique for each process, its use as an index into system global sections guarantees no collisions with other system processes accessing those sections.

The process index is intended to serve users who formerly used the low-order word of the PID as an index number.

JPI$_PROCPRIV

Returns the default privileges of the process in a quadword bit mask.

JPI$_RIGHTSLIST

Returns, as an array of quadword identifiers, all identifiers applicable to the process. This includes the process rights list (JPI$_PROCESS_RIGHTS) and the system rights list (JPI$_SYSTEM_RIGHTS). Each entry consists of a longword identifier value and longword identifier attributes, shown in Table 42. Allocate a buffer that is sufficient to hold the rights list because $GETJPI returns only as much of the list as will fit in the buffer.

JPI$_RIGHTS_SIZE

Returns the number of bytes required to buffer the rights list. The rights list includes both the system rights list and the process rights list. Because the space requirements for the rights list can change between the time you request the size of the rights list and the time you fetch the rights list with JPI$_RIGHTSLIST, you might want to allocate a buffer that is 10 percent larger than this item indicates.

JPI$_RMS_DFMBC

Returns the default multiblock count.

JPI$_RMS_DFMBFIDX

Returns the default multibuffer count for local buffers for indexed file operations.

JPI$_RMS_DFMBFREL

Returns the default multibuffer count for local buffers for relative file operations.

JPI$_RMS_DFMBFSDK

Returns the default multibuffer count for local buffers for sequential file operations on disk.

JPI$_RMS_DFMBFSMT

Returns the default multibuffer count for local buffers for sequential file operations on magnetic tape.

JPI$_RMS_DFMBFSUR

Returns the default multibuffer count for file operation on unit record devices.

JPI$_RMS_DFNBC

Returns the default block count for local buffers for network access to remote files.

JPI$_RMS_EXTEND_SIZE

Returns the default number of blocks used to extend a sequential file.

JPI$_RMS_FILEPROT

Returns the default file protection.

JPI$_RMS_PROLOGUE

Returns the default prolog level for indexed files.

JPI$_SCHED_CLASS_NAME

Returns the name of the scheduling class (as a character string) to which this process belongs. Because the class name can include up to 16 characters, the buffer length field of the item descriptor must specify at least 16 bytes. If the process is not class scheduled, then a return length of 0 is returned to the caller.

JPI$_SCHED_POLICY

On Alpha and Integrity server systems, returns the current scheduling policy of the specified process. Definitions of the policy values are in the $JPIDEF macro. The buffer length of the item descriptor should specify 4 (bytes).

JPI$_SEARCH_SYMLINK_PERM

On Alpha and Integrity server systems, returns the permanent process symlink search modes. Values are JPI$K_SEARCH_SYMLINK_NONE, JPI$K_SEARCH_SYMLINK_ALL, and JPI$K_SEARCH_SYMLINK_NOELLIPS.

JPI$_SEARCH_SYMLINK_TEMP

On Alpha and Integrity server systems, returns the temporary process symlink search modes. Values are JPI$K_SEARCH_SYMLINK_NONE, JPI$K_SEARCH_SYMLINK_ALL, and JPI$K_SEARCH_SYMLINK_NOELLIPS.

JPI$_SHRFILLM

Returns the maximum number of open shared files allowed for the job to which the process specified in the call to $GETJPI belongs. This limit is set in the UAF record of the user who owns the process. The number is returned as a word decimal value. A value of 0 means that there is no limit on the number of open shared files for that job.

JPI$_SITESPEC

Returns the per-process, site-specific longword, which is a longword integer value.

JPI$_STATE

Returns the state of the process, which is a longword integer value. Each state has a symbolic representation. If the process is currently executing, its state is always SCH$K_CUR. STATEDEF defines the following symbols, which identify the various possible states:

State	Description
SCH$C_CEF	Common event flag wait
SCH$C_COM	Computable
SCH$C_COMO	Computable, out of balance set
SCH$C_CUR	Current process
SCH$C_COLPG	Collided page wait

State	Description
SCH$C_FPG	Free page wait
SCH$C_HIB	Hibernate wait
SCH$C_HIBO	Hibernate wait, out of balance set
SCH$C_LEF	Local event flag wait
SCH$C_LEFO	Local event flag wait, out of balance set
SCH$C_MWAIT	Mutex and miscellaneous resource wait
SCH$C_PFW	Page fault wait
SCH$C_SUSP	Suspended
SCH$C_SUSPO	Suspended, out of balance set

JPI$_STS

Returns the first longword of the process status flags, which are contained in a longword bit vector. PCBDEF definitions include the following symbols for these flags:

Symbol	Description
PCB$V_ASTPEN	AST pending
PCB$V_BATCH	Process is a batch job
PCB$V_DELPEN	Delete pending
PCB$V_DISAWS	Disable automatic working set adjustment
PCB$V_FORCPEN	Force exit pending
PCB$V_HARDAFF	Process bound to a particular CPU
PCB$V_HIBER	Hibernate after initial image activate
PCB$V_INQUAN	Initial quantum in progress
PCB$V_INTER	Process is an interactive job
PCB$V_LOGIN	Log in without reading authorization file
PCB$V_NETWRK	Process is a network connect object
PCB$V_NOACNT	No accounting for process
PCB$V_NODELET	No delete
PCB$V_PHDRES	Process header resident
PCB$V_PREEMPTED	Kernel mode suspend has overridden supervisor mode suspend
PCB$V_PSWAPM	Process swap mode (1=noswap)
PCB$V_PWRAST	Power fail AST
PCB$V_RECOVER	Process can recover locks
PCB$V_RES	Resident, in balance set
PCB$V_RESPEN	Resume pending, skip suspend
PCB$V_SECAUDIT	Mandatory security auditing
PCB$V_SOFTSUSP	Process is in supervisor mode suspend
PCB$V_SSFEXC	System service exception enable (kernel)
PCB$V_SSFEXCE	System service exception enable (exec)

Symbol	Description
PCB$V_SSFEXCS	System service exception enable (super)
PCB$V_SSFEXCU	System service exception enable (user)
PCB$V_SSRWAIT	System service resource wait disable
PCB$V_SUSPEN	Suspend pending
PCB$V_WAKEPEN	Wake pending, skip hibernate
PCB$V_WALL	Wait for all events in mask

JPI$_STS2

Returns the second longword of the process status flags, which are contained in a longword bit vector. PCBDEF defines the following symbol for these flags:

Symbol	Description
PCB$V_NOUNSHELVE	Process does not automatically unshelve files.

JPI$_SUBSYSTEM_RIGHTS

On Alpha and Integrity server systems, returns the binary content of the subsystem rights list as an array of quadword identifiers. Each entry consists of a longword identifier value and longword identifier attributes, as listed in Table 42. The subsystem rights list is a set of identifiers associated with a protected subsystem image.

When a process runs a protected subsystem, the subsystem rights are automatically added to the process's image rights list. These identifiers are subsequently removed during image rundown. Allocate a buffer that is sufficient to hold the subsystem rights list, because $GETJPI returns only as much of the list as will fit in the buffer.

JPI$_SUBSYSTEM_RIGHTS_SIZE

On Alpha and Integrity server systems, returns a longword integer containing the number of bytes needed to store the subsystem rights.

JPI$_SWPFILLOC

Returns the location of the process's swapping file, which is a longword hexadecimal value. If the number returned is positive, the fourth byte of this value identifies a specific swapping file, and the lower three bytes contain the VBN within the swapping file. If the number returned is 0 or negative, the swap file location information is not currently available for the process.

JPI$_SYSTEM_RIGHTS

Returns the system rights list as an array of quadword identifiers. Each entry consists of a longword identifier value and longword identifier attributes, listed in Table 42. Allocate a buffer that is sufficient to hold the system rights list because $GETJPI only returns as much of the list as will fit in the buffer.

JPI$_SYSTEM_RIGHTS_SIZE

On Alpha and Integrity server systems, returns a longword integer containing the number of bytes needed to store the system rights.

JPI$_TABLENAME

Returns the file specification of the process's current command language interpreter (CLI) table. Because the file specification can include up to 255 characters, the buffer length field in the item descriptor should specify 255 bytes.

JPI$_TERMINAL

Returns, for interactive users, the process's login terminal name as a character string. Because the terminal name can include up to 16 characters, the buffer length field in the item descriptor should specify at least 16 bytes. Trailing zeros are written to the output buffer if necessary.

JPI$_THREAD_INDEX

On Alpha and Integrity server systems, returns the kernel thread index for the target thread or process, which is a longword integer value.

JPI$_TMBU

Returns the termination mailbox unit number, which is a longword integer value.

JPI$_TQCNT

Returns the remaining timer queue entry quota of the process, which is a longword integer value.

JPI$_TQLM

Returns the process's limit on timer queue entries, which is a longword integer value.

JPI$_TT_ACCPORNAM

Returns the access port name for the terminal associated with the process. (The terminal name is returned by JPI$_TERMINAL.) This item code can return the following information:

- If the terminal is on a terminal server, this item returns the terminal server name and the name of the line port on the server.
- If the terminal is a DECnet for OpenVMS remote terminal, this item returns the source system node name and the user name on the source system.
- If the terminal is on TELNET, this item returns the originating host address and port.
- If the terminal not on a terminal server, on DECnet, or on TELNET, this item returns a null string.

JPI$_TT_PHYDEVNAM

Returns the physical device name of the terminal associated with the process. This name is the same as JPI$_TERMINAL unless virtual terminals are enabled, in which case JPI$_TERMINAL returns the name of the virtual terminal and JPI$_TT_PHYDEVNAM returns the name of the physical terminal. If JPI$_TERMINAL is null or if the virtual terminal is disconnected from the physical terminal, JPI$_TT_PHYDEVNAM returns a null string.

JPI$_UAF_FLAGS

Returns the UAF flags from the UAF record of the user who owns the process. The flags are returned as a longword bit vector. For a list of the symbolic names of these flags, see the UAI$_FLAGS item code under the $GETUAI system service.

JPI$_UIC

Returns the UIC of the process in the standard longword format.

JPI$_USERNAME

Returns the user name of the process as a 12-byte string. If the name is less than 12 bytes, $GETJPI fills out the 12 bytes with trailing blanks and always returns 12 as the string length.

JPI$_VIRTPEAK

Returns the peak virtual address size—in pagelets for Alpha or Integrity servers—of the process.

On Alpha and Integrity server systems, the value returned requires a quadword of storage. If the buffer size supplied is not equal to 8 bytes, and the virtual peak exceeds the maximum value that can be represented in a longword, $GETJPI returns the largest positive 32-bit integer: 2147483647.

JPI$_VOLUMES

Returns the count of volume mount operations that the process has done, which is a longword integer value.

JPI$_WSAUTH

Returns the maximum authorized working set size, in pagelets , of the process. This is a longword integer value.

JPI$_WSAUTHEXT

Returns, in pagelets, the maximum authorized working set extent of the process as a longword integer value.

JPI$_WSEXTENT

Returns, in pagelets, the current working set extent of the process as a longword integer value.

JPI$_WSPEAK

Returns, in pagelets, the peak working set size of the process as a longword integer value.

JPI$_WSQUOTA

Returns, in pagelets, the working set size quota of the process as a longword integer value.

JPI$_WSSIZE

Returns, in pagelets , the current working set limit of the process as a longword integer value.

Description

The Get Job/Process Information service returns information about one or more processes on the system or across the cluster. Using $GETJPI with $PROCESS_SCAN, you can perform selective or clusterwide searches.

Getting information about another process is an asynchronous operation because the information might be contained in the virtual address space of the target process, and that process might be running at a lower priority, be outswapped, or be suspended in a miscellaneous or resource wait state. To

allow your program to overlap other functions with the time needed to access the data in the other process, $GETJPI returns immediately after it has queued its information-gathering request to the other process. You can use the JPI$_GETJPI item code to control the processing of the $GETJPI call and the information-gathering interprocess request itself.

When performing an asynchronous system service call such as $GETJPI, the specifications of the *iosb* argument and a unique event flag are used in conjunction with mechanisms such as the $SYNCH system service to synchronize the final completion of the asynchronous system service call.

Required Access or Privileges

The calling process must have GROUP privilege to obtain information about other processes with the same group UIC number as the calling process. The calling process must have WORLD privilege to obtain information about other processes on the system that are not in the same group as the calling process.

Using *prcnam* and *pidadr* Arguments

Most process control services accept the *prcnam* or the *pidadr* argument or both. However, you should identify a process by its process identification number for the following reasons:

- The service executes faster because it does not have to search a table of process names.
- For a process not in your group, you must use the process identification number.

If you specify the PID address, the service uses the PID address. If you specify the process name without a PID address, the service uses the process name. If you specify both – the process name and PID address – it uses the PID address unless the contents of the PID is 0. In that case, the service uses the process name. If you specify a PID address of 0 without a process name, then the service is performed for the calling process.

If you specify neither the process name argument nor the process identification number argument, the service is performed for the calling process. If the PID address is specified, the service returns the PID of the target process in it. Table 43 summarizes the possible combinations of these arguments and explains how the services interpret them.

Table 43. Process Identification

Process Name Specified?	PID Address Specified?	Contents of PID	Resultant Action by Services
No	No	–	The process identification of the calling process is used, but is not returned.
No	Yes	0	The process identification of the calling process is used and returned.
No	Yes	PID	The process identification is used and returned.
Yes	No	–	The process name is used. The process identification is not returned.
Yes	Yes	0	The process name is used and the process identification is returned.
Yes	Yes	PID	The process identification is used and returned; the process name is ignored.

Required Quota

None

Related Services

$GETJPIW, $HIBER, $PROCESS_SCAN, $RESUME, $SYNCH

Condition Values Returned

SS$_NORMAL

The service completed successfully.

SS$_ACCVIO

The item list cannot be read by the caller, or the buffer length or buffer cannot be written by the caller.

SS$_BADPARAM

The item list contains an invalid identifier. Or, an item list containing both 32-bit and 64-bit item list entries was found.

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$_NOMOREPROC

In a wildcard operation, $GETJPI found no more processes.

SS$_NOMORETHREAD

The search for kernel threads within a process is complete. This condition value is returned by $GETJPIW if you set the JPI$M_THREAD bit in JPI$_GETJPI_CONTROL_FLAGS.

SS$_NONEXPR

The specified process does not exist, or an invalid process identification was specified.

SS$_NOPRIV

The process does not have the privilege to obtain information about the specified process.

SS$_NOSUCHNODE

The specified node is not currently a member 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$_SUSPENDED

The specified process is suspended or in a miscellaneous wait state, and the requested information cannot be obtained.

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.

Condition Values Returned in the I/O Status Block

Same as those returned in R0.

$GETJPIW

Get Job/Process Information and Wait — Returns information about one or more processes on the system.

Format

```
SYS$GETJPIW
   [efn] ,[pidadr] ,[prcnam] ,itmlst ,[iosb] ,[astadr] ,[astprm]
```

C Prototype

```
int sys$getjpiw
   (unsigned int efn, unsigned int *pidadr, void *prcnam, void *itmlst,
    struct _iosb *iosb,  void (*astadr)(__unknown_params), int astprm);
```

Description

The $GETJPIW service completes synchronously; that is, it returns to the caller with the requested information. VSI recommends that you use an IOSB with this service. An IOSB prevents the service from completing prematurely. In addition, the IOSB contains status information.

For asynchronous completion, use the Get Job/Process Information ($GETJPI) service; $GETJPI returns to the caller after queuing the information request, without waiting for the information to be returned.

In all other respects, $GETJPIW is identical to $GETJPI. For all other information about the $GETJPIW service, see the description of $GETJPI in this manual.

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

$GETLKI

Get Lock Information — Returns information about the lock database on a system. The $GETLKI service completes asynchronously; for synchronous completion, use the Get Lock Information and Wait ($GETLKIW) service. The $GETLKI, $GETLKIW, $ENQ, $ENQW, and $DEQ services together provide the user interface to the Lock Management facility.

Format

```
SYS$GETLKI
   [efn] ,lkidadr ,itmlst [,iosb] [,astadr] [,astprm] [,nullarg]
```

C Prototype

```
int sys$getlki
   (unsigned int efn, unsigned int *lkidadr, void *itmlst,
    struct _iosb *iosb, void (*astadr)(__unknown_params),
    int astprm, unsigned int reserved);
```

Arguments

efn

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

Number of the event flag to be set when $GETLKI completes. The *efn* argument is a longword containing this number; however, $GETLKI uses only the low-order byte. If you do not specify *efn*, $GETLKI sets event flag 0.

VSI strongly recommends the use of the EFN$C_ENF "no event flag" value as the event flag if you are not using an event flag to externally synchronize with the completion of this system service call. The $EFNDEF macro defines EFN$C_ENF. For more information, see the *VSI OpenVMS Programming Concepts Manual*.

lkidadr

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

Lock identification (lock ID) for the lock about which information is to be returned. The lock ID is the second longword in the lock status block, which was created when the lock was granted. The *lkidadr* argument is the address of this longword.

If the value specified by *lkidadr* is 0 or –1, $GETLKI assumes a wildcard operation and returns information about each lock to which the calling process has access, one lock per call.

