

---

# OpenVMS System Services Reference Manual: A-GETUAI

Order Number: AA-QSBMD-TE

**April 2001**

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

This manual is in two parts. This first part contains the system services from A through \$GETUAI.

**Revision/Update Information:** This manual supersedes the *OpenVMS System Services Reference Manual*, Version 7.2.

**Software Version:** OpenVMS Alpha Version 7.3  
OpenVMS VAX Version 7.3

**Compaq Computer Corporation  
Houston, Texas**

---

© 2001 Compaq Computer Corporation

Compaq, VAX, VMS, and the Compaq logo Registered in U.S. Patent and Trademark Office.

OpenVMS and Tru64 are trademarks of Compaq Information Technologies Group, L.P in the United States and other countries.

Microsoft, MS-DOS, Visual C++, Windows, and Windows NT are trademarks of Microsoft Corporation in the United States and other countries.

Intel, Intel Inside, and Pentium are trademarks of Intel Corporation in the United States and other countries.

Motif, OSF/1, and UNIX are trademarks of The Open Group in the United States and other countries.

All other product names mentioned herein may be trademarks of their respective companies.

Confidential computer software. Valid license from Compaq required for possession, use, or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Compaq shall not be liable for technical or editorial errors or omissions contained herein. The information in this document is provided "as is" without warranty of any kind and is subject to change without notice. The warranties for Compaq products are set forth in the express limited warranty statements accompanying such products. Nothing herein should be construed as constituting an additional warranty.

ZK6243

The Compaq *OpenVMS* documentation set is available on CD-ROM.

This document was prepared using DECdocument, Version 3.3-1b.

---

# Contents

Preface .....	vii
---------------	-----

## System Service Descriptions

\$ABORT_TRANS .....	SYS1-3
\$ABORT_TRANSW .....	SYS1-7
\$ACQUIRE_GALAXY_LOCK (Alpha Only) .....	SYS1-8
\$ADD HOLDER .....	SYS1-10
\$ADD_IDENT .....	SYS1-13
\$ADD_PROXY .....	SYS1-16
\$ADJSTK .....	SYS1-20
\$ADJWSL .....	SYS1-22
\$ALLOC .....	SYS1-24
\$ASCEFC .....	SYS1-27
\$ASCTIM .....	SYS1-31
\$ASCTOID .....	SYS1-34
\$ASCUTC .....	SYS1-37
\$ASSIGN .....	SYS1-40
\$AUDIT_EVENT .....	SYS1-45
\$AUDIT_EVENTW .....	SYS1-63
\$AVOID_PREEMPT .....	SYS1-64
\$BINTIM .....	SYS1-66
\$BINUTC .....	SYS1-69
\$BRKTHRU .....	SYS1-72
\$BRKTHRUW .....	SYS1-80
\$CANCEL .....	SYS1-81
\$CANEXH .....	SYS1-83
\$CANTIM .....	SYS1-84
\$CANWAK .....	SYS1-86
\$CHECK_ACCESS .....	SYS1-88
\$CHECK_FEN (Alpha Only) .....	SYS1-96
\$CHECK_PRIVILEGE .....	SYS1-97
\$CHECK_PRIVILEGEW .....	SYS1-102
\$CHKPRO .....	SYS1-103
\$CLEAR_SYSTEM_EVENT (Alpha Only) .....	SYS1-111
\$CLRST .....	SYS1-113
\$CLRCLUEVT .....	SYS1-114
\$CLREF .....	SYS1-116
\$CMEXEC .....	SYS1-117

\$CMEXEC_64 (Alpha Only) . . . . .	SYS1-119
\$CMKRNL . . . . .	SYS1-121
\$CMKRNL_64 (Alpha Only) . . . . .	SYS1-124
\$CPU_CAPABILITIES (Alpha Only) . . . . .	SYS1-126
\$CPU_TRANSITION (Alpha Only) . . . . .	SYS1-130
\$CPU_TRANSITIONW (Alpha Only) . . . . .	SYS1-135
\$CREATE_GALAXY_LOCK (Alpha Only) . . . . .	SYS1-136
\$CREATE_GALAXY_LOCK_TABLE (Alpha Only) . . . . .	SYS1-139
\$CREATE_BUFOBJ_64 (Alpha Only) . . . . .	SYS1-142
\$CREATE_GDZRO (Alpha Only) . . . . .	SYS1-146
\$CREATE_GFILE (Alpha Only) . . . . .	SYS1-154
\$CREATE_GPFILE (Alpha Only) . . . . .	SYS1-160
\$CREATE_GPFN (Alpha Only) . . . . .	SYS1-164
\$CREATE_RDB . . . . .	SYS1-169
\$CREATE_REGION_64 (Alpha Only) . . . . .	SYS1-171
\$CREATE_USER_PROFILE . . . . .	SYS1-179
\$CRELMN . . . . .	SYS1-185
\$CRELNT . . . . .	SYS1-193
\$CREMBX . . . . .	SYS1-199
\$CREPRC . . . . .	SYS1-206
\$CRETVA . . . . .	SYS1-224
\$CRETVA_64 (Alpha Only) . . . . .	SYS1-227
\$CRMPSC . . . . .	SYS1-232
\$CRMPSC_FILE_64 (Alpha Only) . . . . .	SYS1-245
\$CRMPSC_GDZRO_64 (Alpha Only) . . . . .	SYS1-251
\$CRMPSC_GFILE_64 (Alpha Only) . . . . .	SYS1-264
\$CRMPSC_GPFILE_64 (Alpha Only) . . . . .	SYS1-272
\$CRMPSC_GPFN_64 (Alpha Only) . . . . .	SYS1-279
\$CRMPSC_PFN_64 (Alpha Only) . . . . .	SYS1-286
\$CVT_FILENAME (Alpha Only) . . . . .	SYS1-291
\$DACEFC . . . . .	SYS1-296
\$DALLOC . . . . .	SYS1-298
\$DASSGN . . . . .	SYS1-300
\$DCLAST . . . . .	SYS1-302
\$DCLCMH . . . . .	SYS1-304
\$DCLEXH . . . . .	SYS1-307
\$DELETE_BUFOBJ (Alpha Only) . . . . .	SYS1-310
\$DELETE_GALAXY_LOCK (Alpha Only) . . . . .	SYS1-312
\$DELETE_GALAXY_LOCK_TABLE (Alpha Only) . . . . .	SYS1-314
\$DELETE_INTRUSION . . . . .	SYS1-315
\$DELETE_PROXY . . . . .	SYS1-318
\$DELETE_REGION_64 (Alpha Only) . . . . .	SYS1-321
\$DELLNM . . . . .	SYS1-324
\$DELMBX . . . . .	SYS1-327
\$DELPRC . . . . .	SYS1-329
\$DELTVA . . . . .	SYS1-332
\$DELTVA_64 (Alpha Only) . . . . .	SYS1-334

\$DEQ	SYS1-338
\$DEVICE_PATH_SCAN	SYS1-343
\$DEVICE_SCAN	SYS1-347
\$DGBLSC	SYS1-351
\$DISMOU	SYS1-355
\$DISPLAY_PROXY	SYS1-359
\$DLCEFC	SYS1-365
\$DNS (VAX Only)	SYS1-367
\$DNSW (VAX Only)	SYS1-394
\$SEND_TRANS	SYS1-395
\$SEND_TRANSW	SYS1-400
\$ENQ	SYS1-401
\$ENQW	SYS1-414
\$ERAPAT	SYS1-415
\$EXIT	SYS1-418
\$EXPREG	SYS1-419
\$EXPREG_64 (Alpha Only)	SYS1-422
\$FAO/SFAOL	SYS1-426
\$FAOL_64 (Alpha Only)	SYS1-447
\$FILESCAN	SYS1-449
\$FIND_HELD	SYS1-455
\$FIND HOLDER	SYS1-458
\$FINISH_RDB	SYS1-461
\$FORCEX	SYS1-463
\$FORMAT_ACL	SYS1-466
\$FORMAT_AUDIT	SYS1-479
\$FREE_USER_CAPABILITY (Alpha Only)	SYS1-483
\$GETDVI	SYS1-485
\$GETDVIW	SYS1-505
\$GETENV (Alpha Only)	SYS1-506
\$GETJPI	SYS1-508
\$GETJPIW	SYS1-533
\$GETLKI	SYS1-534
\$GETLKIW	SYS1-546
\$GETMSG	SYS1-547
\$GETQUI	SYS1-551
\$GETQUIW	SYS1-594
\$GETRMI	SYS1-595
\$GETSYI	SYS1-602
\$GETSYIW	SYS1-626
\$GETTIM	SYS1-627
\$GETUAI	SYS1-628

## Index

### Tables

SYS1-1	Abort Reason Codes . . . . .	SYS1-4
SYS1-2	Description of \$AUDIT_EVENT Types and Subtypes . . . . .	SYS1-49
SYS1-3	Services That Do Not Accept Shared Page Table Regions . . . . .	SYS1-175
SYS1-4	Sample Returned Values from \$CREATE_REGION_64 . . . . .	SYS1-177
SYS1-5	User Privileges . . . . .	SYS1-207
SYS1-6	Required and Optional Arguments for the \$CRMPSC Service . . . . .	SYS1-238
SYS1-7	Shared Page Tables . . . . .	SYS1-260
SYS1-8	\$DNS Item Codes and Their Data Types . . . . .	SYS1-382
SYS1-9	Compatibility of Lock Modes . . . . .	SYS1-403
SYS1-10	Legal QUECVT Conversions . . . . .	SYS1-408
SYS1-11	\$FAO Directives . . . . .	SYS1-430
SYS1-12	\$FAO Output Field Lengths and Fill Characters . . . . .	SYS1-437
SYS1-13	Attributes of an Identifier . . . . .	SYS1-518

## Intended Audience

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

## System Services Support for 64-Bit Alpha 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 *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 *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.

## Document Structure

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

## Related Documents

The *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.

The following documents might also be useful:

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

For a complete list and description of the manuals in the OpenVMS document set, refer to the *OpenVMS Version 7.3 New Features and Documentation Overview*.

For additional information about the Compaq *OpenVMS* products and services, access the Compaq website at the following location:

<http://www.openvms.compaq.com/>

## Reader's Comments

Compaq welcomes your comments on this manual. Please send comments to either of the following addresses:

Internet                    **openvmsdoc@compaq.com**



Mail Compaq Computer Corporation  
OSSG Documentation Group, ZKO3-4/U08  
110 Spit Brook Rd.  
Nashua, NH 03062-2698

## How To Order Additional Documentation

Visit the following World Wide Web address for information about how to order additional documentation:

<http://www.openvms.compaq.com/>

If you need help deciding which documentation best meets your needs, call 800-282-6672.

## Conventions

In this manual, any reference to OpenVMS is synonymous with Compaq OpenVMS.

VMScluster systems are now referred to as OpenVMS Cluster systems. Unless otherwise specified, references to OpenVMS Clusters or clusters in this document are synonymous with VMSclusters.

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

The following conventions are also used in this manual:

- |  |   |
|--|---|
| Ctrl/ <i>x</i>   | A sequence such as Ctrl/ <i>x</i> indicates that you must hold down the key labeled Ctrl while you press another key or a pointing device button.   |
| PF1 <i>x</i>   | A sequence such as PF1 <i>x</i> indicates that you must first press and release the key labeled PF1 and then press and release another key or a pointing device button.   |
| <span style="border: 1px solid black; padding: 2px;">Return</span> | In examples, a key name enclosed in a box indicates that you press a key on the keyboard. (In text, a key name is not enclosed in a box.)<br><br>In the HTML version of this document, this convention appears as brackets, rather than a box.  |
| ...  | A horizontal ellipsis in examples indicate one of the following possibilities: <ul style="list-style-type: none"><li>• Additional optional arguments in a statement have been omitted.</li><li>• The preceding item or items can be repeated one or more times.</li><li>• Additional parameters, values, or other information can be entered.</li></ul> |
| .  | 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 the options in parentheses if you choose more than one.  |

[ ]	In the <i>OpenVMS System Services Reference Manual</i> , 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 separating items inside brackets indicate that you choose one, none, or more than one of the options.
{ }	In command format descriptions, braces indicate required elements; you must choose one of the options listed.
<b>bold text</b>	This text style represents the introduction of a new term or the name of an argument, an attribute, or a reason.
<i>italic text</i>	Italic text indicates important information, complete titles of manuals, or variables. Variables include information that varies in system output (Internal error <i>number</i> ), in command lines ( <i>/PRÓDUCER=name</i> ), and in command parameters in text (where <i>dd</i> represents the predefined code for the device type).
UPPERCASE TEXT	Uppercase text indicates a command, the name of a routine, the name of a file, or the abbreviation for a system privilege.
Monospace text	Monospace type indicates code examples and interactive screen displays.  In the C programming language, monospace type identifies the following elements: keywords, the names of independently compiled external functions and files, syntax summaries, and references to variables or identifiers introduced in an example.
-	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.
numbers	All numbers in text are assumed to be decimal unless otherwise noted. Nondecimal radices—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 not only indicate 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]]

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

DDTM\$M\_SYNC, the only flag currently defined, is described in the following table:

Flag	Description
DDTM\$M_SYNC	Set this flag to specify that successful synchronous completion is to be indicated by returning SSS_SYNC. When SSS_SYNC 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 Condition Values Returned section.

## System Service Descriptions

### \$ABORT\_TRANS

- An abort reason code that gives one reason why the transaction aborted, if the completion status of the service is SSS\_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.

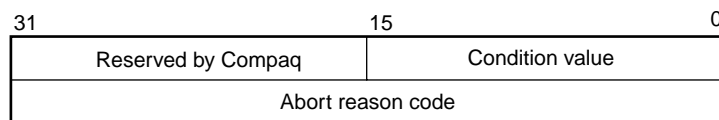
For example, if the call to \$ABORT\_TRANS gives DDTMS\_ABORTED as the reason and the transaction timeout expires at about the same time as the call to \$ABORT\_TRANS, then either the DDTMS\_TIMEOUT or DDTMS\_ABORTED reason code can be returned in the I/O status block.

The \$DDTMMSGDEF macro defines symbolic names for abort reason codes. Those currently defined are shown in Table SYS1-1.

**Table SYS1-1 Abort Reason Codes**

Symbolic Name	Description
DDTMS_ABORTED	The application aborted the transaction.
DDTMS_COMM_FAIL	A communications link failed.
DDTMS_INTEGRITY	A resource manager integrity constraint check failed.
DDTMS_LOG_FAIL	A write operation to the transaction log failed.
DDTMS_PART_SERIAL	A resource manager serialization check failed.
DDTMS_PART_TIMEOUT	The timeout specified by a resource manager expired.
DDTMS_SEG_FAIL	A process or image terminated.
DDTMS_SERIALIZATION	A DECdtm transaction manager serialization check failed.
DDTMS_SYNC_FAIL	The transaction was not globally synchronized.
DDTMS_TIMEOUT	The timeout specified on \$START_TRANS expired.
DDTMS_UNKNOWN	The reason is unknown.
DDTMS_VETOED	A resource manager was unable to commit the transaction.

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



VM-0456A-AI

#### **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 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: `transaction_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. Those currently defined are shown in Table `SYS1-1`. The default value for this argument is `DDTMS_ABORTED`.

## Description

The Abort Transaction service ends a transaction by aborting it. The `DECdtm` transaction manager instructs all the resource managers participating in the transaction to abort the transaction operations so that none of those operations ever takes any effect.

`$ABORT_TRANS` must be called from the process that started the transaction.

`$ABORT_TRANS` does not complete successfully until all quotas allocated for the transaction by calls on the local node to `DECdtm` services have been returned.

`$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) 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).

## System Service Descriptions

### \$ABORT\_TRANS

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.

#### Required Access or Privileges

None

#### Required Quotas

ASTLM

#### Related Services

\$ABORT\_TRANSW, \$SEND\_TRANS, \$SEND\_TRANSW, \$START\_TRANS, \$START\_TRANSW

### Condition Values Returned

SS\$_NORMAL	If this was returned in R0, the request was successfully queued. If it was returned in the I/O status block, the service completed successfully.
SS\$_SYNCH	The service completed successfully and synchronously (returned only if the DDTM\$M_SYNC flag is set).
SS\$_ACCVIO	An argument was not accessible by the caller.
SS\$_BADPARAM	The options flags were invalid.
SS\$_BADREASON	The abort reason code was invalid.
SS\$_CURTIDCHANGE	The <b>tid</b> 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	Not enough arguments were supplied.
SS\$_INSMEM	There was insufficient system dynamic memory for the operation.
SS\$_NOCURTID	An attempt was made to abort the default transaction (the <b>tid</b> 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\$_NOSUCHTID	A transaction with the specified transaction identifier does not exist.
SS\$_NOTORIGIN	The calling process did not start the transaction.
SS\$_TPDISABLED	The TP_SERVER process was not running on the local node.
SS\$_WRONGSTATE	The calling process had already called either \$ABORT_TRANS or \$SEND_TRANS to end the transaction, and processing had not completed.



## \$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]]

### C Prototype

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

## System Service Descriptions

### \$ACQUIRE\_GALAXY\_LOCK (Alpha Only)

---

## \$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, refer to the *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.