To use the $GETLKI service, you must have read/write access to the lock ID.

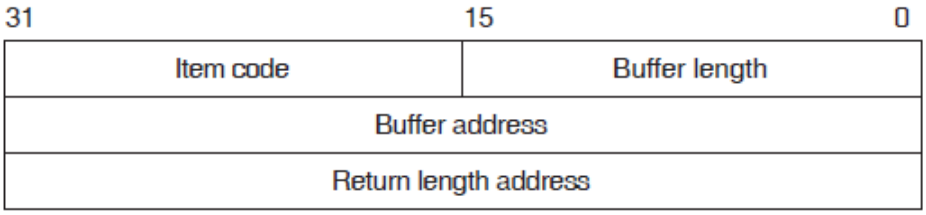
itmlst

OpenVMS usage:	item_list_3
type:	longword (unsigned)

access:	read only
mechanism:	by reference

Item list specifying the lock information that $GETLKI is to return. The *itmlst* argument is the address of a list of item descriptors, each of which describes an item of information. The list of item descriptors is terminated by a longword of 0.

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



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 $GETLKI is to write the information. The length of the buffer needed depends on the item code specified in the item code field of the item descriptor. If the value of the buffer length field is too small, $GETLKI truncates the data and returns the success condition value SS$_NORMAL.	
Item code	A word containing a user-specified symbolic code specifying the item of information that $GETLKI is to return. The $LKIDEF macro defines these codes. Each item code is described in the list of $GETLKI item codes that follows the argument descriptions.	
Buffer address	A longword containing a user-supplied address of the buffer in which $GETLKI is to write the information.	
Return length address	A longword containing the user-supplied address of a longword in which $GETLKI writes return length information. This longword contains the following three bit fields.	
	Bits 0 to 15	In this, field $GETLKI writes the length in bytes of the data actually written to the buffer specified by the buffer address field in the item descriptor.
	Bits 16 to 30	$GETLKI uses this field only when the item code field of the item descriptor specifies LKI$_BLOCKEDBY, LKI$_BLOCKING, or LKI$_LOCKS, each of which requests information about a list of locks. $GETLKI writes in this field the length in bytes of the information returned for a single lock on the list. You can divide this length into the total length returned for all locks (bits 0 to 15) to determine the number of locks located by that item code request.
	Bit 31	$GETLKI sets this bit if the user-supplied buffer length argument specifies too small a buffer to contain the information returned. Note that in such a case $GETLKI will return the SS$_NORMAL condition value in R0. Therefore, to locate any faulty item

Descriptor Field	Definition	
		descriptor, you need to check the state of bit 31 in the longword specified by the return length field of each item descriptor.

iosb

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

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

When $GETLKI is called, it sets the I/O status block to 0. When $GETLKI completes, it writes a condition value to the first longword in the quadword. The remaining two words in the quadword are unused.

Although this argument is optional, VSI strongly recommends that you specify it, for the following reasons:

- If you are using an event flag to signal the completion of the service, you can test the I/O status block for a condition value to be sure that the event flag was not set by an event other than service completion.
- If you are using the $SYNCH service to synchronize completion of the service, the I/O status block is a required argument for $SYNCH.
- The condition value returned in R0 and the condition value returned in the I/O status block provide information about different aspects of the call to the $GETLKI service. The condition value returned in R0 gives you information about the success or failure of the service call itself; the condition value returned in the I/O status block gives you information about the success or failure of the service operation. Therefore, to accurately assess the success or failure of the call to $GETLKI, you must check the condition values returned in both R0 and the I/O status block.

astadr

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

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

If you specify this argument, the AST routine executes at the same access mode as the caller of the $GETLKI 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.

nullarg

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

Placeholding argument reserved to OpenVMS.

Item Codes

LKI$_BLOCKEDBY

When you specify LKI$_BLOCKEDBY, $GETLKI returns information about all locks that are currently blocked by the lock specified by *lkidadr*. The $GETLKI service returns eight items of information about each blocked lock.

The $LKIDEF macro defines the following symbolic names that refer to the eight items in the buffer.

Symbolic Name	Description
LKI$L_MSTLKID	Lock ID of the blocked lock on the system maintaining the resource (4 bytes)
LKI$L_PID	Process ID (PID) of the process that took out the blocked lock (4 bytes)
LKI$L_MSTCSID	OpenVMS Cluster system identifier (CSID) of the node maintaining the resource that is locked by the blocked lock (4 bytes)
LKI$B_RQMODE	Lock mode requested for the blocked lock; this lock mode was specified by the *lkmode* argument in the call to $ENQ (1 byte)
LKI$B_GRMODE	Lock mode granted to the blocked lock; this lock mode is written to the lock value block (1 byte)
LKI$B_QUEUE	Name of the queue on which the blocked lock currently resides (1 byte)
LKI$L_LKID	Lock ID of the lock on the system where the lock was requested (4 bytes)
LKI$L_CSID	OpenVMS Cluster system identifier (CSID) of the system where the lock was requested (4 bytes)

The values that $GETLKI can write into the LKIB_RQMODE, LKIB_GRMODE, and LKI$B_QUEUE items have symbolic names; these symbolic names specify the six lock modes and the three types of queue in which a lock can reside. The Description section describes these names.

Thus, the buffer specified by the buffer address field in the item descriptor will contain the eight items of information, repeated in sequence, for each blocked lock.

The length of the information returned for each blocked lock is returned in bits 16 to 30 of the longword specified by the return length address field in the item descriptor, while the total length of information returned for all blocked locks is returned in bits 0 to 15. Therefore, to determine the number of blocked locks, you divide the value of bits 16 to 30 into the value of bits 0 to 15.

LKI$_BLOCKING

When you specify LKI$_BLOCKING, $GETLKI returns information about all locks that are currently blocking the lock specified by *lkidadr*. The $GETLKI service returns eight items of information about each blocking lock.

The $LKIDEF macro defines the following symbolic names that refer to the eight items in the buffer.

Symbolic Name	Description
LKI$L_MSTLKID	Lock ID of the blocked lock on the system maintaining the resource (4 bytes)
LKI$L_PID	Process ID (PID) of the process that took out the blocking lock (4 bytes)
LKI$L_MSTCSID	OpenVMS Cluster system identifier (CSID) of the node maintaining the resource that is locked by the blocking lock (4 bytes)
LKI$B_RQMODE	Lock mode requested for the blocking lock; this lock mode was specified by the *lkmode* argument in the call to $ENQ (1 byte)
LKI$B_GRMODE	Lock mode granted to the blocking lock; this lock mode is written to the lock value block (1 byte)
LKI$B_QUEUE	Name of the queue on which the blocking lock currently resides (1 byte)
LKI$L_LKID	Lock ID of the lock on the system where the lock was requested (4 bytes)
LKI$L_CSID	OpenVMS Cluster system identifier (CSID) of the system where the lock was requested (4 bytes)

The values that $GETLKI can write into the LKIB_RQMODE, LKIB_GRMODE, and LKI$B_QUEUE items have symbolic names; these symbolic names specify the six lock modes and the three types of queue in which a lock can reside. The Description section describes these names.

Thus, the buffer specified by the buffer address field in the item descriptor will contain the eight items of information, repeated in sequence, for each blocking lock.

The length of the information returned for each blocking lock is returned in bits 16 to 30 of the longword specified by the return length address field in the item descriptor, while the total length of information returned for all blocking locks is returned in bits 0 to 15. Therefore, to determine the number of blocking locks, you divide the value of bits 16 to 30 into the value of bits 0 to 15.

LKI$_CSID

When you specify LKI$_CSID, $GETLKI returns the Cluster System ID (CSID) of the system where the process owning the lock resides. LKI$_CSID returns the CSID of the node where the $GETLKI system service is issued when the resource is mastered on that node. When the processor is not part of a cluster, LKI$_CSID returns 0.

The buffer length field in the item descriptor should specify 4 (bytes).

LKI$_CVTCOUNT

When you specify LKI$_CVTCOUNT, $GETLKI returns the total number of locks that are currently on the conversion queue of the resource associated with the lock. These locks are granted at one mode and are waiting to be converted to another.

The buffer length field in the item descriptor should specify 4 (bytes).

LKI$_GRANTCOUNT

When you specify LKI$_GRANTCOUNT, $GETLKI returns the total number of locks that are currently on the grant queue of the resource associated with the lock. Note that the total number of granted locks on the resource is equal to the sum of LKI$_CVTCOUNT and LKI$_GRANTCOUNT.

The buffer length field in the item descriptor should specify 4 (bytes).

LKI$_LCKREFCNT

When you specify LKI$_LCKREFCNT, $GETLKI returns the number of locks that have this lock as a parent lock. When these locks were created, the *parid* argument in the call to $ENQ or $ENQW specified the lock ID of this lock.

The buffer length field in the item descriptor should specify 4 (bytes).

LKI$_LKID

When you specify LKI$_LKID, $GETLKI returns the lock ID of the lock on the system where the process owning the lock resides. The lock ID returned by this item code is meaningful only on the system specified in the value returned by the LKI$_CSID item code.

The buffer length field in the item descriptor should specify 4 (bytes).

LKI$_LOCKID

When you specify LKI$_LOCKID, $GETLKI returns the lock ID of the current lock. The current lock is the one specified by the *lkidadr* argument unless *lkidadr* is specified as –1 or 0, which indicates a wildcard operation. Thus, this item code is usually specified only in wildcard operations where it is useful to know the lock IDs of the locks that $GETLKI has discovered in the wildcard operation.

The lock ID is a longword value, so the buffer length field in the item descriptor should specify 4 (bytes).

LKI$_LOCKS

When you specify LKI$_LOCKS, $GETLKI returns information about all locks on the resource associated with the lock specified by *lkidadr*.

The $LKIDEF macro defines the following symbolic names that refer to the eight items in the buffer.

Symbolic Name	Description
LKI$L_MSTLKID	Lock ID of the blocked lock on the system maintaining the resource (4 bytes)
LKI$L_PID	Process ID (PID) of the process that took out the lock (4 bytes)
LKI$L_MSTCSID	OpenVMS Cluster system identifier (CSID) of the node maintaining the resource that is locked by the lock (4 bytes)
LKI$B_RQMODE	Lock mode requested for the lock; this lock mode was specified by the *lkmode* argument in the call to $ENQ (1 byte)
LKI$B_GRMODE	Lock mode granted to the lock; this lock mode is written to the lock value block (1 byte)
LKI$B_QUEUE	Name of the queue on which the lock currently resides (1 byte)
LKI$L_LKID	Lock ID of the lock on the system where the lock was requested (4 bytes)
LKI$L_CSID	OpenVMS Cluster system identifier (CSID) of the system where the lock was requested (4 bytes)

The values that $GETLKI can write into the LKIB_RQMODE, LKIB_GRMODE, and LKI$B_QUEUE items have symbolic names; these symbolic names specify the six lock modes and the three types of queue in which a lock can reside. The Description section describes these names.

Thus, the buffer specified by the buffer address field in the item descriptor will contain the eight items of information, repeated in sequence, for each lock.

The length of the information returned for each lock is returned in bits 16 to 30 of the longword specified by the return length address field in the item descriptor, while the total length of information returned for all locks is returned in bits 0 to 15. Therefore, to determine the number of locks, you divide the value of bits 16 to 30 into the value of bits 0 to 15.

LKI$_MSTCSID

When you specify LKI$_MSTCSID, $GETLKI returns the Cluster System ID (CSID) of the node currently mastering the resource that is associated with the specified lock. Although the resource can be locked by processes on any node in the cluster, the resource itself is maintained on a single node. You can use the DCL command SHOW CLUSTER or the $GETSYI service to determine which node in the OpenVMS Cluster is identified by the CSID that $GETLKI returns.

Because the processor mastering the lock can change at any time, multiple calls to $GETLKI for the same lock can produce different values for this item code. LKI$_MSTCSID returns the CSID of the node where the $GETLKI system service is issued when the resource is mastered on that node. When the processor where the $GETLKI was issued is not part of an OpenVMS Cluster, this item code returns 0.

The buffer length field in the item descriptor should specify 4 (bytes).

LKI$_MSTLKID

When you specify LKI$_MSTLKID, $GETLKI returns the lock ID for the current master copy of the lock. Although the resource can be locked by processes on any node in the cluster, the resource itself is maintained on a single node. Because lock IDs are unique to each processor on a cluster, the lock ID returned by this item code has meaning only on the processor that is specified in the value returned by the LKI$_MSTCSID item code.

Because the processor mastering the lock can change at any time, multiple calls to $GETLKI for the same lock can produce different values for this item code. When the lock is mastered on the node where the $GETLKI system service is issued, or when the node is not a member of a cluster, this item code returns the same information as LKI$_LKID.

The buffer length field in the item descriptor should specify 4 (bytes).

LKI$_NAMSPACE

When you specify LKI$_NAMSPACE, $GETLKI returns information about the resource name space. This information is contained in a longword consisting of four bit fields; therefore, the buffer length field in the item descriptor should specify 4 (bytes).

Each of the four bit fields can be referred to by its symbolic name; the $LKIDEF macro defines the symbolic names. The following table lists, in order, the symbolic name of each bit field.

Symbolic Name	Description
LKI$W_GROUP	In this field (bits 0 to 15) $GETLKI writes the UIC group number of the process that took out the first lock on the resource, thereby creating the

Symbolic Name	Description
	resource name. This process issued a call to $ENQ or $ENQW specifying the name of the resource in the *resnam* argument. However, if this process specified the LCK$_SYSTEM flag in the call to $ENQ or $ENQW, the resource name is systemwide. In this case, the UIC group number of the process is not associated with the resource name. Consequently, this field (bits 0 to 15) is significant only if the resource name is not systemwide. $GETLKI sets bit 31 if the resource name is systemwide.
LKI$B_RMOD	In this field (bits 16 to 23), $GETLKI writes the access mode associated with the first lock taken out on the resource.
LKI$B_STATUS	This field (bits 24 to 30) is not used. $GETLKI sets it to 0.
LKI$V_SYSNAM	This field (bit 31) indicates whether the resource name is systemwide. $GETLKI sets this bit if the resource name is systemwide and clears it if the resource name is qualified by the creating process's UIC group number. The state of this bit determines the interpretation of bits 0 to 15.

LKI$_PARENT

When you specify LKI$_PARENT, $GETLKI returns the lock ID of the parent lock for the lock, if a parent lock was specified in the call to $ENQ or $ENQW. If the lock does not have a parent lock, $GETLKI returns the value 0.

Because the parent lock ID is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

LKI$_PID

When you specify LKI$_PID, $GETLKI returns the process identification (process ID) of the process that owns the lock.

The process ID is a longword value, so the buffer length field in the item descriptor should specify 4 (bytes).

LKI$_RESNAM

When you specify LKI$_RESNAM, $GETLKI returns the resource name string and its length, which must be from 1 to 31 bytes. The resource name string was specified in the *resnam* argument in the initial call to $ENQ or $ENQW.

The $GETLKI service returns the length of the string in the return length address field in the item descriptor. However, in the call to $GETLKI, you do not know how long the string is. Therefore, to avoid buffer overflow, you should specify the maximum length (31 bytes) in the buffer length field in the item descriptor.

LKI$_RSBREFCNT

When you specify LKI$_RSBREFCNT, $GETLKI returns the number of subresources of the resource associated with the lock. A subresource has the resource as a parent resource. Note, however, that the number of subresources can differ from the number of sublocks of the lock, because any number of processes can lock the resource. If any of these processes then locks another resource, and in doing so specifies the lock ID of the lock on the first resource as a parent lock, then the second resource becomes a subresource of the first resource.

Thus, the number of sublocks on a lock is limited to the number of sublocks that a single process takes out, whereas the number of subresources on a resource is determined by (potentially) multiple processes.

The subresource reference count is a longword value, so the buffer length field in the item descriptor should specify 4 (bytes).

LKI$_STATE

When you specify LKI$_STATE, $GETLKI returns the current state of the lock. The current state of the lock is described by the following three 1-byte items (in the order specified): (1) the lock mode requested (in the call to $ENQ or $ENQW) for the lock, (2) the lock mode granted (by $ENQ or $ENQW) for the lock, and (3) the name of the queue on which the lock currently resides.

The *buffer length* field in the item descriptor should specify 3 (bytes). The $LKIDEF macro defines the following symbolic names that refer to the three 1-byte items in the buffer.

Symbolic Name	Description
LKI$B_STATE_RQMODE	Lock mode requested
LKI$B_STATE_GRMODE	Lock mode granted
LKI$B_STATE_QUEUE	Name of queue on which the lock resides

The values that $GETLKI can write into each 1-byte item have symbolic names; these symbolic names specify the six lock modes and the three types of queue in which a lock can reside. The Description section describes these names.

LKI$_VALBLK

When you specify LKI$_VALBLK, $GETLKI returns the lock value block of the locked resource. This lock value block is the master copy that the lock manager maintains for the resource, not the process-private copy.

Because the lock value block is 16 bytes, the buffer length field in the item descriptor should specify 16.

LKI$_XVALBLK - Alpha and Integrity servers

Returns the entire 64 bytes of the lock value block of the locked resource. This lock value block is the master copy that the lock manager maintains for the resource, not the process-private copy.

The buffer length field in the item descriptor should specify 64.

LKI$_XVALNOTVALID - Alpha and Integrity servers

Returns a longword value of either zero or one:

- Zero (0) indicates that the extended value block is valid.
- One (1) indicates that the previous writer did not specify the LCK$M_XVALBLK flag and wrote only the first 16 bytes.

The buffer length field in the item descriptor should specify 4 (bytes).

LKI$_WAITCOUNT

When you specify LKI$_WAITCOUNT, $GETLKI returns the total number of locks that are currently on the wait queue of the resource associated with the lock. These locks are waiting to be granted.

The buffer length field in the item descriptor should specify 4 (bytes).

Description

The Get Lock Information service returns information about the lock database on a system.

The access mode of the calling process must be equal to or more privileged than the access mode at which the lock was initially granted.

When locking on a resource is clusterwide, a single master copy of the resource is maintained on the node that owns the process that created the resource by taking out the first lock on it. When a process on another node locks that same resource, a local copy of the resource is copied to the node and the lock is identified by a lock ID that is unique to that node.

In a cluster environment, however, you cannot use $GETLKI to obtain directly information about locks on other nodes in the cluster; that is, you cannot specify in a call to $GETLKI the lock ID of a lock held by a process on another node. The $GETLKI service interprets the *lkidadr* argument as the lock ID of a lock on the caller's node, even though the resource associated with a lock might have its master copy on the caller's node.

However, because a process on another node in the cluster can have a lock on the same resource as the caller of $GETLKI, the caller, in obtaining information about the resource, can indirectly obtain some information about locks on the resource that are held by processes on other nodes. One example of information indirectly obtained about a resource is the contents of lock queues; these queues contain information about all locks on the resource, and some of these locks can be held by processes on other nodes.

Another example of information more directly obtained is the remote lock ID of a lock held by a process on another node. Specifically, if the caller of $GETLKI on node A specifies a lock (by means of *lkidadr*) and that lock is held by a process on node B, $GETLKI returns the lock ID of the lock from node B's lock database if the LKI$_REMLKID item code is specified in the call.

Item codes LKI$_BLOCKEDBY, LKI$_BLOCKING, LKI$_LOCKS, and LKI$_STATE specify that $GETLKI return various items of information; some of these items are the names of lock modes or the names of lock queues. The $LCKDEF macro defines the following symbolic names.

Symbolic Name	**Lock Mode**
LCK$K_NLMODE	Null mode
LCK$K_CRMODE	Concurrent read mode
LCK$K_CWMODE	Concurrent write mode
LCK$K_PRMODE	Protected read mode
LCK$K_PWMODE	Protected write mode
LCK$K_EXMODE	Exclusive mode

Symbolic Name	**Queue Name**
LKI$C_GRANTED	Granted queue, holding locks that have been granted
LKI$C_CONVERT	Converting queue, holding locks that are currently being converted to another lock mode
LKI$C_WAITING	Waiting queue, holding locks that are neither granted nor converting (for example, a blocked lock)

Required Access or Privileges

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

- For locks held by other processes, you need to have joined the resource domain for lock access or hold WORLD privileges.

  You need WORLD privilege to obtain information about locks held by processes in other groups.

- To obtain information about system locks, either you need SYSLCK privilege or the process must be executing in executive or kernel access mode.

To establish a default resource domain, it is necessary to have performed either a call to $SET_RESOURCE_DOMAIN or a previous call to $ENQ[W].

Required Quota

The caller must have sufficient ASTLM or BYTLM quota.

Related Services

$DEQ, $ENQ, $ENQW, $GETLKIW, $SET_RESOURCE_DOMAIN

Condition Values Returned

SS$_NORMAL

The service completed successfully.

SS$_ACCVIO

The item list cannot be read; the areas specified by the buffer address and return length address fields in the item descriptor cannot be written; or the location specified by the `lkidadr` argument cannot be written.

SS$_BADPARAM

You specified an invalid item code.

SS$_EXQUOTA

The caller has insufficient ASTLM or BYTLM quota.

SS$_ILLRSDM

The operation attempted is not allowed on the resource. Use SHOW SECURITY to verify the access allowed to the specified resource domain.

SS$_INSFMEM

The nonpaged dynamic memory is insufficient for the operation.

SS$_IVLOCKID

The `lkidadr` argument specified an invalid lock ID.

SS$_IVMODE

A more privileged access mode is required.

SS$_NOMORELOCK

The caller requested a wildcard operation by specifying a value of 0 or –1 for the *lkidadr* argument, and $GETLKI has exhausted the locks about which it can return information to the caller; or no *lkidadr* argument is specified. This is an alternate success status.

SS$_NOSYSLCK

The caller attempted to acquire information about a systemwide lock and did not have the required SYSLCK privilege.

SS$_NOWORLD

The caller attempted to acquire information about a lock held by a process in another group and did not have the required WORLD privilege.

Condition Values Returned in the I/O Status Block

Same as those returned in R0.

$GETLKIW

Get Lock Information and Wait — Returns information about the lock database on a system.

Format

```
SYS$GETLKIW
   [efn] ,lkidadr ,itmlst [,iosb] [,astadr] [,astprm] [,nullarg]
```

C Prototype

```
int sys$getlkiw
   (unsigned int efn, unsigned int *lkidadr, void *itmlst,
    struct _iosb *iosb, void (*astadr)(__unknown_params), int astprm,
    unsigned int reserved);
```

Description

The $GETLKIW service completes synchronously; that is, it returns to the caller with the requested information.

For asynchronous completion, use the Get Lock Information ($GETLKI) service; $GETLKI returns to the caller after queuing the information request, without waiting for the information to be returned.

In all other respects, $GETLKIW is identical to $GETLKI. For all other information about the $GETLKIW service, see the description of $GETLKI in this manual.

The $GETLKI, $GETLKIW, $ENQ, $ENQW, and $DEQ services together provide the user interface to the Lock Management facility. For additional information about lock management, see the descriptions of these other services.

$GETMSG

Get Message — Returns message text associated with a given message identification code into the caller's buffer. The message can be from the system message file or a user-defined message. On Alpha and Integrity server systems, this service accepts 64-bit addresses.

Format

```
SYS$GETMSG msgid ,msglen ,bufadr ,[flags] ,[outadr]
```

C Prototype

```
int sys$getmsg
   (unsigned int msgid, unsigned short int *msglen, void *bufadr,
    unsigned int flags, unsigned char outadr [4]);
```

Arguments

msgid

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

Identification of the message to be retrieved. The *msgid* argument is a longword value containing the message identification. Each message has a unique identification, contained in bits 3 through 27 of system longword condition values.

msglen

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

Length of the message string returned by $GETMSG. The *msglen* argument is the 32- or 64-bit address of a word into which $GETMSG writes this length.

bufadr

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

Buffer to receive the message string. The *bufadr* argument is the 32- or 64-bit address of a character string descriptor pointing to the buffer into which $GETMSG writes the message string. The maximum size of any message string is 256 bytes.

flags

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

Message components to be returned. The *flags* argument is a longword bit vector wherein a bit, when set, specifies that the message component is to be returned. The following table describes the significant bits.

Bit	Value	Description
0	1	Include text of message
	0	Do not include text of message
1	1	Include message identifier
	0	Do not include message identifier
2	1	Include severity indicator
	0	Do not include severity indicator
3	1	Include facility name
	0	Do not include facility name

If you omit this argument in a VAX MACRO or BLISS-32 service call, it defaults to a value of 15; that is, all flags are set and all components of the message are returned. If you omit this argument in a Fortran service call, it defaults to a value of 0; the value 0 causes $GETMSG to use the process default flags.

outadr

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

Optional information to be returned by $GETMSG. The *outadr* argument is the 32- or 64-bit address of a 4-byte array into which $GETMSG writes the following information.

Byte	Contents
0	Reserved
1	Count of FAO arguments associated with message
2	User-specified value in message, if any
3	Reserved

Description

The Get Message service locates and returns message text associated with a given message identification code into the caller's buffer. The message can be from the system message file or a user-defined message.

The operating system uses this service to retrieve messages based on unique message identifications and to prepare to output the messages.

The message identifications correspond to the symbolic names for condition values returned by system components; for example, SS$_code from system services, RMS$_code for RMS messages, and so on.

When you set all bits in the *flags* argument, $GETMSG returns a string in the following format:

```
facility-severity-ident, message-text
```

where:

facility	Identifies the component of the operating system
severity	Is the severity code (the low-order three bits of the condition value)
ident	Is the unique message identifier
message-text	Is the text of the message

For example, if you specify the MSGID=#SS$_DUPLNAM argument, the $GETMSG service returns the following string:

```
%SYSTEM-F-DUPLNAM, duplicate process name
```

You can define your own messages with the Message utility. See the *VSI OpenVMS System Management Utilities Reference Manual* for additional information.

The message text associated with a particular 32-bit message identification can be retrieved from one of several places. This service takes the following steps to locate the message text:

1. All message sections linked into the currently executing image are searched for the associated information.
2. If the information is not found, the process-permanent message file is searched. (You can specify the process-permanent message file by using the SET MESSAGE command.)
3. If the information is not found, the systemwide message file is searched.
4. If the information is not found, the SS$_MSGNOTFND condition value is returned in R0 and a message in the following form is returned to the caller's buffer:

   ```
   %facility-severity-NONAME, message=xxxxxxxx[hex], (facility=n, message=n[dec])
   ```

Required Access or Privileges

None

Required Quota

None

Related Services

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

Condition Values Returned

SS$_NORMAL

The service completed successfully.

SS$_BUFFEROVF

The service completed successfully. The string returned overflowed the buffer provided and has been truncated.

SS$_INSFARG

The call arguments are insufficient.

SS$_MSGNOTFND

The service completed successfully; however, the message code cannot be found, and a default message has been returned.

Example

```
#include <stdio.h>
#include <ssdef.h>
#include <stsdef.h>
#include <descrip.h>
#include <starlet.h>

int     status,                         /* Status of system calls */
        msg_flag = 0x0001,              /* Text only */
        msg_code = SS$_DUPLNAM;         /* Message code to retrieve */
short int
        outlen;                         /* Length of output string from $FAO */

char    out_buffer[256],                /* Buffer for $FAO output */
        msg_info[4];                    /* Buffer for message information */

$DESCRIPTOR(out_desc, out_buffer);      /* VMS Descriptor for out_buffer */

main()
{
    status = sys$getmsg(msg_code,       /* Error message number */
                        &outlen,        /* Length of retrived message */
                        &out_desc,      /* Descriptor for output buffer */
                        msg_flag,       /* Message options flag */
                        msg_info);      /* Return information area */

    if ((status & STS$M_SUCCESS) != 0)
    {
        /* $GETMSG directive succeeded, output resultant string */
        out_buffer[outlen] = '\0';      /* add string terminator to buffer */
        puts(out_buffer);               /* output the result */
    }
    return (status);
}
```

This example shows a segment of a program used to obtain only the text portion of the message associated with the system message code SS$_DUPLNAM. The $GETMSG service returns the following string:

```
duplicate process name
```

$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, see the Synchronize ($SYNCH) service.

Format

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

C Prototype

```
int sys$getqui
   (unsigned int efn, unsigned short int func, unsigned int *context,
    void *itmlst, struct _iosb *iosb, void (*astadr)(__unknown_params),
    int 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).

VSI strongly recommends the use of the EFN$C_ENF "no event flag" value as the event flag if you are not using an event flag to externally synchronize with the completion of this system service call. The $EFNDEF macro defines EFN$C_ENF. For more information, see the *VSI OpenVMS Programming Concepts Manual*.

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).

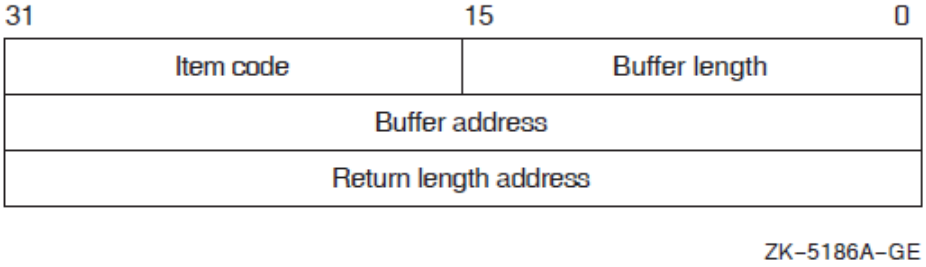
To generate a new context stream, the specified longword must contain –1. $GETQUI then modifies the longword to hold the context number for that stream of operation. The context is marked with the caller's mode (user, supervisor, executive, or kernel). Any attempt to use that context in successive calls 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.



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

Descriptor Field	Definition
	buffer varies, depending on the item code specified, and is given in the description of each item code.
Item code	A word containing an item code, which identifies the nature of the information supplied for $GETQUI or 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, VSI 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 might 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$_RAD
QUI$_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$_QUEUE_NAME
QUI$_RAD
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 name for a logical queue to the execution name for the queue. The name for the logical queue 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.

This item applies to logical queues only. For additional information, see the Description section.

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 can be followed by the optional colon [:].

For more information on the autostart feature, see the *VSI 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 the vector of three longwords. The first longword gives the completion status of the specified job. The second and third longwords give additional status about the print 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 *VSI 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 *VSI 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 *VSI 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 *VSI 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 *VSI 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 might 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 *VSI 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$V_JOB_LOWERCASE	Job is to be printed on printer that can print both uppercase and lowercase letters.
QUI$V_JOB_NOTIFY	Message is broadcast to terminal when job completes or aborts.
QUI$V_JOB_RAD	Indicates if the job uses the RAD processing option. RAD is supported on AlphaServer GS series systems and starting from OpenVMS Version 8.4, support is extended to NUMA capable Integrity servers.
QUI$V_JOB_REQUEUE	Job will requeue after a system failure or can be requeued during execution.
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 *VSI 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.

Symbolic Name	Description
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 can be expressed in either an absolute or delta time format.

For more information, see the /RETAIN qualifier for PRINT or SUBMIT in the *VSI 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$_JOB_STATUS
QUI$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.

Symbol Name	Description
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.

Symbolic Name	Description
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

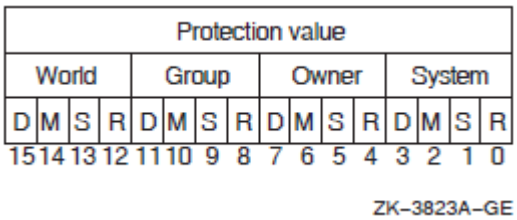
When you specify QUI$_PROCESSOR, $GETQUI returns, as an OpenVMS RMS file name component, the name of the symbiont image that executes print jobs initiated from the specified queue. The file name assumes the device and directory name SYS$SYSTEM and the file type .EXE. Because an RMS file name can include 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.



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 $GET_SECURITY service.
QUI$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$V_QUEUE_CPU_LIMIT	A maximum CPU time limit has been specified for all jobs in the queue.
QUI$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.

Symbolic Name	Description
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_RAD (Alpha and Integrity servers)	Indicates if the RAD processing option attribute is specified for the batch queue. RAD is supported on AlphaServer GS series systems and starting from OpenVMS Version 8.4, support is extended to NUMA capable Integrity servers.
QUI$V_QUEUE_RECORD_ BLOCKING	The symbiont is permitted to concatenate, or block together, the output records it sends to the output device.
QUI$V_QUEUE_RETAIN_ALL	All jobs initiated from the queue remain in the queue after they finish executing. Completed jobs are marked with a completion status.
QUI$V_QUEUE_RETAIN_ERROR	Only jobs that do not complete successfully are retained in the queue.
QUI$V_QUEUE_SECURITY_ INACCESSIBLE	The requestor does not have access to security information.
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$_RAD (Alpha and Integrity servers)

When you specify QUI$_RAD, $GETQUI returns the value of the RAD attribute attached to a queue or a job. The item code expects a 32-bit buffer in which to write the value of the RAD. A value of -1 is returned to the buffer if no RAD is specified for the queue or the job.

RAD is supported on AlphaServer GS series systems and starting from OpenVMS Version 8.4, support is extended to NUMA capable Integrity servers.

(Valid for QUI$_DISPLAY_ENTRY, QUI$_DISPLAY_JOB, 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$V_QUEUE_IDLE[1]	Queue contains no job requests capable of being processed.
QUI$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	Does not advance wildcard context on completion of this service call.

Symbolic Name	Function Code	Description
	QUI$_DISPLAY_QUEUE QUI$_DISPLAY_MANAGER	
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$_DISPLAY_QUEUE	Selects server queues.
QUI$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.

Classes and Types of Queues

The $GETQUI item codes and function codes can be specific to the type and class of the queue. For example, QUI$_LAST_PAGE applies only in the context of a printer or terminal execution queue, and QUI$_LOG_SPECIFICATION applies only to batch files.

The following information shows the two classes and the various types of queues, while the descriptions of the individual $GETQUI item codes and function codes indicate the associated queue class and queue type requirements:

- **Execution queues**—Queues that accept batch or print jobs for processing.
- **Generic queues**—Queues that hold jobs until an appropriate execution queue becomes available. The queue manager then requeues the job to the available execution queue.

The following table lists the Execution queue types:

Type	Description
Batch execution queue	Batch execution queues accept only batch jobs.
Output execution queues	Output execution queues accept jobs that a symbiont processes, and include the following types: • Printer execution queue—Uses a symbiont to direct output to a printer. • Terminal execution queue—Uses a symbiont to direct output to a terminal printer. • Server execution queue—Uses a user-modified or user-written symbiont to process the files that belong to jobs in the queue.

The following table lists the Generic queue types:

Type	Description
Generic batch queues	Generic batch queues direct jobs only to batch execution queues.
Generic output queues	Generic output queues direct jobs to any of the three types of output execution queues: print, terminal, or server.
Logical queues	Logical queues are a special type of generic output queue that transfers jobs to another output execution queue.

For additional information, see the *VSI OpenVMS System Manager's Manual, Volume 1: Essentials*.

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 form) function code; specify a name in the QUI$_SEARCH_NAME item code that includes one or more wildcard characters (* or %). QUI$_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.

o 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.

Format

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

C Prototype