## System Service Descriptions \$ACQUIRE\_GALAXY\_LOCK (Alpha Only)

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 SSS\_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 SSS\_NOWAIT is returned.

Due to system events such as 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 SSS\_BROKEN. If GLOCK\$C\_NOBREAK is specified, \$ACQUIRE\_GALAXY\_LOCK will not break the lock ownership and returns SSS\_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

SSS_NORMAL	Normal completion.
SSS_BADPARAM	Bad parameter value.
SSS_BROKEN	Lock acquired after lock ownership was broken.
SSS_IVLOCKID	Invalid lock id.
SSS_IVLOCKOP	Invalid lock operation.
SSS_IVLOCKTBL	Invalid lock table.
SSS_LOCK_TIMEOUT	Failed to acquire lock; request has timed out.
SSS_NOBREAK	Failed to acquire lock; lock ownership is broken.
SSS_NOWAIT	Failed to acquire lock; NOWAIT was specified.

## \$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:

## System Service Descriptions \$ADD HOLDER

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 <b>holder</b> 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.

## System Service Descriptions

### \$ADD\_HOLDER

SS\$_INSMEM	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 *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]
```

#### C 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.

## System Service Descriptions

### \$ADD\_IDENT

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 <b>name</b> argument cannot be read by the caller, or the <b>resid</b> 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\$INSMEM	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 *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]
```

#### C 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 PRXSM\_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
PRXSM_BYPASS_EXPAND	The service should not convert the node name specified in the <b>rem_node</b> 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.
PRXSM_DEFAULT	The user name specified by the <b>local_user</b> argument is the default user for the proxy. If this flag is not specified, the user name specified by the <b>local_user</b> argument is added to the proxy record's local user list.

## System Service Descriptions

### \$ADD\_PROXY

Symbolic Name	Description
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 <b>rem_node</b> , <b>rem_user</b> , <b>local_user</b> , or <b>flags</b> argument cannot be read by the service.
SS\$ _BADPARAM	An invalid flag was specified in the <b>flags</b> argument.
SS\$ _BADBUFLEN	The length of the <b>rem_node</b> , <b>rem_user</b> , or <b>local_user</b> 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 <b>local_user</b> argument already exists in the proxy record's local user list.
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.

**System Service Descriptions**  
**\$ADD\_PROXY**

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 Description section.

#### 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 Description section.

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

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

### 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 <b>newadr</b> 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 pages (on VAX systems) or pagelets (on Alpha systems) and returns the new value to the caller. The working set limit specifies the maximum number of process pages or pagelets that can be resident in physical memory.

On Alpha 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 pages (on VAX systems) or pagelets (on Alpha systems) 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 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 pages (on VAX systems) or pagelets (on Alpha systems), returned by \$ADJWSL. The **wsetlm** argument is the 32-bit address (on VAX systems) or the 32- or 64-bit address (on Alpha systems) 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 pages (on VAX systems) or pagelets (rounded up or down to a whole page count on Alpha systems) and returns the new value to the caller. The working set limit specifies the maximum number of process pages that can be resident in physical memory.

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

## System Service Descriptions

### \$ALLOC

#### 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\$_DEV MOUNT	The specified device is currently mounted and cannot be allocated, or the device is a mailbox.
SS\$_DEV OFFLINE	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.

## System Service Descriptions

### \$ASCEFC

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 *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.

## System Service Descriptions

### \$ASCEFC

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

---

†VAX specific



---

## \$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 systems, this service accepts 64-bit addresses.

#### Format

SYS\$ASCTIM [timlen] ,timbuf ,[timadr] ,[cvtfllg]

#### C Prototype

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

#### Arguments

##### timlen

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

Length (in bytes) of the ASCII string returned by \$ASCTIM. The **timlen** argument is the 32-bit address (on VAX systems) or the 32- or 64-bit address (on Alpha systems) 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 (Alpha)  
mechanism: by 32-bit descriptor (VAX)

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 (on Alpha systems) of a character string descriptor pointing to the buffer.

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

##### timadr

OpenVMS usage: date\_time  
type: quadword  
access: read only  
mechanism: by 32- or 64-bit reference (Alpha)  
mechanism: by 32-bit reference (VAX)

Time value that \$ASCTIM is to convert. The **timadr** argument is the 32-bit address (on VAX systems) or the 32- or 64-bit address (on Alpha systems) 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.

## System Service Descriptions

### \$ASCTIM

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 SSS\_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

Field	Length (Bytes)	Contents	Range of Values
.	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 **cvtfldg** and **timbuf** arguments are as follows:

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

**Required Access or Privileges**

None

**Required Quota**

None

**Related Services**

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

**Condition Values Returned**

SS\$NORMAL	The service completed successfully.
SS\$BUFFEROVF	The buffer length specified in the <b>timbuf</b> 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 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 (Alpha)

mechanism: by 32-bit descriptor-fixed-length string descriptor (VAX)

Identifier name translated when \$ASCTOID completes execution. The **name** argument is the 32- or 64-bit address (on Alpha systems) or the 32-bit address (on VAX systems) 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 (Alpha)

mechanism: by 32-bit reference (VAX)

Identifier value resulting when \$ASCTOID completes execution. The **id** argument is the 32- or 64-bit address (on Alpha systems) or the 32-bit address (on VAX systems) 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 (Alpha)

mechanism: by 32-bit reference (VAX)

Attributes associated with the identifier returned in **id** when \$ASCTOID completes execution. The **attrib** argument is the 32- or 64-bit address (on Alpha systems) or the 32-bit address (on VAX systems) 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 SKGBDEF 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 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 <b>name</b> argument cannot be read by the caller, or the <b>id</b> or <b>attrib</b> arguments cannot be written by the caller.

## System Service Descriptions

### \$ASCTOID

SS\$_INSMEM	The process dynamic memory is insufficient for opening the rights database.
SS\$_IIDENT	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 *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 systems, this service accepts 64-bit addresses.

#### Format

SYS\$ASCUTC [timlen] ,timbuf ,[utcadr] ,[cvtfllg]

#### C Prototype

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

#### Arguments

##### timlen

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

Length (in bytes) of the ASCII string returned by \$ASCUTC. The **timlen** argument is the 32-bit address (on VAX systems) or the 32- or 64-bit address (on Alpha systems) 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 (Alpha)  
 mechanism: by 32-bit descriptor-fixed-length string descriptor (VAX)

Buffer into which \$ASCUTC writes the ASCII string. The **timbuf** argument is the 32-bit address (on VAX systems) or the 32- or 64-bit address (on Alpha systems) of a character string descriptor pointing to the buffer. The buffer length specified in the **timbuf** argument, together with the **cvtfllg** 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 (Alpha)  
 mechanism: by 32-bit reference (VAX)

Time value that \$ASCUTC is to convert. The **timadr** argument is the 32-bit address (on VAX systems) or the 32- or 64-bit address (on Alpha systems) 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.

## System Service Descriptions

### \$ASCUTC

#### **cvtfllg**

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 **cvtfllg** 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 SSS\_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
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 **cvtfllg** and **timbuf** arguments are as follows:



## System Service Descriptions \$ASCUTC

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 <b>timbuf</b> 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.

## System Service Descriptions

### \$ASSIGN

---

## \$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 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 (Alpha)

mechanism: by 32-bit descriptor-fixed-length string descriptor (VAX)

Name of the device to which \$ASSIGN is to assign a channel. The **devnam** argument is the 32- or 64-bit address (on Alpha systems) or the 32-bit address (on VAX systems) 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 (Alpha)

mechanism: by 32-bit reference (VAX)

Number of the channel that is assigned. The **chan** argument is the 32- or 64-bit address (on Alpha systems) or the 32-bit address (on VAX systems) 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, refer to the section on access modes in the *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 (Alpha)  
 mechanism: by 32-bit descriptor-fixed-length string descriptor (VAX)

Logical name of the mailbox to be associated with the device. The **mbxnam** argument is the 32- or 64-bit address (on Alpha systems) or the 32-bit address (on VAX systems) 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 *OpenVMS I/O User's Reference Manual*.

**flags**

OpenVMS usage: mask\_longword  
 type: longword (unsigned)  
 access: read only  
 mechanism: by value

## System Service Descriptions

### \$ASSIGN

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, refer to the *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. Refer to the *OpenVMS Programming Concepts Manual* for additional information.

#### 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\$_CONNCFAIL	For network operations, the connection to a network object timed out or failed.

## System Service Descriptions

### \$ASSIGN

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\$_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 SY\$\$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.

## System Service Descriptions

### \$ASSIGN

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 Compaq.
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 Compaq.

## System Service Descriptions

### \$AUDIT\_EVENT

Symbolic Name	Description
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 Compaq.