```
int sys$getquiw
   (unsigned int efn, unsigned short int func, unsigned int *context,
    void *itmlst, struct _iosb *iosb, void (*astadr)(__unknown_params),
    int astprm);
```

Description

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, see the description of $GETQUI in this manual.

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

$GETRMI

Get Resource Monitor Information — Returns system performance information about the local system. $GETRMI is an asynchronous system service and requires the $SYNCH service or another wait-state synchronous mechanism to guarantee that the required information is available. There is no synchronous wait form for this system service. For additional information about system service completion, see the Synchronize ($SYNCH) service.

Format

```
SYS$GETRMI [efn] [,nullarg] [,nullarg] ,itmlst [,iosb] [,astadr] [,astprm]
```

C Prototype

```
nt sys$getrmi
   (unsigned int efn, unsigned int nullarg, unsigned int nullarg,
    void *itmlst, struct _iosb *iosb, void (*astadr)(__unknown_params),
    int astprm);
```

Arguments

efn

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

Number of event flag to be set when the $GETRMI request completes. The *efn* argument is a longword containing this number; however, $GETRMI uses only the low order byte.

nullarg

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

Placeholding argument reserved to OpenVMS.

nullarg

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

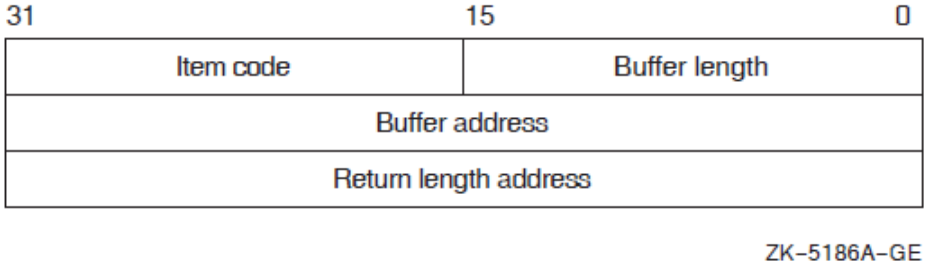
Placeholding argument reserved to OpenVMS.

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 local node. The *itmlst* argument is the address of a list of item descriptors, each of which describes an item of information. The list of descriptors is terminated by a longword of 0.

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



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 $GETRMI is to write the information. The length of the buffer needed depends upon the item code specified in the item code field. If the buffer length is too small, $GETRMI truncates the data.
Item code	A word containing a user-supplied code specifying the item of information that $GETRMI is to return. The RMIDEF 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 a buffer in which $GETRMI returns the requested information.
Return length address	A longword containing the user-supplied address of a word in which $GETRMI writes the length in bytes of the information returned.

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.

astadr

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

mechanism:	by reference

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

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.

Item Codes

RMI$_ACCESS

Returns the count of file name lookup operations in file directories.

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

RMI$_ACCLCK

Returns the systemwide count of access locks.

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

RMI$_ALLOC

Returns the number of QIO requests that caused allocation of disk space.

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

RMI$_ARRLOCPK

Returns the accumulated systemwide count of arriving local DECnet packets.

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

RMI$_ARRTRAPK

Returns the accumulated systemwide count of arriving transit DECnet packets.

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

RMI$_BADFLTS

Returns the accumulated systemwide count of bad-list faults.

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

RMI$_BLKIN

Returns the accumulated systemwide count of blocking ASTs queued that originated on a remote system and were processed on the local system.

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

RMI$_BLKLOC

Returns the accumulated systemwide count of blocking ASTs queued that originated and were processed on the local system.

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

RMI$_BLKOUT

Returns the accumulated systemwide count of blocking ASTs queued that originated on the local system and were processed on a remote system.

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

RMI$_BLKAST

Returns the number of blocking ASTs.

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

RMI$_BUFIO

Returns the number of buffered I/Os.

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

RMI$_BUFOBJPAG

Returns the number of buffer object physical pages currently allocated.

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

RMI$_BUFOBJPAGPEAK

Returns the maximum number of buffer object physical pages currently allocated.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_BUFOBJPAGS01

Returns the number of buffer object pages currently allocated in S0 and S1 space.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_BUFOBJPAGS2

Returns the number of buffer object physical pages currently allocated in S2 space.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_BUFOBJPAGMAXS01

Returns the available number of buffer object pages in S0 and S1 space.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_BUFOBJPAGMAXS2

Returns the available number of buffer object pages in S2 space.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_BUFOBJPAGPEAKS01

Returns the maximum number of buffer object pages currently allocated in S0 and S1 space.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_BUFOBJPAGPEAKS2

Returns the maximum number of buffer object pages currently allocated in S2 space.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_BUFOBJPGLTMAXS01

Returns the available number of buffer object pagelets in S0 and S1 space.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_BUFOBJPGLTMAXS2

Returns the available number of buffer object pagelets in S2 space.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_CEF

Returns the number of processes in the common event flag wait state.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_COLPG

Returns the number of processes in the collided page wait state.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_COM

Returns the number of processes in the computable state.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_COMO

Returns the number of outswapped processes in the computable state.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_CPUEXEC

Returns the amount of time, in 10-millisecond units, spent by all CPUs in executive mode.

Because this number is a quadword, the buffer length field in the item descriptor should specify 8 (bytes).

RMI$_CPUID

Returns the primary CPU ID.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_CPUIDLE

Returns the amount of time, in 10-millisecond units, spent by all CPUs in idle mode.

Because this number is a quadword, the buffer length field in the item descriptor should specify 8 (bytes).

RMI$_CPUINTSTK

Returns the amount of time, in 10-millisecond units, spent by all CPUs in processing interrupts.

Because this number is a quadword, the buffer length field in the item descriptor should specify 8 (bytes).

RMI$_CPUKERNEL

Returns the amount of time, in 10-millisecond units, spent by all CPUs in kernel mode.

Because this number is a quadword, the buffer length field in the item descriptor should specify 8 (bytes).

RMI$_CPUMPSYNCH

Returns the amount of time, in 10-millisecond units, spent by the primary CPU in synchronization mode.

Because this number is a quadword, the buffer length field in the item descriptor should specify 8 (bytes).

RMI$_CPUSUPER

Returns the amount of time, in 10-millisecond units, spent by all CPUs in supervisor mode.

Because this number is a quadword, the buffer length field in the item descriptor should specify 8 (bytes).

RMI$_CPUUSER

Returns the amount of time, in 10-millisecond units, spent by all CPUs in user mode.

Because this number is a quadword, the buffer length field in the item descriptor should specify 8 (bytes).

RMI$_CUR

Returns the number of currently-executing processes.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_CWPSBYTESIN

Returns the number of clusterwide process services (CWPS) message bytes received by the system.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_CWPSBYTESOUT

Returns the number of CWPS message bytes sent by the system.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_CWPSJPISIN

Returns the number of CWPS $GETJPI requests received by the system.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_CWPSJPISOUT

Returns the number of CWPS $GETJPI requests sent by the system.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_CWPSMSGSIN

Returns the number of CWPS messages received by the system.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_CWPSMSGSOUT

Returns the number of CWPS messages sent by the system.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_CWPSPCNTRLIN

Returns the number of CWPS PCNTRL requests received by the system.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_CWPSPCNTRLOUT

Returns the number of CWPS PCNTRL requests sent by the system.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_CWPSRSRCIN

Returns the number of CWPS resource-fail messages received by the system.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_CWPSRSRCOUT

Returns the number of CWPS resource-fail messages sent by the system.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_DDTM_ABORTS

Returns the accumulated systemwide count of transactions aborted (planned and unplanned).

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_DDTM_ADDS

Returns the accumulated systemwide count of transaction branches added on the local node.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_DDTM_BAD_LINKS

Returns the total number of bad message links received.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_DDTM_BAD_PARTS

Returns the number of invalid part IDs found.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_DDTM_BAD_TYPECODE

Returns the total number of bad message type codes received.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_DDTM_BRANCHS

Returns the accumulated systemwide count of transaction branches started on the local node.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_DDTM_BUCKETS1

Returns the accumulated systemwide count of transactions with a duration of less than 1 second.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_DDTM_BUCKETS2

Returns the accumulated systemwide count of transactions with a duration of 1 to 2 (1.99) seconds.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_DDTM_BUCKETS3

Returns the accumulated systemwide count of transactions with a duration of 2 to 3 (2.99) seconds.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_DDTM_BUCKETS4

Returns the accumulated systemwide count of transactions with a duration of 3 to 4 (3.99) seconds.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_DDTM_BUCKETS5

Returns the accumulated systemwide count of transactions with a duration of 4 to 5 (4.99) seconds.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_DDTM_BUCKETS6

Returns the accumulated systemwide count of transactions with a duration of at least 5 seconds.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_DDTM_DECLARES

Returns the total number of $DECLARE_RMs.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_DDTM_DISC_COMP

Returns the number of disconnected complete events.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_DDTM_ENDS

Returns the accumulated systemwide count of transactions ended.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_DDTM_FOR_UNLINKS

Returns the number of forced unlinks.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_DDTM_FORGETS

Returns the total number of $FORGET_RMs.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_DDTM_JOINS

Returns the total number of $JOIN_RMs.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_DDTM_LOG_COMMITS

Returns the total number of commit records written.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_DDTM_LOG_FORGETS

Returns the total number of forget records written.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_DDTM_LOG_PREPARES

Returns the total number of prepare records written.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_DDTM_ONE_PHASE

Returns the accumulated systemwide count of 1-phase commit events initiated.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_DDTM_PREPARES

Returns the accumulated systemwide count of transactions that have been prepared.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_DDTM_SEQNO

Returns the total number of XCBs created.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_DDTM_STARTS

Returns the accumulated systemwide count of transactions successfully started.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_DDTM_TWOPHASE_ACKRCV

Returns the accumulated systemwide count of 2-phase commit ACK messages received.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_DDTM_TWOPHASE_ACKSNT

Returns the accumulated systemwide count of 2-phase commit ACK messages sent.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_DDTM_TWOPHASE_CANRCV

Returns the accumulated systemwide count of 2-phase commit cancel messages received.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_DDTM_TWOPHASE_CANSNT

Returns the accumulated systemwide count of 2-phase commit cancel messages sent.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_DDTM_TWOPHASE_COMMITS

Returns the accumulated systemwide count of 2-phase commit events initiated.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_DDTM_TWOPHASE_RDYRCV

Returns the accumulated systemwide count of 2-phase commit ready messages received.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_DDTM_TWOPHASE_RDYSNT

Returns the accumulated systemwide count of 2-phase commit ready messages sent.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_DDTM_TWOPHASE_REQRCV

Returns the accumulated systemwide count of 2-phase commit requests received.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_DDTM_TWOPHASE_REQSNT

Returns the accumulated systemwide count of 2-phase commit requests sent.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_DDTM_VOL_UNLINKS

Returns the number of voluntary unlinks.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_DDTM_WRITES_FORKED

Returns the total number of forked writes.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_DDTM_WRITES_STARTED

Returns the total number of writes started.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_DEPLOCPK

Returns the accumulated systemwide count of departing local DECnet packets.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_DEQ

Returns the number of DEQ operations.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_DEQIN

Returns the accumulated systemwide count of unlock (dequeue) lock requests that originated on a remote system and were processed on the local system.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_DEQLOC

Returns the accumulated systemwide count of unlock (dequeue) requests that originated and were processed on the local system.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_DEQOUT

Returns the accumulated systemwide count of unlock (dequeue) requests that originated on the local system and were processed on a remote system.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_DIRDATA_HIT

Returns the systemwide count of directory data cache hits.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_DIRDATA_MISS

Returns the systemwide count of directory data cache misses.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_DIRFCB_HIT

Returns the systemwide count of directory FCB cache hits.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_DIRFCB_MISS

Returns the systemwide count of directory FCB cache misses.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_DIRIN

Returns the accumulated systemwide count of directory operations serviced by the local system that originated on remote systems.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_DIROUT

Returns the accumulated systemwide count of directory operations that originated on the local system and were serviced by remote systems.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_DIRIO

Returns the number of direct I/Os.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_DLCK_INCMPLT

Returns the accumulated systemwide count of incomplete deadlock searches.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_DLCKFND

Returns the number of deadlocks found.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_DLCKMSGS_IN

Returns the systemwide count of incoming deadlock detection messages.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_DLCKMSGS_OUT

Returns the accumulated systemwide count of outgoing deadlock detection messages.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_DLCKSRCH

Returns the number of deadlock searches.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_DZROFLTS

Returns the number of demand zero page faults.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_ENQCVT

Returns the number of ENQ conversion operations.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_ENQCVTIN

Returns the accumulated systemwide count of lock conversion requests that originated on a remote system and were processed on the local system.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_ENQCVTLOC

Returns the accumulated systemwide count of lock conversion requests that originated and were processed on the local system.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_ENQCVTOUT

Returns the accumulated systemwide count of lock conversion requests that originated on the local system and were processed on a remote system.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_ENQNEW

Returns the number of new ENQ operations.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_ENQNEWIN

Returns the accumulated systemwide count of new lock requests that originated on a remote system and were processed on the local system.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_ENQNEWLOC

Returns the accumulated systemwide count of new lock requests that originated and were processed on the local system.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_ENQNEWOUT

Returns the accumulated systemwide count of new lock requests that originated on the local system and were processed on a remote system.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_ENQNOTQD

Returns the number of ENQ operations not queued.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_ENQWAIT

Returns the number of ENQ operations forced to wait.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_EXEFAULTS

Returns the number of execute page faults.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_EXTHIT

Returns the systemwide count of extent cache hits.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_EXTMISS

Returns the systemwide count of extent cache misses.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_FAULTS

Returns the number of page faults since last system initialization.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_FCPCACHE

Returns the total number of cache hits by the FCP.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_FCPCALLS

Returns the total number of calls to the FCP.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_FCPCPU

Returns the total number of CPU tics used by the FCP.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_FCPCREATE

Returns the number of new files created since the system was booted.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_FCPERASE

Returns the number of erase I/O operations issued.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_FCPFAULT

Returns the number of FCP page faults.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_FCPHIT

Returns the total number of file I/O transfers for which no disk access was required.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_FCPREAD

Returns the total number of disk reads by the FCP.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_FCPSPLIT

Returns the number of split transfers performed by the FCP.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_FCPTURN

Returns the number of file-map window misses.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_FCPWRITE

Returns the total number of disk writes by the FCP.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_FIDHIT

Returns the systemwide count of File ID cache hits.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_FIDMISS

Returns the systemwide count of File ID cache misses.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_FILHDR_HIT

Returns the systemwide count of file header cache hits.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_FILHDR_MISS

Returns the systemwide count of file header cache misses.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_FPG

Returns the number of processes in the free page wait state.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_FREFLTS

Returns the number of page faults from the free list.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_FRLIST

Returns the number of pages on the free list.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes) on Alpha systems and 8 (bytes) on Integrity server systems.

RMI$_GBP_CURMAP

(Alpha and Integrity servers) Returns the count of global pages currently mapped.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_GBP_CURMAP_GRP

(Alpha and Integrity servers) Returns the count of group global pages currently mapped.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_GBP_CURMAP_GRPWRT

(Alpha and Integrity servers) Returns the count of writable group global pages currently mapped.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_GBP_CURMAP_SYS

(Alpha and Integrity servers) Returns the count of system global pages currently mapped.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_GBP_CURMAP_SYSWRT

(Alpha and Integrity servers) Returns the count of writable system global pages currently mapped.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_GBP_MAXMAP

(Alpha and Integrity servers) Returns the maximum count of global pages simultaneously mapped.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_GBS_CURMAP

(Alpha and Integrity servers) Returns the count of global sections currently mapped.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_GBS_CURMAP_GRP

(Alpha and Integrity servers) Returns the count of group global sections currently mapped.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_GBS_CURMAP_GRPWRT

(Alpha and Integrity servers) Returns the count of writable group global sections currently mapped.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_GBS_CURMAP_SYS

(Alpha and Integrity servers) Returns the count of system global sections currently mapped.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_GBS_CURMAP_SYSWRT

(Alpha and Integrity servers) Returns the count of writable system global sections currently mapped.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_GBS_MAXMAP

(Alpha and Integrity servers) Returns the maximum count of global sections simultaneously mapped.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_GBS_NOREF

(Alpha and Integrity servers) Returns the current count of global sections not mapped to a process (reference count is 0).

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_GVALFLTS

Returns the number of global valid page faults.

Because this number is a longword the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_HDRINSWAPS

Returns the accumulated systemwide count of process header inswap operations.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_HDROUTSWAPS

Returns the accumulated systemwide count of process header outswap operations.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_HIB

Returns the number of processes in the hibernate state.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_HIBO

Returns the number of outswapped processes in the hibernate state.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_IOPAGCNT

Returns the systemwide count of pages in transit to disk from the modified page list.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_ISWPCNT

Returns the number of process inswaps.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_ISWPCNTPG

Returns the accumulated systemwide count of pages inswapped.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_LCKMGR_CPU

Returns the ID of the CPU on which the lock manager process runs.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_LCKMGR_PID

Returns the PID of the lock manager process.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_LCKMGR_REQCNT

Returns the accumulated count of requests handled by the lock manager.

Because this number is a quadword, the buffer length field in the item descriptor should specify 8 (bytes).

RMI$_LCKMGR_REQTIME

Returns the accumulated time spent by the lock manager servicing requests.

Because this number is a quadword, the buffer length field in the item descriptor should specify 8 (bytes).

RMI$_LCKMGR_SPINCNT

Returns the accumulated count of times the lock manager entered a spin loop.

Because this number is a quadword, the buffer length field in the item descriptor should specify 8 (bytes).

RMI$_LCKMGR_SPINTIME

Returns the accumulated spin time, in cycles, of the lock manager.

Because this number is a quadword, the buffer length field in the item descriptor should specify 8 (bytes).

RMI$_LEF

Returns the number of processes in the local event flag wait state.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_LEFO

Returns the number of outswapped processes in the local event flag wait state.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_LOCK_MAX

Returns the lock ID table length.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_LOGNAM

Returns the number of logical name translations.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_LPZ_ALLOC2

Returns the number of allocations from other than the first page of the lock manager's pool zone.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_LPZ_ALLOCF

Returns the number of failed allocations from the lock manager's pool zone.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_LPZ_EMPTY

Returns the number of empty pages in the lock manager's pool zone.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_LPZ_EXPCNT

Returns the accumulated number of expansions of the lock manager's pool zone.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_LPZ_HITS

Returns the number of hits for the lock manager's pool zone.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_LPZ_MAXPAG

Returns the maximum number of pages in the lock manager's pool zone.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_LPZ_MISSES

Returns the number of misses for the lock manager's pool zone.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_LPZ_PAGCNT

Returns the number of pages currently in the lock manager's pool zone.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_LPZ_PAKSIZ

Returns the packet size for the lock manager's pool zone.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_MBREADS

Returns the number of mailbox reads.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_MBWRITES

Returns the number of mailbox writes.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_MCHKERRS

Returns the accumulated count of machine checks since the system was booted.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_MEMERRS

Returns the accumulated count of memory errors since the system was booted.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_MODES

Allows you to monitor the modes of each CPU in a multiple-CPU system (not just the CPU usage of all the nodes together).

RMI$_MODES returns the amount of time, in 10-ms units, spent by all currently active CPUs in all processor modes since the system was booted. Each increment in the returned time for a mode represents an additional 10 ms spent by the CPU in that mode. An active CPU is one that is actively participating in the processor scheduling that the OpenVMS instance performs.

The buffer length field in the item descriptor is dependent on the number of CPUs attached to the system. The size of the buffer passed should be as follows:

```
n * sizeof ( CPU_struct ) + 4
```

where *n* is the available CPU count. Use $GETSYI item code SYI$_POTENTIALCPU_CNT to get *n*, the count of potentially available CPUs. (For details, see the description of the $GETSYI service.)

Data for an individual CPU is returned by means of a CPU_struct structure. Declare the CPU_struct with no member alignment, as in the following example:

```
#pragma member_alignment save
#pragma _nomember_alignment
typedef struct _CPU_struct
{
    unsigned char   cpu_id;     // physical cpu id
    unsigned int    interrupt;  // is actually the sum of interrupt and idle1[1]
    unsigned int    mpsynch;    // multi-processor synchronization
    unsigned int    kernel;     // kernel mode
```

[1]For compatibility with existing applications, the returned value for interrupt time is the sum of interrupt and idle times. Calculate the actual interrupt time by subtracting the returned value of idle time from the returned value of interrupt time.

```
    unsigned int    exec;       // executive mode
    unsigned int    super;      // supervisor mode
    unsigned int    user;       // user mode
    unsigned int    reserved;   // reserved, will be zero
    unsigned int    idle;       // CPU idle
} CPU_struct;
#pragma member_alignment restore
```

For a multiple-CPU system, the data for all active CPUs is returned as an array of type CPU_struct, with the number of elements in the array equal to the number of potentially available CPUs. If a CPU is inactive, the CPU_struct array element corresponding to that CPU contains 0 in all CPU_struct members.

Be sure to allocate a buffer large enough to hold the counters for all available CPUs.

The following code example shows one method for collecting CPU modes:

```
#include <starlet.h>
#include <syidef.h>
#include <rmidef.h>
#include <efndef.h>
#include <iledef.h>
#include <iosbdef.h>

CPU_struct   *cpu_counters = NULL;
IOSB          myiosb = { 0 };
char         *buffer;
unsigned long buffer_size;
int           status;

// Set up the $GETSYI item list: potential CPUs and active CPU bitmask
// unsigned long
CPU_Count = 0;
unsigned long long ActiveCPUs = 0;

ILE3 SYIitmlst[] = { {sizeof CPU_Count, SYI$_POTENTIALCPU_CNT, &CPU_Count, 0},
                     {sizeof ActiveCPUs, SYI$_ACTIVE_CPU_MASK, &ActiveCPUs, 0},
                     {0,0}};

// Get the available CPU count and a bitmask of active CPUs
status = SYS$GETSYIW( EFN$C_ENF, NULL, NULL, SYIitmlst, &myiosb, NULL, 0 );
buffer_size = CPU_Count * sizeof( CPU_struct ) + 4;
buffer = malloc( buffer_size );

// Set up the $GETRMI item list: CPU modes
ILE3 RMIitmlst[] = {{buffer_size, RMI$_MODES, buffer, 0},
                    {0,0}};

// Call the service
status = SYS$GETRMI( EFN$C_ENF, 0, 0, RMIitmlst, &myiosb, NULL, 0 );
   ...
status = SYS$SYNCH
   ...
// THE DATA COUNTERS BEGIN 4 BYTES OFF THE START OF THE BUFFER;
// The first 4 bytes of the buffer are reserved for internal use.
cpu_counters = (CPU_struct *) ( buffer + 4 );

// Use the counters for all active CPUs
for ( int i = 0; i < CPU_Count; i++)
{
    if ( 1 == ( 1 & (ActiveCPUs >> i) ) )
    {
       cpu_counters[i].interrupt
       ---
       ---
```

```
        ---
    }
}

free( buffer );
```

RMI$_MODLIST

Returns the number of pages on the modified page list.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes) on Alpha systems and 8 (bytes) on Integrity server systems.

RMI$_MSCP_EVERYTHING

Returns all the performance data items in the following order:

MSCP_BUFAVL	Current number of free MSCP buffers
MSCP_BUFSMALL	Smallest MSCP buffer size allowed
MSCP_BUFWAITCUR	Number of requests currently queued waiting for MSCP buffer memory
MSCP_BUFWAITPEAK	Maximum number of requests simultaneously queued waiting for MSCP buffer
MSCP_DSKSRV	Number of MSCP served disks
MSCP_HSTSRV	Number of MSCP served hosts
MSCP_LB_FAILCNT	MSCP server's count of failed load-balancing requests
MSCP_LB_INITCNT	MSCP server's count of load-balancing requests sent
MSCP_LB_LMLOAD1	MSCP server's previous interval's load 1 value
MSCP_LB_LMLOAD2	MSCP server's previous interval's load 2 value
MSCP_LB_LMLOAD3	MSCP server's previous interval's load 3 value
MSCP_LB_LMLOAD4	MSCP server's previous interval's load 4 value
MSCP_LB_LOAD	MSCP server's target load for load-balancing requests
MSCP_LB_LOAD_AVAIL	MSCP server's current load available value
MSCP_LB_LOAD_CAP	MSCP server's load capacity value
MSCP_LB_MONINT	MSCP server's load-monitoring interval size
MSCP_LB_MONTIME	The time that the last load-balancing monitor pass was made
MSCP_LB_REQCNT	MSCP server's count of load-balancing requests received from other servers
MSCP_LB_REQTIME	The time that the last load-balancing request was sent
MSCP_LB_RESPCNT	MSCP server's count of load-balancing requests to which it responded
MSCP_LB_RESP	MSCP server's load available from another server
MSCP_OPCOUNT	Count of I/O transfer requests by remote processors
MSCP_VCFAIL	Count of virtual cache failures on MSCP-served requests
MSCP_READ	Count of Read I/O transfer requests by remote processors
MSCP_WRITE	Count of Write I/O transfer requests by remote processors

MSCP_FRAGMENT	Count of extra fragments issued by the MSCP server
MSCP_SPLITXFER	Count of fragmented requests issued by the MSCP server
MSCP_BUFWAIT	Count of requests that had to wait for MSCP buffer memory
MSCP_SIZE1	Count of MSCP-served I/O requests with a length of 1 block
MSCP_SIZE2	Count of MSCP-served I/O requests with a length of 2-3 blocks
MSCP_SIZE3	Count of MSCP-served I/O requests with a length of 4-7 blocks
MSCP_SIZE4	Count of MSCP-served I/O requests with a length of 8-15 blocks
MSCP_SIZE5	Count of MSCP-served I/O requests with a length of 16-31 blocks
MSCP_SIZE6	Count of MSCP-served I/O requests with a length of 32-63 blocks
MSCP_SIZE7	Count of MSCP-served I/O requests with a length of 64 or more blocks

Because this an array of 35 longwords, the buffer length field in the item descriptor should specify 4 times 35 (bytes).

RMI$_MWAIT

Returns the number of processes in the miscellaneous resource wait state.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_NP_POOL_ALLOC

Returns the accumulated count of nonpaged pool allocation requests.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_NP_POOL_ALLOCF

Returns the accumulated count of unsuccessful nonpaged pool allocation requests.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_NP_POOL_EXP

Returns the accumulated count of successful expansions of nonpaged pool.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_NP_POOL_EXPF

Returns the accumulated count of unsuccessful attempts to expand nonpaged pool.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_NUMLOCKS

Returns the total number of locks.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_NUMRES

Returns the total number of resources.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_OPENS

Returns the systemwide count of files opened.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_OSWPCNT

Returns the accumulated systemwide count of process outswap operations.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_OSWPCNTPG

Returns the accumulated systemwide count of pages outswapped.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_PFW

Returns the number of processes in the page fault wait state.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_PG_POOL_ALLOC

Returns the accumulated count of paged pool allocation requests.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_PG_POOL_ALLOCF

Returns the accumulated count of unsuccessful paged pool allocation requests.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_PG_POOL_EXPF

Returns the accumulated count of paged pool failures.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_PREADIO

Returns physical page read I/Os.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_PREADS

Returns the number of pages read.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_PROCBALSETCNT

Returns the number of processes in the balance set.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_PROCBATCNT

Returns the number of batch processes known to the system.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_PROCCNTMAX

Returns the maximum number of concurrent processes seen by the system.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_PROCINTCNT

Returns the number of interactive processes known to the system.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_PROCLOADCNT

Returns the accumulated systemwide count of process context load operations.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_PROCNETCNT

Returns the number of network processes known to the system.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_PROCS

Returns the number of processes currently known to the system.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_PROCSWITCHCNT

Returns the number of switches from the currently-executing process.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_PWRITES

Returns the number of pages written.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_PWRITIO

Returns physical page write I/Os.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_QUOHIT

Returns the systemwide count of quota cache hits.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_QUOMISS

Returns the systemwide count of quota cache misses.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_RCVBUFFL

Returns the accumulated systemwide count of DECnet receiver buffer failures.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_RDFAULTS

Returns the number of fault-on-read page faults.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_RML_ACQUIRE

Returns the accumulated systemwide count of lock trees moved to this node.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_RML_BETTER

Returns the accumulated systemwide count of lock trees moved from this node to a cluster node with a higher value for the system parameter LOCKDIRWT.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_RML_MORE_ACT

Returns the accumulated systemwide count of lock trees moved from this node due to higher locking activity on another node in the cluster.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_RML_MSGRCV

Returns the accumulated systemwide count of remaster messages received by this node.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_RML_MSGSENT

Returns the accumulated systemwide count of remaster messages sent from this node.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_RML_NOQUOTA

Returns the accumulated systemwide count of remaster operations that failed due to a lack of quota.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_RML_NOTAKER

Returns the accumulated systemwide count of remaster operations that were proposed and declined.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_RML_OPCNT

Returns the accumulated systemwide count of remaster operations that have been completed.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_RML_RBLDMSGRCV

Returns the accumulated systemwide count of remaster rebuild messages received by this node.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_RML_RBLDMSGSENT

Returns the accumulated systemwide count of remaster rebuild messages sent from this node.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_RML_SINGLE

Returns the accumulated systemwide count of lock trees moved from this node to another cluster node because that node is the only one with locks remaining on the tree.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_RML_UNLOAD

Returns the accumulated systemwide count of lock trees moved from this node.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_SMP_CURMAP

(Alpha and Integrity servers) Returns the count of global pages currently mapped for Galaxy shared memory.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_SMP_CURMAP_GRP

(Alpha and Integrity servers) Returns the count of group global pages currently mapped for Galaxy shared memory.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_SMP_CURMAP_GRPWRT

(Alpha Only) Returns the count of writable group global pages currently mapped for Galaxy shared memory.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_SMP_CURMAP_SYS

(Alpha Only Returns the count of system global pages currently mapped for Galaxy shared memory.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_SMP_CURMAP_SYSWRT

(Alpha Only) Returns the count of writable system global pages currently mapped for Galaxy shared memory.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_SMS_CURMAP

(Alpha Only) Returns the count of global sections currently mapped for Galaxy shared memory.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_SMS_CURMAP_GRP

(Alpha Only) Returns the count of group global sections currently mapped for Galaxy shared memory.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_SMS_CURMAP_GRPWRT

(Alpha Only) Returns the count of writable group global sections currently mapped for Galaxy shared memory.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_SMS_CURMAP_SYS

(Alpha Only) Returns the count of system global sections currently mapped for Galaxy shared memory.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_SMS_CURMAP_SYSWRT

(Alpha Only) Returns the count of writable system global sections currently mapped for Galaxy shared memory.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_SMS_NOREF

(Alpha Only) Returns the current count of global sections for Galaxy shared memory that are not mapped to a process (reference count is 0).

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_STORAGMAP_HIT

Returns the systemwide count of storage bitmap cache hits.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_STORAGMAP_MISS

Returns the systemwide count of storage bitmap cache misses.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_SUSP

Returns the number of processes in the suspended state.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_SUSPO

Returns the number of outswapped processes in the suspended state.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_SYNCHLCK

Returns the systemwide count of directory- or file-synch locks.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_SYNCHWAIT

Returns the systemwide count of times the XQP waited for a directory- or file-synch lock.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_SYSFAULTS

Returns the number of system page faults.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_TMSCP_EVERYTHING

Returns all the performance data items in the following order:

TMSCP_BUFWAIT	Count of requests that had to wait for TMSCP buffer memory
TMSCP_HSTSRV	Number of TMSCP served hosts
TMSCP_TAPSRV	Number of TMSCP served tapes
TMSCP_OPCOUNT	Total operations count
TMSCP_ABORTCNT	Total abort operations count
TMSCP_BUFAVAIL	Free TMSCP pool bytes
TMSCP_ONLINCNT	Count of online tapes
TMSCP_ACCESSCNT	Total access count
TMSCP_FLUSHCNT	Total flush count
TMSCP_RDCOUNT	Count of read I/O requests by remote processors
TMSCP_WRCOUNT	Count of write I/O requests by remote processors
TMSCP_VCFAIL	Number of virtual cache failures on TMSCP served requests in location 23

TMSCP_FRAGMENT	Extra fragments
TMSCP_SIZE1	Count of TMSCP served I/O requests with a length of 1 block
TMSCP_SIZE2	Count of TMSCP served I/O requests with a length of 2-3 blocks
TMSCP_SIZE3	Count of TMSCP served I/O requests with a length of 4-7 blocks
TMSCP_SIZE4	Count of TMSCP served I/O requests with a length of 8-15 blocks
TMSCP_SIZE5	Count of TMSCP served I/O requests with a length of 16-31 blocks
TMSCP_SIZE6	Count of TMSCP served I/O requests with a length of 32-63 blocks
TMSCP_SIZE7	Count of TMSCP served I/O requests with a length of 64 or more blocks

Because this is an array of 20 longwords, the buffer length field in the item descriptor should specify 4 times 20 (bytes).

RMI$_TQESYSUB

Returns the accumulated systemwide count of timer requests made by the OpenVMS operating system.

Because this number is a quadword, the buffer length field in the item descriptor should specify 8 (bytes).

RMI$_TQETOTAL

Returns the accumulated systemwide count of timer requests.

Because this number is a quadword, the buffer length field in the item descriptor should specify 8 (bytes).

RMI$_TQEUSRTIMR

Returns the accumulated systemwide count of timer requests made by application programs through the SYS$SETIMR system service.

Because this number is a quadword, the buffer length field in the item descriptor should specify 8 (bytes).

RMI$_TQEUSRWAKE

Returns the accumulated systemwide count of timer requests made by application programs through the SYS$SCHDWK system service.

Because this number is a quadword, the buffer length field in the item descriptor should specify 8 (bytes).

RMI$_TRANSFLTS

Returns the accumulated systemwide count of transition (release pending or read-in-progress) faults.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_TRCNGLOS

Returns the accumulated systemwide count of DECnet packets lost due to transit congestion.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_TTWRITES

Returns the accumulated systemwide count of writes to terminals.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_USERPAGES

Returns the number of pages available for use by applications.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes) on Alpha systems and 8 (bytes) on Integrity server systems.

RMI$_VMSPAGES

Returns the number of pages actually allocated to OpenVMS.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_VOLLCK

Returns the accumulated systemwide count of volume-synch locks.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_VOLWAIT

Returns the number of times the XQP entered a wait state due to volume lock contention.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_WRTFAULTS

Returns the number of fault-on-write page faults.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_WRTINPROG

Returns the number of page faults from a write in progress.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

RMI$_XQPCACHEWAIT

Returns the systemwide count of times the XQP waited for free space in a cache.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

Description

The Get Resource Monitor Information service returns performance information about the local system.

Required Access or Privileges

None.

Required Quota
This service uses the process's AST limit quota (ASTLM).

Related Services
None.

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 the 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.

$GETSYI

Get Systemwide Information — Returns information about the local system or about other systems in an OpenVMS Cluster 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. On Alpha and Integrity server systems, this service accepts 64-bit addresses.

Format