#### 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.

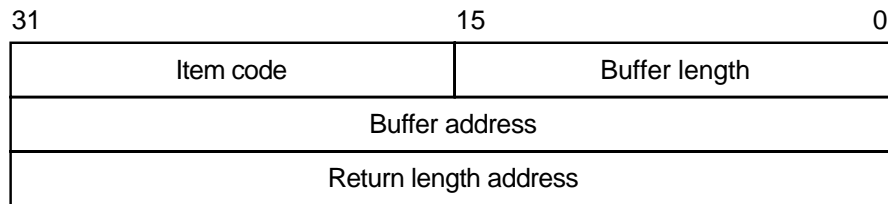
The item list for all calls to \$AUDIT\_EVENT must include the following item codes:

- NSA\$\_EVENT\_TYPE (see Table SYS1-2)
- NSA\$\_EVENT\_SUBTYPE (see Table SYS1-2)
- 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.



## System Service Descriptions \$AUDIT\_EVENT

The item list is a standard format item list. The following diagram depicts the general structure of an item descriptor:



ZK-5186A-GE

The following table defines the item descriptor fields:

Descriptor Field	Definition
Buffer length	A word specifying the length (in bytes) of the buffer; 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 Compaq. You must specify 0.

See the Item Codes section 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 SSS\_EVTNOTENAB. In addition, the caller must either make use of the **astadr** argument or use the \$AUDIT\_EVENTW service before attempting to access **audsts**.

## System Service Descriptions

### \$AUDIT\_EVENT

#### **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**

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**

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**

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**

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**

NSA\$\_EVENT\_SUBTYPE is a longword value specifying an event message subtype. See Table SYS1-2 for a list of valid event subtypes.

#### **NSA\$\_EVENT\_TYPE**

NSA\$\_EVENT\_TYPE is a longword value specifying an event message type. See Table SYS1-2 for a list of valid event types.

Table SYS1-2 Description of \$AUDIT\_EVENT Types and Subtypes

Symbol of Event Type	Meaning
NSASC_MSG_AUDIT	Systemwide change to auditing
<b>Subtype and Meaning</b>	
NSASC_AUDIT_DISABLED	Audit events disabled
NSASC_AUDIT_ENABLED	Audit events enabled
NSASC_AUDIT_INITIATE	Audit server startup
NSASC_AUDIT_TERMINATE	Audit server shutdown
NSASC_AUDIT_LOG_FINAL	Final entry in audit log (forward link)
NSASC_AUDIT_LOG_FIRST	First entry in audit log (backward link)
NSASC_MSG_BREAKIN	Break-in attempt detected
<b>Subtype and Meaning</b>	
NSASC_DETACHED	Detached process
NSASC_DIALUP	Dialup interactive process
NSASC_LOCAL	Local interactive process
NSASC_NETWORK	Network server process
NSASC_REMOTE	Interactive process from another network node
NSASC_MSG_CONNECTION	Logical link connection or termination
<b>Subtype and Meaning</b>	
NSASC_CNX_ABORT	Connection aborted
NSASC_CNX_ACCEPT	Connection accepted
NSASC_CNX_DECNET_CREATE	DECnet for OpenVMS logical link created
NSASC_CNX_DECNET_DELETE	DECnet for OpenVMS logical link disconnected
NSASC_CNX_DISCONNECT	Connection disconnected
NSASC_CNX_IPC_CLOSE	Interprocess communication association closed
NSASC_CNX_IPC_OPEN	Interprocess communication association opened
NSASC_CNX_REJECT	Connection rejected
NSASC_CNX_REQUEST	Connection requested
NSASC_CNX_INC_REQUEST	Incoming connection requested
NSASC_CNX_INC_ACCEPT	Incoming connection accepted
NSASC_CNX_INC_REJECT	Incoming connection rejected
NSASC_CNX_INC_DISCONNECT	Incoming connection disconnected
NSASC_CNX_INC_ABORT	Incoming connection aborted
NSASC_MSG_INSTALL	Use of the Install utility (INSTALL)
<b>Subtype and Meaning</b>	
NSASC_INSTALL_ADD	Known image installed
NSASC_INSTALL_REMOVE	Known image deleted
NSASC_MSG_LOGFAIL	Login failure

(continued on next page)

## System Service Descriptions

### \$AUDIT\_EVENT

Table SYS1-2 (Cont.) Description of \$AUDIT\_EVENT Types and Subtypes

Symbol of Event Type	Meaning
<b>Subtype and Meaning</b>	
NSASC_BATCH	Batch process
NSASC_DETACHED	Detached process
NSASC_DIALUP	Dialup interactive process
NSASC_LOCAL	Local interactive process
NSASC_NETWORK	Network server process
NSASC_REMOTE	Interactive process from another network node
NSASC_SUBPROCESS	Subprocess
NSASC_MSG_LOGIN	Successful login
<b>Subtype and Meaning</b>	
See subtypes for NSASC_MSG_LOGFAIL	
NSASC_MSG_LOGOUT	Successful logout
<b>Subtype and Meaning</b>	
See subtypes for NSASC_MSG_LOGFAIL	
NSASC_MSG_MOUNT	Volume mount or dismount
<b>Subtype and Meaning</b>	
NSASC_VOL_DISMOUNT	Volume dismount
NSASC_VOL_MOUNT	Volume mount
NSASC_MSG_NCP	Modification to network configuration database
<b>Subtype and Meaning</b>	
NSASC_NCP_COMMAND	Network Control Program (NCP) command issued
NSASC_MSG_NETPROXY	Modification to network proxy database
<b>Subtype and Meaning</b>	
NSASC_NETPROXY_ADD	Record added to network proxy database
NSASC_NETPROXY_DELETE	Record removed from network proxy database
NSASC_NETPROXY_MODIFY	Record modified in network proxy database
NSASC_MSG_OBJ_ACCESS	Object access attempted
<b>Subtype and Meaning</b>	
NSASC_OBJ_ACCESS	Object access attempted
NSASC_MSG_OBJ_CREATE	Object created
<b>Subtype and Meaning</b>	
NSASC_OBJ_CREATE	Object created

(continued on next page)

Table SYS1-2 (Cont.) Description of \$AUDIT\_EVENT Types and Subtypes

Symbol of Event Type	Meaning
NSASC_MSG_OBJ_DEACCESS	Object deaccessed
<b>Subtype and Meaning</b>	
NSASC_OBJ_DEACCESS	Object deaccessed
NSASC_MSG_OBJ_DELETE	Object deleted
<b>Subtype and Meaning</b>	
NSASC_OBJ_DELETE	Object deleted
NSASC_MSG_PROCESS	Process control system service issued
<b>Subtype and Meaning</b>	
NSASC_PRC_CANWAK	Process wakeup canceled
NSASC_PRC_CREPRC	Process created
NSASC_PRC_DELPRC	Process deleted
NSASC_PRC_FORCEX	Process exit forced
NSASC_PRC_GETJPI	Process information gathered
NSASC_PRC_GRANTID	Process identifier granted
NSASC_PRC_RESUME	Process resumed
NSASC_PRC_REVOKID	Process identifier revoked
NSASC_PRC_SCHDWK	Process wakeup scheduled
NSASC_PRC_SETPRI	Process priority altered
NSASC_PRC_SIGPRC	Process exception issued
NSASC_PRC_SUSPND	Process suspended
NSASC_PRC_WAKE	Process wakeup issued
NSASC_PRC_PRCTERM	Process termination notification requested
NSASC_MSG_PRVAUD	Attempt to use privilege
<b>Subtype and Meaning</b>	
NSASC_PRVAUD_FAILURE	Unsuccessful use of privilege
NSASC_PRVAUD_SUCCESS	Successful use of privilege
NSASC_MSG_RIGHTSDB	Modification to rights database
<b>Subtype and Meaning</b>	
NSASC_RDB_ADD_ID	Identifier added to rights database
NSASC_RDB_CREATE	Rights database created
NSASC_RDB_GRANT_ID	Identifier given to user
NSASC_RDB_MOD_HOLDER	List of identifier holders modified
NSASC_RDB_MOD_ID	Identifier name or attributes modified
NSASC_RDB_REM_ID	Identifier removed from rights database
NSASC_RDB_REVOKE_ID	Identifier revoked from user
NSASC_MSG_SYSGEN	Modification of a system parameter using the System Generation utility (SYSGEN)

(continued on next page)

## System Service Descriptions

### \$AUDIT\_EVENT

Table SYS1–2 (Cont.) Description of \$AUDIT\_EVENT Types and Subtypes

Symbol of Event Type	Meaning
<b>Subtype and Meaning</b>	
NSASC_SYSGEN_SET	System parameter modified
NSASC_MSG_SYSTIME	Modification to system time
<b>Subtype and Meaning</b>	
NSASC_SYSTIM_SET	System time set
NSASC_SYSTIM_CAL	System time calibrated
NSASC_MSG_SYSUAF	Modification to system user authorization file (SYSUAF)
<b>Subtype and Meaning</b>	
NSASC_SYSUAF_ADD	Record added to SYSUAF
NSASC_SYSUAF_COPY	Record copied in SYSUAF
NSASC_SYSUAF_DELETE	Record deleted from SYSUAF
NSASC_SYSUAF_MODIFY	Record modified in SYSUAF
NSASC_SYSUAF_RENAME	Record renamed in SYSUAF

#### NSA\$\_FIELD\_NAME

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

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

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 SYSSMESSAGE:.EXE. By default, \$FORMAT\_AUDIT uses the resident system message file.

#### NSA\$\_NEW\_DATA

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

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**

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**

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**

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**

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**

NSA\$\_SUPPRESS is a longword bit mask 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 section provides a list of additional item codes that are valid as an item descriptor in the **itmlst** argument.

## System Service Descriptions

### \$AUDIT\_EVENT

#### **NSA\$\_ACCESS\_DESIRED**

NSA\$\_ACCESS\_DESIRED is a longword value specifying the access request mask as defined in \$ARMDEF.

#### **NSA\$\_ACCESS\_MODE**

NSA\$\_ACCESS\_MODE is a byte value specifying an access mode associated with the event.

#### **NSA\$\_ACCOUNT**

NSA\$\_ACCOUNT is a string of 1 to 32 characters specifying the account name associated with the event.

#### **NSA\$\_ASSOCIATION\_NAME**

NSA\$\_ASSOCIATION\_NAME is a string of 1 to 256 characters specifying an association name.

#### **NSA\$\_COMMAND\_LINE**

NSA\$\_COMMAND\_LINE is a string of 1 to 2048 characters specifying a command line.

#### **NSA\$\_CONNECTION\_ID**

NSA\$\_CONNECTION\_ID is a longword value specifying a connection identification.

#### **NSA\$\_DECNET\_LINK\_ID**

NSA\$\_DECNET\_LINK\_ID is a longword value specifying a DECnet for OpenVMS logical link identification.

#### **NSA\$\_DECNET\_OBJECT\_NAME**

NSA\$\_DECNET\_OBJECT\_NAME is a string of 1 to 16 characters specifying a DECnet for OpenVMS object name.

#### **NSA\$\_DECNET\_OBJECT\_NUMBER**

NSA\$\_DECNET\_OBJECT\_NUMBER is a longword value specifying a DECnet for OpenVMS object number.

#### **NSA\$\_DEFAULT\_USERNAME**

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**

NSA\$\_DEVICE\_NAME is a string of 1 to 64 characters specifying the name of the device where the volume resides.

#### **NSA\$\_DIRECTORY\_ENTRY**

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**

NSA\$\_DIRECTORY\_ID is an array of three words specifying the directory file identification.

#### **NSA\$\_DISMOUNT\_FLAGS**

NSA\$\_DISMOUNT\_FLAGS is a longword value specifying the dismount flags that are defined by the \$DMTDEF macro in STARLET.



**NSA\$\_EFC\_NAME**

NSA\$\_EFC\_NAME is a string of 1 to 16 characters specifying the event flag cluster name.

**NSA\$\_FILE\_ID**

NSA\$\_FILE\_ID is an array of three words specifying the file identification.

**NSA\$\_FINAL\_STATUS**

NSA\$\_FINAL\_STATUS is a longword value specifying the successful or unsuccessful status that caused the auditing facility to be invoked.

**NSA\$\_HOLDER\_NAME**

NSA\$\_HOLDER\_NAME is a string of 1 to 32 characters specifying the name of the user holding the identifier.

**NSA\$\_HOLDER\_OWNER**

NSA\$\_HOLDER\_OWNER is a longword value specifying the owner (UIC) of the holder.

**NSA\$\_ID\_ATTRIBUTES**

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**

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**

NSA\$\_ID\_NAME is a string of 1 to 32 characters specifying the name of the identifier.

**NSA\$\_ID\_NEW\_ATTRIBUTES**

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**

NSA\$\_ID\_NEW\_NAME is a string of 1 to 32 characters specifying the new name of the identifier.

**NSA\$\_ID\_NEW\_VALUE**

NSA\$\_ID\_NEW\_VALUE is a longword value specifying the new value of the identifier.

**NSA\$\_ID\_VALUE**

NSA\$\_ID\_VALUE is a longword value specifying the value of the identifier.

**NSA\$\_ID\_VALUE\_ASCII**

NSA\$\_ID\_VALUE\_ASCII is a longword specifying the value of the identifier.

**NSA\$\_IMAGE\_NAME**

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**

NSA\$\_INSTALL\_FILE is a string of 1 to 255 characters specifying the name of the installed file.

## System Service Descriptions

### \$AUDIT\_EVENT

#### **NSA\$\_INSTALL\_FLAGS**

NSA\$\_INSTALL\_FLAGS is a longword value specifying the INSTALL flags. They correspond to qualifiers for the Install utility; for example, NSASM\_INS\_EXECUTE\_ONLY.

#### **NSA\$\_LNM\_PARENT\_NAME**

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**

NSA\$\_LNM\_TABLE\_NAME is a string of 1 to 31 characters specifying the name of the logical name table.

#### **NSA\$\_LOCAL\_USERNAME**

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**

NSA\$\_LOGICAL\_NAME is a string of 1 to 255 characters specifying the logical name associated with the device.

#### **NSA\$\_MAILBOX\_UNIT**

NSA\$\_MAILBOX\_UNIT is a longword value specifying the mailbox unit number.

#### **NSA\$\_MATCHING\_ACE**

NSA\$\_MATCHING\_ACE is an array of bytes specifying the ACE granting or denying access.

#### **NSA\$\_MOUNT\_FLAGS**

NSA\$\_MOUNT\_FLAGS is a quadword value specifying mount flags that are defined by the \$MNTDEF macro in STARLET.

#### **NSA\$\_NEW\_IMAGE\_NAME**

NSA\$\_NEW\_IMAGE\_NAME is a string of 1 to 1024 characters specifying the name of the new image.

#### **NSA\$\_NEW\_OWNER**

NSA\$\_NEW\_OWNER is a longword value specifying the new process owner (UIC).

#### **NSA\$\_NEW\_PRIORITY**

NSA\$\_NEW\_PRIORITY is a longword value specifying the new process priority.

#### **NSA\$\_NEW\_PRIVILEGES**

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**

NSA\$\_NEW\_PROCESS\_ID is a longword value specifying the new process identification.

#### **NSA\$\_NEW\_PROCESS\_NAME**

NSA\$\_NEW\_PROCESS\_NAME is a string of 1 to 15 characters specifying the name of the new process.

**NSA\$ \_NEW\_PROCESS\_OWNER**

NSA\$ \_NEW\_PROCESS\_OWNER is a longword value specifying the owner (UIC) of the new process.

**NSA\$ \_NEW\_USERNAME**

NSA\$ \_NEW\_USERNAME is a string of 1 to 32 characters specifying the new user name.

**NSA\$ \_OBJECT\_CLASS**

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**

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**

NSA\$ \_OBJECT\_MAX\_CLASS is a 20-byte record specifying the maximum access classification of the object.

**NSA\$ \_OBJECT\_MIN\_CLASS**

NSA\$ \_OBJECT\_MIN\_CLASS is a 20-byte record specifying the minimum access classification of the object.

**NSA\$ \_OBJECT\_NAME**

NSA\$ \_OBJECT\_NAME is a string of 1 to 255 characters specifying an object's name.

**NSA\$ \_OBJECT\_NAME\_2**

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**

NSA\$ \_OBJECT\_OWNER is a longword value specifying the UIC or general identifier of the process causing the auditable event.

**NSA\$ \_OBJECT\_PROTECTION**

NSA\$ \_OBJECT\_PROTECTION is a word, or an array of four longwords, specifying the UIC-based protection of the object.

**NSA\$ \_OLD\_PRIORITY**

NSA\$ \_OLD\_PRIORITY is a longword value specifying the former process priority.

**NSA\$ \_OLD\_PRIVILEGES**

NSA\$ \_OLD\_PRIVILEGES is a quadword privilege mask specifying the former privileges. The \$PRVDEF macro defines the list of available privileges.

**NSA\$ \_PARAMS\_INUSE**

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**

NSA\$ \_PARAMS\_WRITE is a string of 1 to 255 characters specifying the file name for the SYSGEN command WRITE.

## System Service Descriptions

### \$AUDIT\_EVENT

#### **NSA\$\_PARENT\_ID**

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**

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**

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**

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**

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**

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**

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**

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**

NSA\$\_PROCESS\_ID is a longword value specifying the PID of the process causing the auditable event.