```
SYS$GETSYI
   [efn] ,[csidadr] ,[nodename] ,itmlst [,iosb] [,astadr] [,astprm]
```

C Prototype

```
int sys$getsyi
   (unsigned int efn, unsigned int *csidadr, void *nodename, void *itmlst,
    struct _iosb *iosb, void (*astadr)(__unknown_params),
    unsigned __int64 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.

VSI strongly recommends the use of the EFN$C_ENF "no event flag" value as the event flag if you are not using an event flag to externally synchronize with the completion of this system service call. The $EFNDEF macro defines EFN$C_ENF. For more information, see the *VSI OpenVMS Programming Concepts Manual*.

csidadr

OpenVMS usage:	process_id
type:	longword (unsigned)
access:	modify
mechanism:	by 32- or 64-bit reference

OpenVMS Cluster system identification of the node about which $GETSYI is to return information. The *csidadr* argument is the 32- or 64-bit address of a longword containing this identification value.

The cluster-connection software assigns the OpenVMS Cluster 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 32- or 64-bit descriptor–fixed-length string descriptor

Name of the node about which $GETSYI is to return information. The *nodename* argument is the 32- or 64-bit 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.

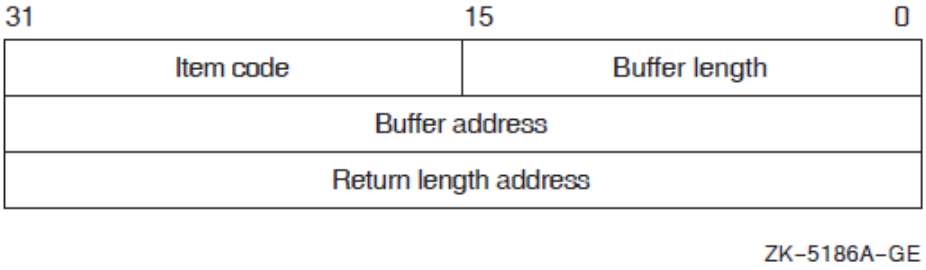
You can also specify a node to $GETSYI by using the *csidadr* argument. See the description of *csidadr*.

itmlst

OpenVMS usage:	32-bit item_list_3 or 64-bit item_list_64b
type:	longword (unsigned) for 32-bit; quadword (unsigned) for 64-bit
access:	read only
mechanism:	by 32- or 64-bit reference

Item list specifying which information is to be returned about the node or nodes. The `itmlst` argument is the 32- or 64-bit address of a list of item descriptors, each of which describes an item of information. An item list in 32-bit format is terminated by a longword of 0; an item list in 64-bit format is terminated by a quadword of 0. All items in an item list must be of the same format—either 32-bit or 64-bit.

The following diagram depicts the 32-bit format of a single item descriptor.

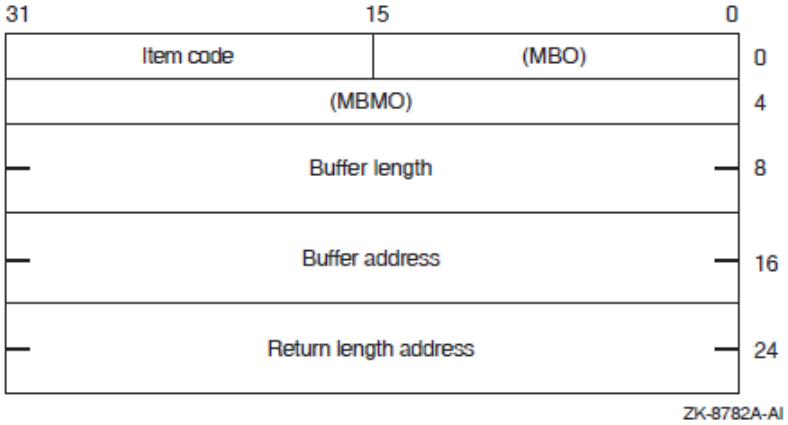


ZK–5186A–GE

The following table defines the item descriptor fields for 32-bit item list entries.

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.
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 32-bit address of the buffer into which $GETSYI is to write the information.
Return length address	A longword containing the user-supplied 32-bit address of a word in which $GETSYI writes the length in bytes of the information it actually returned.

The following diagram depicts the 64-bit format of a single item descriptor.



ZK-8782A-AI

The following table defines the item descriptor fields for 64-bit item list entries.

Descriptor Field	Definition
MBO	The field must contain a 1. The MBO and MBMO fields are used to distinguish 32-bit and 64-bit item list entries.
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.
MBMO	The field must contain a –1. The MBMO and MBO fields are used to distinguish 32-bit and 64-bit item list entries.
Buffer address	A quadword containing the user-supplied 64-bit address of the buffer into which $GETSYI is to write the information.
Return length address	A quadword containing the user-supplied 64-bit 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 32- or 64-bit reference

I/O status block to receive the final completion status. The *iosb* argument is the 32- or 64-bit 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, VSI 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 32- or 64-bit reference

AST service routine to be executed when $GETSYI completes. The *astadr* argument is the 32- or 64-bit 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$_ACTIVE_CPU_BITMAP

The return argument is a bitmap with a bit indicating a member of the instance's active set—those currently participating in the OpenVMS SMP scheduling activities.

The size of the returned bitmap is determined by the number of supported CPUs on the system. To compute the number of bytes needed for the bitmap, use the $GETSYI system service with an item code of SYI$_MAX_CPUS to find the minimum number of bits needed, round this number up to a multiple of 64, and divide the result by 8.

SYI$_ACTIVE_CPU_MASK

Note that this item code is becoming obsolete; VSI recommends that you not use it because it represents only up to 64 CPUs. The service continues to return the correct data for systems with up to 64 CPUs but fails for systems with more than 64 CPUs. For greater flexibility, use item code SYI$_ACTIVE_CPU_BITMAP instead.

On Alpha and Integrity server systems, returns a value that represents a CPU-indexed bitvector. When a particular bit position is set, the processor with that CPU ID value is a member of the instance's active set—those currently participating in the OpenVMS SMP scheduling activities.

SYI$_ACTIVECPU_CNT

Returns a count of the CPUs actively participating in the current boot of the symmetric multiprocessing (SMP) system.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

SYI$_ARCHFLAG

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

Returns, as a character string, the name of the CPU architecture on which the process is executing: "IA64" for Integrity servers, "Alpha" for OpenVMS Alpha, and "VAX" for OpenVMS 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

Returns the type of CPU architecture on which the process is executing: 3 for Integrity servers, 2 for Alpha, and 1 for VAX.

As this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

SYI$_AVAIL_CPU_BITMAP

The return argument is a bitmap with a bit indicating a member of the instance's configure set—those owned by the partition and controlled by the issuing instance.

The size of the returned bitmap is determined by the number of supported CPUs on the system. To compute the number of bytes needed for the bitmap, use the $GETSYI system service with an item code of SYI$_MAX_CPUS to find the minimum number of bits needed, round this number up to a multiple of 64, and divide the result by 8.

SYI$_AVAIL_CPU_MASK

Note that this item code is becoming obsolete; VSI recommends that you not use it because it represents only up to 64 CPUs. The service continues to return the correct data for systems with up to 64 CPUs but fails for systems with more than 64 CPUs. For greater flexibility, use item code SYI$_AVAIL_CPU_BITMAP instead.

On Alpha and Integrity server systems, returns a value that represents a CPU-indexed bitvector. When a particular bit position is set, the processor with that CPU ID value is a member of the instance's configure set—those owned by the partition and controlled by the issuing instance.

SYI$_AVAILCPU_CNT

Returns the number of CPUs available in the current boot of the symmetric multiprocessing (SMP) system.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

SYI$_BOOT_DEVICE

Returns the name of the device from which the system was booted as a character string. For a system with a shadowed system disk, SYI$_BOOT_DEVICE returns the name of the member device from which the shadow set was formed.

SYI$_BOOTTIME

Returns the time when the node was booted.

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

Returns the number 1 if the character string instructions are emulated on the CPU and the value 0 if they are not.

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

Returns the number of votes expected to be found in the OpenVMS Cluster 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

Returns the system identification of the founding node, which is the first node in the OpenVMS Cluster 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

Returns the time when the founding node is booted. The founding node is the first node in the OpenVMS Cluster 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

Returns the membership status of the node in the OpenVMS Cluster 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

Returns the number (in decimal) of nodes currently in the OpenVMS Cluster system.

Because this number is a word in length, the buffer length field in the item descriptor should specify 2 (bytes).

SYI$_CLUSTER_QUORUM

Returns the number (in decimal) that is the total of the quorum values held by all nodes in the OpenVMS Cluster 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

Returns the total number of votes held by all nodes in the OpenVMS Cluster 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$_COMMUNITY_ID

On Alpha and Integrity server systems, returns the hardware community ID for the issuing instance within the hard partition. Supported only on AlphaServer systems that support partitioning.

SYI$_CONTIG_GBLPAGES

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).

SYI$_CPU

As the processor type is a longword decimal number, the buffer length field in the item descriptor should specify 4 (bytes). On Alpha and Integrity server systems, $GETSYI returns PR$_SID_TYP_NOTAVAX.

For information about extended processor type codes, see the description for the SYI$_XCPU item code.

SYI$_CPU_AUTOSTART

On Alpha and Integrity server systems, returns a list of zeroes and ones, separated by commas and indexed by CPU ID. Any entry with a value of one indicates that specific CPU will be brought into the OpenVMS active set if it transitions into the current instance from outside, or is powered up while already owned.

SYI$_CPU_FAILOVER

Returns list of numeric partition IDs, separated by commas and indexed by CPU ID, that define the destination of the processor if the current instance should crash. Supported only on AlphaServer systems that support partitioning.

SYI$_CPUCAP_MASK

On Alpha and Integrity server systems, returns an array of quadword user capability masks for all CPUs in the system. This array is indexed by CPU ID and contains as many elements as the amount of space specified by the buffer length field in the item descriptor.

To minimize wasted space, a prior call to $GETSYI with SYI$_MAX_CPUS will provide the number of CPUs that need to be retrieved. Multiplying that value by 8 bytes for each quadword provides the value to be written in the buffer length field of the item descriptor.

SYI$_CPUCONF

Note that this item code is becoming obsolete; VSI recommends that you not use it because it represents only up to 64 CPUs. The service continues to return the correct data for systems with up to 64 CPUs but fails for systems with more than 64 CPUs. For greater flexibility, use item code SYI$_AVAIL_CPU_BITMAP instead.

On Alpha and Integrity server systems, returns a value that represents a CPU-indexed bitvector. When a particular bit position is set, the processor with that CPU ID value is a member of the instance's configure set—those owned by the partition and controlled by the issuing instance.

SYI$_CPUTYPE

On Alpha and Integrity server systems, returns the processor type, as stored in the hardware restart parameter block (HWRPB).

For example, 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).

The following table shows the processor codes and processors:

Processor Code	**Processor**
2	21064
4	21066, 21068, 21066A, 21068A
5	21164
6	21064A
7	21164A
8	21264
11	21264A
12	21264C
13	21264B
14	21264D
15	21364
16	21364
. . .	
31	Itanium 2
32	Itanium 3

SYI$_CWLOGICALS

Returns the number 1 if the clusterwide logical name database has been initialized on the CPU, or the value 0 if it has not been initialized. Because this number is a Boolean value (1 or 0), the buffer length field in the item descriptor should specify 1 (byte).

SYI$_DAY_OVERRIDE

Returns the number 1 if the SET DAY command has been used to override the default primary and secondary day types in the user authorization file that are used to control user logins. $GETSYI returns

the number 0 if no override is currently in effect, and the contents of user authorization file records for each user are being honored.

SYI$_DAY_SECONDARY

Returns the number 1 if any override with the SET DAY command has been used to specify that the current day is to be considered a Secondary day for user login purposes. $GETSYI returns the number 0 if any override with the SET DAY command has been used to specify that the current day is to be considered a Primary day for user login purposes.

If $GETSYI returns the number 0 for SYI$_DAY_OVERRIDE, the number returned for SYI$_DAY_SECONDARY is meaningless.

		SYI$_DAY_OVERRIDE	
		False	True
SYI$_DAY_SECONDARY	False	Use values from UAF	Today is a Primary day
	True	Use values from UAF	Today is a Secondary day

VM-0461A-AI

SYI$_DECIMAL_EMULATED

Returns the number 1 if the decimal string instructions are emulated on the CPU and the value 0 if they are not.

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

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

Returns the number 1 if the D_floating instructions are emulated on the CPU and 0 if they are not.

Because this number is a Boolean value (1 or 0), the buffer length field in the item descriptor should specify 1 (byte).

SYI$_DEF_PRIO_MAX

On Alpha and Integrity server systems, 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).

SYI$_DEF_PRIO_MIN

On Alpha and Integrity server systems, 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

Returns the number of pagelets 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

Returns the number of system pagelets 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

Returns the number 1 if the F_floating instructions are emulated on the CPU and 0 if they are not.

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

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

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

Returns the number 1 if the G_floating instructions are emulated on the CPU and the value 0 if they are not.

Because this number is a Boolean value (1 or 0), the buffer length field in the item descriptor should specify 1 (byte).

SYI$_GALAXY_ID

On Alpha systems, returns the 128-bit Galaxy ID. Supported only on AlphaServer GS series systems.

SYI$_GALAXY_MEMBER

On Alpha systems, returns 1 if you are member of a Galaxy sharing community, 0 if you are not a member. Supported only on AlphaServer GS series systems.

SYI$_GALAXY_PLATFORM

On Alpha systems, returns 1 if you are running on a Galaxy platform, 0 if you are not running on a Galaxy platform. Supported only on AlphaServer GS series systems.

SYI$_GALAXY_SHMEMSIZE

On Alpha systems, returns the number of shared memory pages. If the current instance is not a member of a Galaxy, no shared memory is reported. Supported only on AlphaServer GS series systems.

SYI$_GH_RSRVPGCNT

On Alpha and Integrity server systems, 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$_GLX_FORMATION

On Alpha systems, returns the a time-stamp string when the Galaxy configuration, of which this instance is a member, was created. Supported only on AlphaServer GS series systems.

SYI$_GLX_MAX_MEMBERS

On Alpha systems, returns the maximum count of instances that may join the current Galaxy configuration. Supported only on AlphaServer GS series systems.

SYI$_GLX_MBR_MEMBER

On Alpha systems, returns the 64-byte integer. Each 8 bytes represents a Galaxy member number, listed from 7 to 0. The value is 1 if the instance is currently a member, 0 if not a member. Supported only on AlphaServer GS series systems.