#### **NSA\$\_PROCESS\_NAME**

NSA\$\_PROCESS\_NAME is a string of 1 to 15 characters specifying the process name that caused the auditable event.

#### **NSA\$\_REM\_ASSOCIATION\_NAME**

NSA\$\_REM\_ASSOCIATION\_NAME is a string of 1 to 256 characters specifying the interprocess communication (IPC) remote association name.

#### **NSA\$\_REMOTE\_LINK\_ID**

NSA\$\_REMOTE\_LINK\_ID is a longword value specifying the remote logical link ID.

**NSA\$\_REMOTE\_NODE\_FULLNAME**

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**

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**

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**

NSA\$\_REMOTE\_USERNAME is a string of 1 to 32 characters specifying the user name of the remote process.

**NSA\$\_REQUEST\_NUMBER**

NSA\$REQUEST\_NUMBER is a longword value specifying the request number associated with the system service call.

**NSA\$RESOURCE\_NAME**

NSA\$RESOURCE\_NAME is a string of 1 to 32 characters specifying the lock resource name.

**NSA\$SECTION\_NAME**

NSA\$SECTION\_NAME is a string of 1 to 42 characters specifying the global section name.

**NSA\$SNAPSHOT\_BOOTFILE**

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**

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**

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**

NSA\$SOURCE\_PROCESS\_ID is a longword value specifying the process identification of the process originating the request.

**NSA\$SUBJECT\_CLASS**

NSA\$SUBJECT\_CLASS is a 20-byte record specifying the current access class of the process causing the auditable event.

**NSA\$SUBJECT\_OWNER**

NSA\$SUBJECT\_OWNER is a longword value specifying the owner (UIC) of the process causing the event.

## System Service Descriptions

### \$AUDIT\_EVENT

#### **NSA\$\_SYSTEM\_ID**

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**

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**

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**

NSA\$\_SYSTIM\_NEW is a quadword value specifying the new system time.

#### **NSA\$\_SYSTIM\_OLD**

NSA\$\_SYSTIM\_OLD is a quadword value specifying the old system time.

#### **NSA\$\_TARGET\_DEVICE\_NAME**

NSA\$\_TARGET\_DEVICE\_NAME is a string of 1 to 64 characters specifying the target device name.

#### **NSA\$\_TARGET\_PROCESS\_CLASS**

NSA\$\_TARGET\_PROCESS\_CLASS is a 20-byte record specifying the target process classification.

#### **NSA\$\_TARGET\_PROCESS\_ID**

NSA\$\_TARGET\_PROCESS\_ID is a longword value specifying the target process identifier (PID).

#### **NSA\$\_TARGET\_PROCESS\_NAME**

NSA\$\_TARGET\_PROCESS\_NAME is a string of 1 to 64 characters specifying the target process name.

#### **NSA\$\_TARGET\_PROCESS\_OWNER**

NSA\$\_TARGET\_PROCESS\_OWNER is a longword value specifying the target owner (UIC).

#### **NSA\$\_TARGET\_USERNAME**

NSA\$\_TARGET\_USERNAME is a string of 1 to 32 characters specifying the target process user name.

#### **NSA\$\_TERMINAL**

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**

NSA\$\_TIME\_STAMP is a quadword value specifying the time when the event occurred.

#### **NSA\$\_TRANSPORT\_NAME**

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**

NSA\$\_UAF\_ADD is a string of 1 to 32 characters specifying the name of the authorization record being added.

**NSA\$\_UAF\_COPY**

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**

NSA\$\_UAF\_DELETE is a string of 1 to 32 characters specifying the name of the authorization record being removed.

**NSA\$\_UAF\_MODIFY**

NSA\$\_UAF\_MODIFY is a string of 1 to 32 characters specifying the name of the authorization record being modified.

**NSA\$\_UAF\_RENAME**

NSA\$\_UAF\_RENAME is a string of 1 to 32 characters specifying the name of the authorization record being renamed.

**NSA\$\_UAF\_SOURCE**

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**

NSA\$\_USERNAME is a string of 1 to 32 characters specifying the user name of the process causing the auditable event.

**NSA\$\_VOLUME\_NAME**

NSA\$\_VOLUME\_NAME is a string of 1 to 15 characters specifying a volume name.

**NSA\$\_VOLUME\_SET\_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 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

## System Service Descriptions

### \$AUDIT\_EVENT

#### 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\$_BADBUFLN	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\$_BADITMCD	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, refer to 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 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 (Alpha)

mechanism: by 32-bit descriptor-fixed-length string descriptor (VAX)

Buffer that holds the ASCII time to be converted. The **timbuf** argument specifies the 32-bit address (on VAX systems) or the 32- or 64-bit address (on Alpha systems) 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 (Alpha)

mechanism: by 32-bit reference (VAX)

Time value that \$BINTIM has converted. The **timadr** argument is the 32-bit address (on VAX systems) or the 32- or 64-bit address (on Alpha systems) 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.  
For absolute time values, the \$BINTIM service supplies the current system date and time for nonspecified fields. Trailing fields can be truncated. If leading fields are omitted, you must specify the punctuation (hyphens, blanks, colons, periods). For example, the following string results in an absolute time of 12:00 on the current day:

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

## System Service Descriptions

### \$BINTIM

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, \$SCANWAK, \$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-1994 04:15:28.00.

Input to \$BINTIM	\$ASCTIM Output String
-- :50	30-DEC-1994 04:50:28.00
--1994 0:0:0.0	29-DEC-1994 00:00:00.00
30-DEC-1994 12:32:1.1161	30-DEC-1994 12:32:01.12
29-DEC-1994 16:35:0.0	29-DEC-1994 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 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 (Alpha)  
mechanism: by 32-bit descriptor–fixed-length string descriptor (VAX)

Buffer that holds the ASCII time to be converted. The **timbuf** argument specifies the 32-bit address (on VAX systems) or the 32- or 64-bit address (on Alpha systems) 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 (Alpha)  
mechanism: by 32-bit reference (VAX)

Time value that \$BINUTC has converted. The **utcadr** argument is the 32-bit address (on VAX systems) or the 32- or 64-bit address (on Alpha systems) 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 SSS\_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.

## System Service Descriptions

### \$BINUTC

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]  
            [,timeout] [,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 timeout, 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 <b>sendto</b> 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 <b>sendto</b> argument.

**iosb**

OpenVMS usage: io\_status\_block  
 type: quadword (unsigned)  
 access: write only  
 mechanism: by reference

## System Service Descriptions

### \$BRKTHRU

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 *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 BRKSM\_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

## System Service Descriptions \$BRKTHRU

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 <b>carcon</b> argument. \$BRKTHRU sends screen-formatted messages to terminals with the DEC_CRT characteristic, and it sends messages formatted by <b>carcon</b> to those without the DEC_CRT characteristic. You set the DEC_CRT characteristic for the terminal by using the DCL command SET TERMINAL/DEC_CRT. A screen-formatted message is displayed at the top of the terminal screen, and the cursor is repositioned at the point it was prior to the broadcast message. However, the BRK\$V_ERASE_LINES and BRK\$M_BOTTOM flags also affect the display.
BRK\$M_BOTTOM	When BRK\$M_BOTTOM is specified and BRK\$M_SCREEN is also specified, \$BRKTHRU writes the message to the bottom of the terminal screen instead of the top. BRK\$M_BOTTOM is ignored if the BRK\$M_SCREEN flag is not set.
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.

## System Service Descriptions

### \$BRKTHRU

#### **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 <b>reqid</b> 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.

Class Name	Component
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.

**timeout**

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 **timeout** 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 **timeout** argument, \$BRKTHRU uses a default value of 0 seconds, which specifies infinite time (no timeout occurs).

The value specified by **timeout** 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 **timeout** 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.

## System Service Descriptions

### \$BRKTHRU

#### 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 IOS\_WRITEVBLK function code, together with the IOSM\_BREAKTHRU, IOSM\_CANCTRLO, and (optionally) IOSM\_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 to send a message to more than one terminal or to a terminal that is allocated to another user.

The calling process must have WORLD privilege to send a message to a specific user by specifying the BRKSC\_USERNAME symbolic code for the **sndtyp** argument.

#### 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, \$SENDERR, \$SENDJBC, \$SENDJBCW, \$SENDOPR



**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 TIMEOUT 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]  
              [,timeout] [,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 timeout, 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 SSS\_CANCEL if the I/O request is queued, or to SSS\_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.

## System Service Descriptions

### \$CANCEL

#### 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, \$SENDERR, \$SENDJBC, \$SENDJBCW, \$SENDOPR

#### Condition Values Returned

SS\$NORMAL	The service completed successfully.
SS\$EXQUOTA	The process has exceeded its buffered I/O limit (BIOLM) quota.
SS\$INSMEM	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.