SYI$_GLX_MBR_NAME

On Alpha systems, returns a string indicating the names that are known in the Galaxy membership. Returned names are separated by two spaces, with an additional two spaces following the last name in the list. Supported only on AlphaServer GS series systems.

YI$_GLX_TERMINATION

On Alpha systems, returns a time-stamp string when the Galaxy configuration, of which this instance last was a member, was terminated. Supported only on AlphaServer GS series systems.

SYI$_H_FLOAT_EMULATED

Returns the number 1 if the H_floating instructions are emulated on the CPU and the value 0 if they are not.

Because this number is a Boolean value (1 or 0), the buffer length field in the item descriptor should specify 1 (byte).

SYI$_HP_ACTIVE_CPU_CNT

Returns the number of active CPUs in this hard partition that are not currently in firmware console mode. For OpenVMS, this implies that the CPU is in, or in the process of joining, the active set in one of the instances in the hard partition. Supported only on AlphaServer systems that support partitioning.

SYI$_HP_ACTIVE_SP_CNT

Returns the count of active operating system instances currently executing within the hard partition. Supported only on AlphaServer systems that support partitioning.

SYI$_HP_CONFIG_SBB_CNT

Returns a count of the existing system building blocks within the current hard partition. Supported only on AlphaServer systems that support partitioning.

SYI$_HP_CONFIG_SP_CNT

Returns the maximum count of soft partitions within the current hard partition. This count does not imply that an operating system instance is currently running within any given soft partition. Supported only on AlphaServer systems that support partitioning.

SYI$_HW_MODEL

Returns a small integer that can be used to identify the model type of the node.

An integer greater than 1023 indicates an Alpha and Integrity servers node.

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.

Because SYI$_HW_MODEL is a word, the buffer length field in the item descriptor should specify 2 (bytes).

SYI$_HW_NAME

Returns the model name string of the node. The model name is a character string that describes the model of the node. 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 60 characters plus one for the byte count, the buffer length field in the item descriptor should specify 61 (bytes). An example of an Integrity servers processor follows:

```
HP rx4640  (1.10GHz/4.0MB)
```

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
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

Alpha Model Processor Name	Alpha Model Type
DEC 10000 Model 630	ALPHA$K_A10000_630
DEC 10000 Model 640	ALPHA$K_A10000_640

SYI$_IO_PRCPU_BITMAP

The return argument is a bitmap with a bit indicating a preferred CPU—one available for Fast Path operations.

The size of the returned bitmap is determined by the number of supported CPUs on the system. To compute the number of bytes needed for the bitmap, use the $GETSYI system service with an item code of SYI$_MAX_CPUS to find the minimum number of bits needed, round this number up to a multiple of 64, and divide the result by 8.

SYI$_IO_PREFER_CPU

Note that this item code is becoming obsolete; VSI recommends that you not use it because it represents only up to 64 CPUs. The service continues to return the correct data for systems with up to 64 CPUs but fails for systems with more than 64 CPUs. For greater flexibility, use item code SYI$_IO_PRCPU_BITMAP instead.

On Alpha and Integrity server systems, returns the bit mask of CPUs available to be Fast Path preferred CPUs. The service continues to return the correct data for systems with up to 64 CPUs but fails for systems with more than 64 CPUs.

SYI$_ITB_ENTRIES

On Alpha and Integrity server systems, 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$_MAX_CPUS

On Alpha and Integrity server systems, returns the maximum number of CPUs that could be recognized by this instance.

SYI$_MAX_PFN

Returns the highest numbered PFN in use by the operating system. The highest numbered PFN used by OpenVMS is influenced by the PHYSICAL_MEMORY system parameter.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

SYI$_MEMSIZE

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

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

Returns the OpenVMS Cluster 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

Returns the number of votes the node expects to find in the OpenVMS Cluster 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

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

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

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

Returns the type of operating system software used by the node. The operating system software type indicates whether the node is an Alpha or Integrity servers 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

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

Returns the system identification of the node.

The OpenVMS 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$_NODE_VOTES

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

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

Returns the number of free pages in the currently installed page files.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

SYI$_PAGEFILE_PAGE

Returns the number of pages in the currently installed page files.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

SYI$_PAGE_SIZE

Returns the number of CPU-specific bytes per page in the system.

On Alpha and Integrity server systems, CPU page size varies from system to system.

On Alpha, and Integrity servers, because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

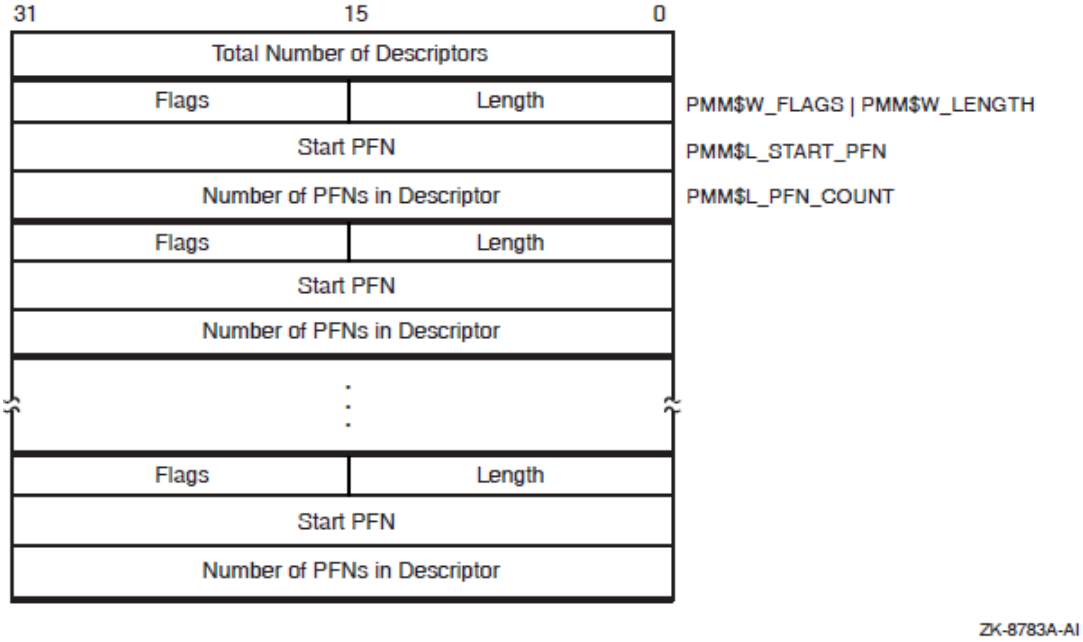
SYI$_PARTITION_ID

On Alpha and Integrity server systems, returns the soft partition ID. Supported only on AlphaServer systems that support partitioning.

SYI$_PFN_MEMORY_MAP (Alpha Only)

Returns a map describing the system's use of physical memory. Figure 1 shows an example of a physical memory map.

Figure 1. SYI$_PFN_MEMORY_MAP (Alpha Only)



The first longword of the physical memory contains a count of descriptors. This number is equal to the value returned when the SYI$_PMD_COUNT item code is specified.

Each descriptor contains at least 3 longwords: a word containing the length of the descriptor (always use PMM$C_LENGTH when determining descriptor size); a flags word (whose bits are defined in the following table); and two longwords containing the starting PFN for that physical memory cluster and the number of PFNs in that cluster.

Bit	Meaning When Set
PMM$V_CONSOLE	The physical memory descriptor is in use by the console (hardware).
PMM$V_OPENVMS	The physical memory descriptor is in use by OpenVMS.
PMM$V_AVAILABLE	The physical descriptor is not in use by either the console (hardware) or OpenVMS.
Remaining bits	The remaining bits in the PMM$W_FLAGS word are reserved to OpenVMS.

The structure definition for the physical memory descriptor resides in PMMDEF.H.

Because the size of the physical memory map returned by $GETSYI can vary from system to system, VSI recommends using the following steps when using this item code.

1. Call $GETSYI first using the SYI$_PMD_COUNT to obtain the number of physical memory descriptors.

2. Dynamically create a buffer to which $GETSYI can copy the physical memory map. The size of the buffer can be computed with the following formula:

```
map_buffer_size = (PMM$C_LENGTH * ret-val) + 4
```

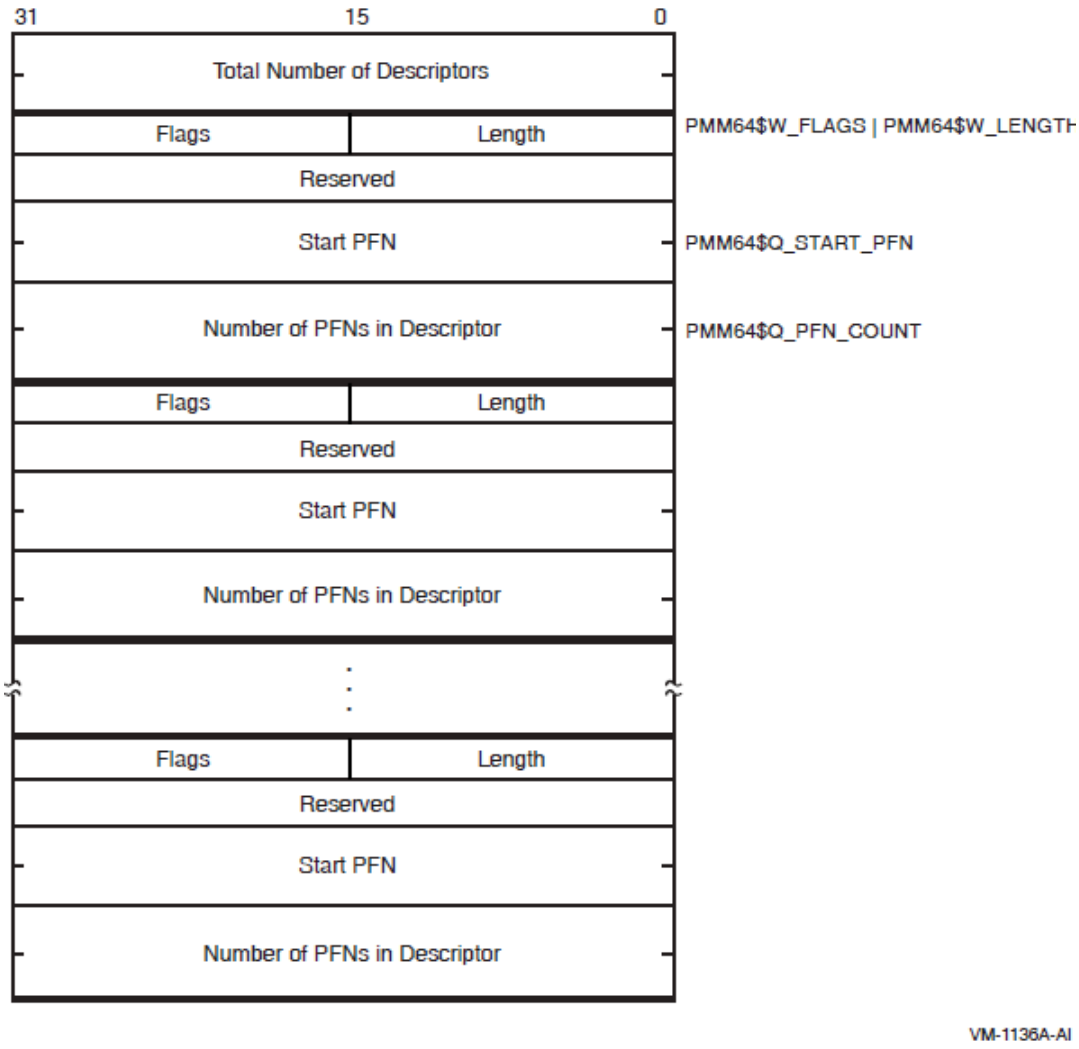
where:

- PMM$C_LENGTH is the size of an individual physical memory descriptor.
- ret-val is the return value from a call to $GETSYI specifying the SYI$_PMD_COUNT item code.
- 4 is the number of bytes occupied by the descriptor count in the physical memory map.

SYI$_PFN_MEMORY_MAP_64 (64-bit systems)

Returns a map describing the system's use of physical memory on 64-bit systems. Figure 2 shows an example of a physical memory map.

Figure 2. SYI$_PFN_MEMORY_MAP_64 (64-bit systems)



The first quadword of the physical memory contains a count of descriptors. This number is equal to the value returned when the SYI$_PMD_COUNT item code is specified.

Each descriptor contains at least 3 quadwords: a word containing the length of the descriptor (always use PMM64$C_LENGTH when determining descriptor size); a flags word (whose bits are defined in the following table); a reserved longword; and two quadwords containing the starting PFN for that physical memory cluster and the number of PFNs in that cluster.

Bit	Meaning When Set
PMM64$V_CONSOLE	The physical memory descriptor is in use by the console (hardware).
PMM64$V_OPENVMS	The physical memory descriptor is in use by OpenVMS.
PMM64$V_AVAILABLE	The physical descriptor is not in use by either the console (hardware) or OpenVMS.
Remaining bits	The remaining bits in the PMM64$W_FLAGS word are reserved to OpenVMS.

The structure definition for the physical memory descriptor resides in PMMDEF.H.

Because the size of the physical memory map returned by $GETSYI can vary from system to system, VSI recommends using the following steps when using this item code:

1. Call $GETSYI first using the SYI$_PMD_COUNT to obtain the number of physical memory descriptors.
2. Dynamically create a buffer to which $GETSYI can copy the physical memory map. The size of the buffer can be computed with the following formula:

```
map_buffer_size = (PMM64$C_LENGTH * ret-val) + 8
```

where:

- PMM64$C_LENGTH is the size of an individual physical memory descriptor.
- ret-val is the return value from a call to $GETSYI specifying the SYI$_PMD_COUNT item code.
- 8 is the number of bytes occupied by the descriptor count in the physical memory map.

SYI$_PHYSICALPAGES

Returns the total number of PFNs that exist between the first PFN (typically PFN 0) and the highest numbered PFN.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

SYI$_PMD_COUNT

Returns the total number of physical memory descriptors defined by the system. The return value of this parameter can be used to determine the buffer size to use when specifying the SYI$_PFN_MEMORY_MAP item code.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

SYI$_POTENTIAL_CPU_BITMAP

The return argument is a bitmap with a bit indicating a member of the instance's potential set. A CPU in the potential set implies that it could actively join the OpenVMS active set for this instance if it is ever owned by it. To meet this rule the CPU's characteristics must match hardware and software compatibility rules defined particularly for that instance.

The size of the returned bitmap is determined by the number of supported CPUs on the system. To compute the number of bytes needed for the bitmap, use the $GETSYI system service with an item code of SYI$_MAX_CPUS to find the minimum number of bits needed, round this number up to a multiple of 64, and divide the result by 8.

SYI$_POTENTIAL_CPU_MASK

Note that this item code is becoming obsolete; VSI recommends that you not use it because it represents only up to 64 CPUs. The service continues to return the correct data for systems with up to 64 CPUs but fails for systems with more than 64 CPUs. For greater flexibility, use item code SYI$_POTENTIAL_CPU_BITMAP instead.

On Alpha and Integrity server systems, returns a value that represents a CPU-indexed bit vector. When a particular bit position is set, the processor with that CPU ID value is a member of the instance's potential set. A CPU in the potential set implies that it could actively join the OpenVMS active set for this instance if it is ever owned by it. To meet this rule, the CPU's characteristics must match hardware and software compatibility rules defined particularly for that instance.

SYI$_POTENTIALCPU_CNT

On Alpha and Integrity server systems, returns the count of CPUs in the hard partition that are members of the potential set for this instance. A CPU in the potential set implies that it could actively join the OpenVMS active set for this instance if it is ever owned by it. To meet this rule the CPU's characteristics must match hardware and software compatibility rules defined particularly for that instance.

SYI$_POWERED_CPU_BITMAP

The return argument is a bitmap with a bit indicating a member of the instance's powered set—those CPUs physically existing within the hard partition and powered up for operation.

The size of the returned bitmap is determined by the number of supported CPUs on the system. To compute the number of bytes needed for the bitmap, use the $GETSYI system service with an item code of SYI$_MAX_CPUS to find the minimum number of bits needed, round this number up to a multiple of 64, and divide the result by 8.

SYI$_POWERED_CPU_MASK

Note that this item code is becoming obsolete; VSI recommends that you not use it because it represents only up to 64 CPUs. The service continues to return the correct data for systems with up to 64 CPUs but fails for systems with more than 64 CPUs. For greater flexibility, use item code SYI$_POWERED_CPU_BITMAP instead.

On Alpha and Integrity server systems, returns a value that represents a CPU-indexed bitvector. When a particular bit position is set, the processor with that CPU ID value is a member of the instance's powered set—those CPUs physically existing within the hard partition and powered up for operation.

SYI$_POWEREDCPU_CNT

On Alpha and Integrity server systems, returns the count of CPUs in the hard partition that are physically powered up.

SYI$_PRESENT_CPU_BITMAP

The return argument is a bitmap with a bit indicating a member of the instance's present set—those CPUs physically existing within the hard partition. Being in the present set does not imply that it is part of the powered set.