## System Service Descriptions

### \$CANTIM

---

## \$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: longword (unsigned)  
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 longword 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, \$SCANWAK, \$GETTIM, \$NUMTIM, \$SCHDWK, \$SETIME, \$SETIMR

## **Condition Values Returned**

SS\$\_NORMAL

The service completed successfully.

## System Service Descriptions

### \$SCANWAK

---

## \$SCANWAK

### Cancel Wakeup

Removes all scheduled wakeup requests for a process from the timer queue, including those made by the caller or by other processes. The Schedule Wakeup (SSCHDWK) service makes scheduled wakeup requests.

#### Format

SYS\$SCANWAK [pidadr] ,[prcnam]

#### C Prototype

```
int sys$scanwak (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 for which wakeups are to be canceled. The **pidadr** argument is the address of a longword specifying 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.

##### prcnam

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

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

A process running on the local node can be identified with a 1- to 15-character string. To identify a process on a specific 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 operating system interprets the UIC group number of the calling process as part of the process name; the names of processes are unique to UIC groups. Because of this, you can use the **prcnam** argument only on behalf of processes in the same group as the calling process.

#### Description

The Cancel Wakeup service removes from the timer queue all scheduled wakeup requests for a process, including those made by the caller or by other processes. The Schedule Wakeup (SSCHDWK) service makes scheduled wakeup requests.

If the longword at address **pidadr** is 0, the PID of the target process is returned.



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

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

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

#### Required Access or Privileges

Depending on the operation, the calling process might need one of the listed privileges to use \$SCANWAK:

- You need GROUP privilege to cancel wakeups for processes in the same group that do not have the same UIC.
- You need WORLD privilege to cancel wakeups for any process in the system.

#### Required Quota

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

#### Related Services

\$ASCTIM, \$BINTIM, \$CANTIM, \$GETTIM, \$NUMTIM, \$SCHDWK, \$SETIME, \$SETIMR

### Condition Values Returned

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

## System Service Descriptions

### \$CHECK\_ACCESS

---

## \$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

## System Service Descriptions \$CHECK\_ACCESS

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

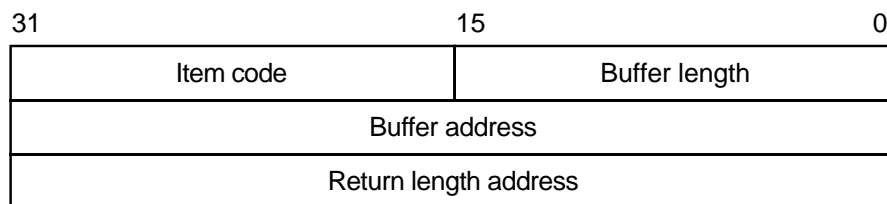
Name of the user attempting access. The **usrnam** argument is the address of a descriptor that points to a character string that contains the name of the user attempting to gain access to the specified object. The user name string can contain a maximum of 12 alphanumeric characters.

### 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 (CHPS\_END). This is shown in the following diagram:



ZK-5186A-GE

The following table defines the item descriptor fields:

Descriptor Field	Definition
Buffer length	A word containing a user-supplied integer specifying the length (in bytes) of the 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.

## System Service Descriptions

### \$CHECK\_ACCESS

Descriptor Field	Definition
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 <b>bufadr</b> . If the buffer pointed to by <b>bufadr</b> is used to pass information to \$CHECK_ACCESS, <b>retlenadr</b> 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, refer to the *OpenVMS Guide to System Security*.

## System Service Descriptions

### \$CHECK\_ACCESS

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.
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 (SPSLDEF). The following access modes and their symbols are defined in the system macro library (SPSLDEF). 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 NSAS\_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 NSAS\_EVENT\_TYPE and NSAS\_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 NSAS\_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.
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 Compaq.
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.

## System Service Descriptions

### \$CHECK\_ACCESS

Symbol	Meaning
CHPSV_SYSNAM	SYSNAM was used to gain the requested access.
CHPSV_GROUP	GROUP was used to gain the requested access.
CHPSV_WORLD	WORLD was used to gain the requested access.
CHPSV_PRMCEB	PRMCEB was used to gain the requested access.
CHPSV_UPGRADE	UPGRADE was used to gain the requested access.
CHPSV_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 SSS\_NORMAL status code; otherwise, \$CHECK\_ACCESS returns SSS\_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 CHPSV\_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 CHPSV\_CREATE flag to force an object creation audit instead of an object access audit. Similarly, the CHPSV\_DELETE flag forces an object deletion audit. The CHPSV\_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. Refer to the *OpenVMS I/O User's Reference Manual* for further information on access requirements for devices.

#### 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.

## System Service Descriptions

### **\$CHECK\_FEN (Alpha Only)**

---

### **\$CHECK\_FEN (Alpha Only)**

#### **Check Floating Point**

On Alpha systems, indicates whether floating point is enabled for the current image.

#### **Format**

SYS\$CHECK\_FEN

#### **C Prototype**

```
int sys$check_fen (void);
```

#### **Arguments**

None.

#### **Description**

The Check Floating Point service returns a Boolean value in R0 indicating whether floating point is 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.

#### **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] ,privadr ,[altprv] ,[flags] ,[itmlst] ,[audsts] ,[astadr]
                    ,[astprm]
```

#### C Prototype

```
int sys$check_privilege (unsigned int efn, struct _generic_64 *privadr, 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.

##### privadr

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 **privadr** 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 NSASM\_IDENTIFIER bit in the **flags** longword. The identifier structure is defined by the \$KGBDEF macro. The identifier attributes (KGBS) are reserved for future use and should be set to 0.

## System Service Descriptions

### \$CHECK\_PRIVILEGE

#### 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 `NSASM_AUTHPRIV`, `NSASM_IDENTIFIER`, and `NSASM_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 `NSASM_AUTHPRIV`, `NSASM_IDENTIFIER`, and `NSASM_PROCPRIV` are mutually exclusive; therefore, you can specify only one of these flag options.

Symbolic Name	Description
<code>NSASM_AUTHPRIV</code>	Checks the authorized privileges of the process instead of the current (active) privileges.
<code>NSASM_FLUSH</code>	Specifies that all messages in the audit server buffer be written to the audit log file.
<code>NSASM_IDENTIFIER</code>	Interprets the <b>prvadr</b> argument as the address of an identifier instead of a privilege mask.
<code>NSASM_INTERNAL</code>	Specifies that the <code>\$CHECK_PRIVILEGE</code> 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 <code>SECAUDTCB</code> bugcheck. This flag is reserved to Compaq.
<code>NSASM_MANDATORY</code>	Specifies that an audit is to be performed, regardless of system alarm and audit settings.
<code>NSASM_PROCPRIV</code>	Checks the permanent privileges of the process, instead of the privileges in the current (active) mask.

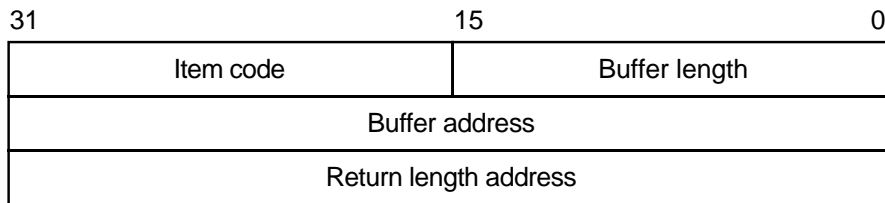
Symbolic Name	Description
NSASM_SERVER	<p>Indicates that the call originates in a TCB server process and that the event should be audited regardless of the state of a process-specific no-audit bit.</p> <p>Trusted servers use this flag to override the no-audit bit when they want to perform explicit auditing on behalf of a client process. This flag is reserved to Compaq.</p>

**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.



ZK-5186A-GE

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 Compaq. You should specify 0.

## System Service Descriptions

### \$CHECK\_PRIVILEGE

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 SSS\$\_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 SSS\$\_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\$_BADBUFLFN	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\$_BADITMCD	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 ,privadr ,[altprv] ,[flags] ,[itmlst] ,audsts ,[astadr]  
,[astprm]
```

#### C Prototype

```
int sys$check_privilegew (unsigned int efn, struct _generic_64 *privadr, 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);
```

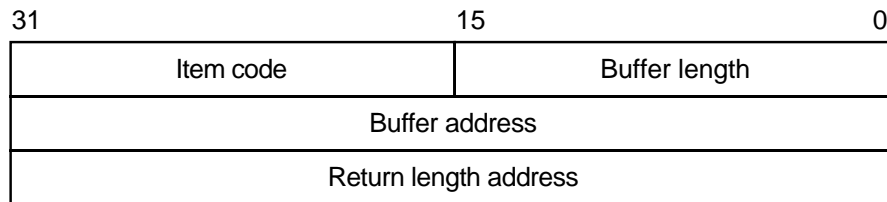
### Argument

#### 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:



ZK-5186A-GE

The following table defines the item descriptor fields:

Descriptor Field	Definition
Buffer length	A word containing a user-supplied integer specifying the length (in bytes) of the 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.