The size of the returned bitmap is determined by the number of supported CPUs on the system. To compute the number of bytes needed for the bitmap, use the $GETSYI system service with an item code of SYI$_MAX_CPUS to find the minimum number of bits needed, round this number up to a multiple of 64, and divide the result by 8.

SYI$_PRESENT_CPU_MASK

Note that this item code is becoming obsolete, and VSI recommends that you not use it because it represents only up to 64 CPUs. The service continues to return the correct data for systems with up to 64 CPUs but fails for systems with more than 64 CPUs. For greater flexibility, use item code SYI$_PRESENT_CPU_BITMAP instead.

On Alpha and Integrity server systems, returns a value that represents a CPU-indexed bitvector. When a particular bit position is set, the processor with that CPU ID value is a member of the instance's present set—those CPUs physically existing within the hard partition. Being in the present set does not imply that it is part of the powered set.

SYI$_PRESENTCPU_CNT

On Alpha and Integrity server systems, returns the count of CPUs in the hard partition that physically reside in a hardware slot.

SYI$_PRIMARY_CPUID

On Alpha and Integrity server systems, returns the CPU ID of the primary processor for this OpenVMS instance.

SYI$_PROCESS_SPACE_LIMIT

On Alpha and Integrity server 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).

SYI$_PSXFIFO_PRIO_MAX

On Alpha and Integrity server systems, 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).

SYI$_PSXFIFO_PRIO_MIN

On Alpha and Integrity server systems, 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).

SYI$_PSXRR_PRIO_MAX

On Alpha and Integrity server systems, 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).

SYI$_PSXRR_PRIO_MIN

On Alpha and Integrity server systems, 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).

SYI$_PT_BASE

On Alpha and Integrity server systems, 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$_PTES_PER_PAGE

On Alpha and Integrity server systems, returns the maximum number of CPU-specific pages that can be mapped by one page table page.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

SYI$_RAD_CPUS

On Alpha and Integrity server systems, returns a longword array of RAD/CPU pairs that can potentially be in this operating system instance. If there is no RAD support, all potential CPUs are in RAD 0. The array is terminated with a -1,-1 pair.

RAD is supported on AlphaServer GS series systems and starting from OpenVMS Version 8.4, support is extended to NUMA capable Integrity servers.

SYI$_RAD_MEMSIZE

On Alpha and Integrity server systems, returns a longword array of RAD/page count pairs. The number of pages of private memory is returned. If there is no RAD support, all memory is reported in RAD 0. The array is terminated with a -1,-1 pair.

RAD is supported on AlphaServer GS series systems and starting from OpenVMS Version 8.4, support is extended to NUMA capable Integrity servers.

SYI$_RAD_MAX_RAD

On Alpha and Integrity server systems, returns the maximum number of RADs possible on this platform. If there is no RAD support, 1 is returned.

RAD is supported on AlphaServer GS series systems and starting from OpenVMS Version 8.4, support is extended to NUMA capable Integrity servers.

SYI$_RAD_SHMEMSIZE

On Alpha and Integrity server systems, returns a longword array of RAD/page count pairs. The number of pages of shared memory is returned. If there is no RAD support, all shared memory is reported in RAD 0. If the current instance is not a member of a Galaxy, no shared memory is reported. The array is terminated with a -1,-1 pair.

RAD is supported on AlphaServer GS series systems and starting from OpenVMS Version 8.4, support is extended to NUMA capable Integrity servers.

SYI$_REAL_CPUTYPE

Returns the actual CPU type of the primary CPU of the system.

See the SYI$_CPUTYPE item code for a list of symbols and processors.

SYI$_SCSNODE

Returns the Galaxy instance name. Supported only on AlphaServer systems that support partitioning.

SYI$_SCS_EXISTS

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.

SYI$_SERIAL_NUMBER

Returns the system serial number from the Hardware Restart Parameter Block (HWRPB).

SYI$_SHARED_VA_PTES

On Alpha and Integrity server systems, 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

Returns the contents of the system identification register of the node.

On Alpha and Integrity server 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

Returns the number of free pages in the currently installed swapping files.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

SYI$_SWAPFILE_PAGE

Returns the number of pages in the currently installed swapping files.

Because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

SYI$_SYSTEM_RIGHTS

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.

SYI$_SYSTEM_UUID

On Integrity server systems, returns the Universal Unique Identifier (UUID) for the system. If the UUID is not supported or is not available, a null UUID (all zeroes) is returned.

Because SYI$_SYSTEM_UUID is 128 bits long, the buffer length field in the item descriptor must specify 16 bytes.

SYI$_SYSTYPE

On Alpha and Integrity server systems, 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

Returns, as a character string, the software version number of the OpenVMS operating system running on the 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

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$_VIRTUAL_MACHINE

On Integrity and x86-64 server systems, returns 1 if OpenVMS is running as a virtual machine guest, and returns 0 if it is not.

Because this number is a Boolean value (1 or 0), the buffer length field in the item descriptor should specify 1 (byte)

SYI$_VP_MASK

Returns a longword mask, the bits of which, when set, indicate which processors in the system have vector coprocessors.

SYI$_VP_NUMBER

Returns an unsigned longword containing the number of vector processors in the system.

SYI$_XCPU

Returns the extended CPU processor type of the 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).

SYI$_XSID

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

Returns the current value of the system parameter named *xxxx* for the 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 *VSI OpenVMS System Manager's Manual*.

Description

The Get Systemwide Information service returns information about the local system or about other systems in an OpenVMS Cluster configuration.

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 OpenVMS Cluster system.

Condition Values Returned in the I/O Status Block

Same as those returned in R0.

Example

```
/*  Defining __NEW_STARLET enables the program to benefit from better type
    checking for prototypes and structures provided by OpenVMS.             */

#define     __NEW_STARLET       1

#include    <efndef>            /*  No Event Flag Event Flag  */
#include    <iledef>            /*  Item List Entry Definitions  */
#include    <iosbdef>           /*  I/O Status Block Structure Definition  */
#include    <starlet>           /*  Function Prototypes for System Services  */
#include    <stdio>             /*  C Standard I/O Functions  */
#include    <string>            /*  memset Prototype  */
#include    <syidef>            /*  $GETSYI Item Code Definitions  */

#define     NUM_ILE              3
#define     BUFFER_SIZE         20


/*  Macro to initialize a 32-bit item_list_3.                               */

#define init_ile32(ile, length, code, bufaddr, retlen_addr) \
{   (ile)->ile3$w_length = (length);           \
    (ile)->ile3$w_code = (code);         \
    (ile)->ile3$ps_bufaddr = (bufaddr); \
    (ile)->ile3$ps_retlen_addr = (retlen_addr); }


/*  Simple status checking macro.                                           */

#define bad_status(status) (((status) & 1) != 1)

main ()
{
char
    node_name [BUFFER_SIZE],
    version_string [BUFFER_SIZE];

int
    status;
```

```
unsigned short
    node_name_length,
    version_string_length;

ILE3
    syi_ile [NUM_ILE];

IOSB
    iosb;


/*  Zeroing the item list has the effect of creating the terminating entry. */

    memset (syi_ile, 0, ILE3$K_LENGTH*NUM_ILE);


/*  Initialize the item list entries to fetch the operating system version
    and the node name.                                                      */

    init_ile32 (
        &syi_ile [0],
        BUFFER_SIZE,
        SYI$_VERSION,
        version_string,
        &version_string_length);

    init_ile32 (
        &syi_ile [1],
        BUFFER_SIZE,
        SYI$_NODENAME,
        node_name,
        &node_name_length);

    status = sys$getsyiw (
        EFN$C_ENF,
        NULL,
        NULL,
        &syi_ile,
        &iosb,
        NULL,
        0);

    if (bad_status (status)) return status;
    if (bad_status (iosb.iosb$w_status)) return iosb.iosb$w_status;


/*  Zero terminate the strings before displaying them.                      */

    version_string [version_string_length] = '\0';
    node_name [node_name_length] = '\0';

    printf ("Version:  %s    Node Name:  %s\n",
        version_string,
        node_name);
}
```

This example C program demonstrates how to use $GETSYIW to obtain the operating system version number string and the node name.

$GETSYIW

Get Systemwide Information and Wait — Returns information about the local system or about other systems in a cluster.

Format

```
SYS$GETSYIW
   [efn] ,[csidadr] ,[nodename] ,itmlst [,iosb] [,astadr]
   [,astprm]
```

C Prototype

```
int sys$getsyiw
   (unsigned int efn, unsigned int *csidadr, void *nodename, void *itmlst,
    struct _iosb *iosb, void (*astadr)(__unknown_params), int astprm);
```

You must specify either the *csidadr* or the *nodename* argument, but not both. For wildcard operations, however, you must use the *csidadr* argument.

Description

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; For information about the $GETSYIW service, see the documentation about $GETSYI.

For additional information about system service completion, see the Synchronize ($SYNCH) service.

On Alpha and Integrity server systems, this service accepts 64-bit addresses.

$GETTIM

Get Time — Returns the current system time in a 64-bit format. On Alpha and Integrity server systems, this service accepts 64-bit addresses.

Format

```
SYS$GETTIM timadr,[flags]
```

C Prototype

```
int sys$gettim (struct _generic_64 *timadr,...);
```

Argument

timadr

OpenVMS usage:	date_time
type:	quadword (unsigned)
access:	write only
mechanism:	by 32- or 64-bit reference

The 32- or 64-bit address of a quadword to receive the current time in 64-bit format.

flags

OpenVMS usage:	mask_longword
type:	longword (unsigned)
access:	read only
mechanism:	by value (Alpha and Integrity servers)

An optional argument that modifies the form of the returned time value. If this argument is not specified, it is same as specifying an argument of 0.

Description

The Get Time service returns time data in different formats depending on the *flags* parameter. If the *flags* parameter is not specified, or is specified as 0, the service returns the current system time in 64-bit format. The quadword is the number of 100-nanosecond intervals since November 17, 1858.

If the *flags* parameter is set to 1, the service returns a 64-bit time value as the number of 100-nanosecond intervals since an arbitrary point in the past such as system boot time. This base time does not change unless the system is rebooted. Since the returned value does not represent a time of day, it will not change if the system time is changed (for example, by $SETIME or an automatic timezone change). It increments at the same rate as the system time. Other flag values are reserved for future use.

On Alpha and Integrity server systems, the exact frequency at which system time is updated varies, depending on the characteristics of the server; however the update frequency is approximately 1 millisecond.

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 *VSI 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.

SS$_BADPARAM

The *flags* parameter contains a reserved value.

$GETTIM_PREC

Get Time Precision — Returns the current high precision system time in a 64-bit format. On Alpha and Integrity server systems, this service accepts 64-bit addresses.

Format

```
SYS$GETTIM_PREC timadr
```

C Prototype

```
int sys$gettim_prec (struct _generic_64 *timadr);
```

Arguments

timadr

OpenVMS usage:	date_time
type:	quadword (unsigned)
access:	write only
mechanism:	by 64-bit reference

The 64-bit address of a quadword to receive the current time in 64-bit format.

Description

The Get Time Precision service returns the current system time in 64-bit format. The quadword is the number of 100 nanoseconds since November 17, 1858.

On Alpha and Integrity server systems, the frequency at which system time is updated varies, depending on the clock frequency of the Alpha or Integrity server processor, or approximately 1 millisecond.

On Integrity servers, $GETTIM_PREC returns the system time after accounting for the time elapsed since the last one millisecond update of the system time.

On Alpha, this service is equivalent to $GETTIM service and just returns the last updated system time. The successful completion status of the service on ALPHA is SS$_LOWPREC.

Required Access or Privileges

None

Required Quota

None

Related Services

$GETTIM

For additional information about the system time, see the *VSI OpenVMS System Manager's Manual*.

Condition Values Returned

SS$_NORMAL

The service completed successfully.

SS$_LOWPREC

The platform does not support high precision time or it is not possible to return high precision time at this point. The time returned is just the low precision time after successful completion of the service.

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]
```

C Prototype

```
int sys$getuai
   (unsigned int efn, unsigned int *contxt, void *usrnam, void *itmlst,
    struct _iosb *iosb, void (*astadr)(__unknown_params), int astprm);
```

Arguments

nullarg

OpenVMS usage:	null_arg
type:	longword (unsigned)
access:	read only
mechanism:	by value

Placeholding argument reserved to OpenVMS.

contxt

OpenVMS usage:	longword
type:	longword (unsigned)
access:	modify
mechanism:	by reference

An optional longword used to maintain an open channel to the authorization file. The *contxt* argument is the address of a longword to receive a $GETUAI context value. If the *contxt* argument is specified on the initial call, this longword should contain the value –1, and 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

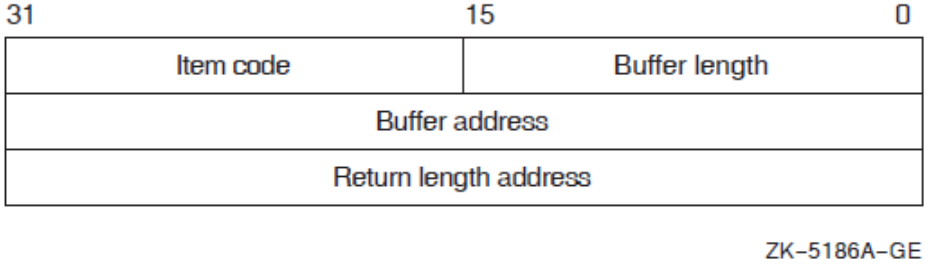
Name of the user about whom $GETUAI returns authorization information. The *usrnam* argument is the address of a descriptor pointing to a character text string containing the user name. The user name string can contain a maximum of 12 alphanumeric characters.

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.



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.

Descriptor Field	Definition
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.

nullarg

OpenVMS usage:	nullarg
type:	quadword (unsigned)
access:	write only
mechanism:	by reference

Placeholding argument reserved to OpenVMS.

nullarg

OpenVMS usage:	nullarg
type:	procedure entry mask
access:	call without stack unwinding
mechanism:	by reference

Placeholding argument reserved to OpenVMS.

nullarg

OpenVMS usage:	nullarg
type:	longword (unsigned)
access:	read only
mechanism:	by value

Placeholding argument reserved to OpenVMS.

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

Returns the default working set size in pagelets .

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_DISPWDSYNCH	When set, prevents synchronizing a user's SYSUAF.DAT password using an external authentication password (when UAI$V_EXTAUTH is set).
UAI$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$V_EXTAUTH	User is considered to be externally authenticated by their external user ID and password, and not by the SYSUAF user ID and password. The SYSUAF record is still used for checking login restrictions and quotas and for creating the user's OpenVMS process profile.
UAI$V_GENPWD	User is required to use generated passwords.
UAI$V_LOCKPWD	SET PASSWORD command is disabled.
UAI$V_MIGRATEPWD	User's SYSUAF password has been set using AUTHORIZE or SYS$SETUAI and is likely to be inconsistent with the user's LAN Manager password. If password migration is enabled, the system will attempt to update the LAN Manager the next time the user attempts a login.
UAI$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_PWDMIX	Case-sensitive and extended-character passwords are enabled. Password verification will be case sensitive.

Symbolic Name	Description
UAI$V_RESTRICTED	User is limited to operating under a restricted account. See the *VSI OpenVMS Guide to System Security* for a description of restricted and captive accounts.
UAI$V_VMSAUTH	When set, the user is allowed to authenticate using the user's SYSUAF.DAT user name and password (when UAI$V_EXTAUTH is set).

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

Returns the paging file quota in blocks (on Alpha and Integrity server 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 byte 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 *VSI OpenVMS Guide to System Security*.

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 *VSI OpenVMS Guide to System Security*.

UAI$_WSEXTENT

Returns the working set extent, in pagelets , 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

Returns the working set quota, in pagelets , 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.

The *contxt* value returned by $GETUAI should never be used as a value to the $SETUAI system service.

You examine for a valid login by checking the bits of UAI$V_PWD_EXPIRED and UAI$V_DISUSER, and by doing a comparison of the UAI$_PWD_DATE item code against the UAI$_PWD_LIFETIME item code.

The UAI$V_PWD_EXPIRED bit is only set by the system when the bit UAI$V_DISFORCE_PWD_CHANGE is set in the user's SYSUAF record and the comparison between the UAI$_PWD_DATE and UAI$_PWD_LIFETIME indicates a password is past its valid life.

During a normal login when the UAI$V_DISFORCE_PWD_CHANGE bit is not set, the system compares VAI$_PWD_DATE against UAI$_PWD_LIFETIME and, if expired, forces the user to change the password. With this configuration, the UAI$V_PWD_EXPIRED bit is not set.

During a normal login when the VAI$V_DISFORCE_PWD_EXPIRED is set, the system compares UAI$_PWD_DATE against UAI$_PWD_LIFETIME and, if expired, sets the UAI$_PWD_EXPIRED bit and notifies the user to change the now-expired password. In this case, the user is not forced to change the password.

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 *VSI OpenVMS Record Management Services Reference Manual*.