## System Service Descriptions

### \$CHKPRO

Descriptor Field	Definition
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 Compaq.

#### **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
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.

## System Service Descriptions

### \$CHKPRO

#### 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 (\$SHPDEF):

Symbol	Access
CHP\$V_ALTER	Accessor desires write access to object.
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 Compaq.
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



## System Service Descriptions

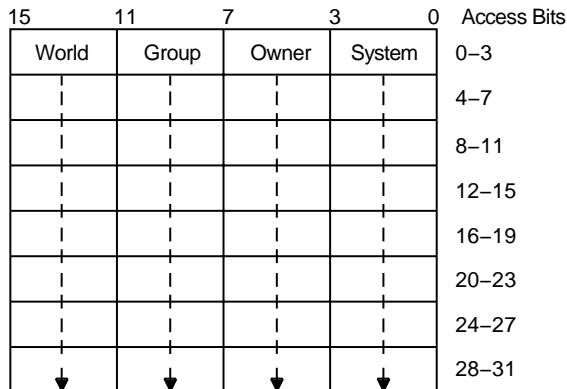
### \$CHKPRO

You can also obtain the values as masks with the appropriate bit set by using the prefix `CHPSM` rather than `CHPSV`. The symbols are defined in the system macro library (`$SCHPDEF`). The following symbols are used as offsets to the bits within the longword:

Symbol	Meaning
<code>CHPSV_SYSPRV</code>	<code>SYSPRV</code> was used to gain the requested access.
<code>CHPSV_GRPPRV</code>	<code>GRPPRV</code> was used to gain the requested access.
<code>CHPSV_BYPASS</code>	<code>BYPASS</code> was used to gain the requested access.
<code>CHPSV_READALL</code>	<code>READALL</code> was used to gain the requested access.
<code>CHPSV_OPER</code>	<code>OPER</code> was used to gain the requested access.
<code>CHPSV_GRPNAM</code>	<code>GRPNAM</code> was used to gain the requested access.
<code>CHPSV_SYSNAM</code>	<code>SYSNAM</code> was used to gain the requested access.
<code>CHPSV_GROUP</code>	<code>GROUP</code> was used to gain the requested access.
<code>CHPSV_WORLD</code>	<code>WORLD</code> was used to gain the requested access.
<code>CHPSV_PRMCEB</code>	<code>PRMCEB</code> was used to gain the requested access.
<code>CHPSV_UPGRADE</code>	<code>UPGRADE</code> was used to gain the requested access.
<code>CHPSV_DOWNGRADE</code>	<code>DOWNGRADE</code> was used to gain the requested access.

### CHPS\_PROT

A vector describing the object's SOGW protection mask. The following diagram depicts the format for describing the object's protection:



ZK-1704-GE

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 `CHPS_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 SSS\_NORMAL status code; otherwise, \$CHKPRO returns SSS\_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 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

## System Service Descriptions

### \$CHKPRO

requires that the caller be in executive or kernel mode or possess the AUDIT privilege.

Normally, \$CHKPRO generates an object access audit when an audit is required. The caller can specify the CHPSV\_CREATE flag to force an object creation audit instead of an object access audit. Similarly, the CHPSV\_DELETE flag forces an object deletion audit. The CHPS\_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 CHPS_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 CHPS_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 CHPS_ADDRIGHTS items were given.



## \$CLEAR\_SYSTEM\_EVENT (Alpha Only) 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

```
int 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.

## System Service Descriptions

### \$CLEAR\_SYSTEM\_EVENT (Alpha Only)

#### 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 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 of more arguments has an invalid value, such as an invalid handle.
SS\$NOSUCHOBJ	No request was found that matches the description supplied.

---

## \$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.

---

### Note

---

The explicit use of \$CLRAST is strongly discouraged, as it complicates synchronization issues and may lead to the unbounded consumption of stack space.

---

### Format

SY\$CLRAST

### Arguments

None.

### Description

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: access\_mode  
type: longword (unsigned)  
access: read only  
mechanism: 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$ceref (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\$_WASET	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 SSS\_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 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.

On Alpha systems, to conform to the OpenVMS calling standard, you must not omit the **arglst** argument.

On VAX systems, if no argument list is specified, the argument pointer (AP) contains a 0. However, to conform to the OpenVMS calling standard, you must not omit the **arglst** argument.

## System Service Descriptions

### \$CMEXEC

On Alpha and VAX systems, 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 **routine** 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 <b>arglst</b> 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 (Alpha Only)

### Change to Executive Mode with Quadword Argument List

On Alpha 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 SSS\_ACCVIO.

Alpha 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.

## System Service Descriptions

### \$CMEXEC\_64 (Alpha Only)

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 **routine\_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 <b>arglst</b> 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 SSS\_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.

## System Service Descriptions

### \$CMKRNL

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 SSS_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 SSS_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**

SSS_ACCVIO	The <b>arglst</b> argument or routine is not accessible.
SSS_BADPARAM	The routine specified is in a translated image.
SSS_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 (Alpha Only) Change to Kernel Mode with Quadword Argument List

On Alpha 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 SSS\_ACCVIO.

Alpha 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.

## System Service Descriptions \$CMKRNL\_64 (Alpha Only)

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 <b>arglst</b> argument or routine is not accessible.
SS\$_BADPARAM	The routine specified is in a translated image.
SS\$_NOCKRNL	The process does not have the privilege to change mode to kernel.
All other values	The routine executed returns all other values.

## System Service Descriptions

### \$CPU\_CAPABILITIES (Alpha Only)

---

## \$CPU\_CAPABILITIES (Alpha Only)

### Modify CPU User Capabilities

On Alpha 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 CAPSK\_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 CAPSM\_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 CAPSK\_ALL\_ACTIVE\_CPUS constant, however. If both constants are specified, CAPSK\_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



## System Service Descriptions \$CPU\_CAPABILITIES (Alpha Only)

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 cleared 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

## System Service Descriptions

### \$CPU\_CAPABILITIES (Alpha Only)

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 <b>cpu_id</b> but does not override the all active set behavior (CAP\$K_ALL_ACTIVE_CPUS). Specifying this bit 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 nonrunnable 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 <b>flags</b> is not specified, the kernel thread is checked to ensure that 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.

## System Service Descriptions \$CPU\_CAPABILITIES (Alpha Only)

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 SYSS\$CPU\_CAPABILITIES to modify CPU user capabilities.

No privileges are required if SYSS\$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 or 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.

## System Service Descriptions

### \$CPU\_TRANSITION (Alpha Only)

---

## \$CPU\_TRANSITION (Alpha Only)

### CPU Transition

On Alpha 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.

Refer to the *OpenVMS Alpha Partitioning and Galaxy Guide* for more information.

### 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 timeout);
```

### 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.

## System Service Descriptions \$CPU\_TRANSITION (Alpha Only)

Symbolic Name	Description
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.

### 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 SYIS\_MAX\_CPUS - 1. In a Galaxy, this CPU must be in the configure set of the local partition or not assigned to any other partitions. The symbolic values, CST\$K\_ANY\_LOCAL\_CPU can be used in certain types of transitions to allow the next available processor of the appropriate type be used instead of a specific one.

### node\_id

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

Identifier of the target Galaxy partition in CST\$K\_CPU\_ASSIGN, CST\$K\_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:

## System Service Descriptions

### \$CPU\_TRANSITION (Alpha Only)

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 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 SYSSSTARLET\_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

## System Service Descriptions \$CPU\_TRANSITION (Alpha Only)

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 SYSSCPU\_TRANSITION to modify CPU states.

### Related Services

\$CPU\_TRANSITIONW

## System Service Descriptions

### \$CPU\_TRANSITION (Alpha Only)

#### 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\$_NOCMKRNL	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 Only) CPU Transition and Wait

On Alpha 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, refer to the description of \$CPU\_TRANSITION in this manual.

This service accepts 64-bit addresses.

Refer to the *OpenVMS Alpha Partitioning and Galaxy Guide* for more information.

### 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 timeout);
```

## System Service Descriptions

### \$CREATE\_GALAXY\_LOCK (Alpha Only)

---

## \$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, refer to the *OpenVMS Alpha Partitioning and Galaxy Guide*.

#### Format

```
SYS$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: ASCIID string  
access: read  
mechanism: input by reference

The **name** parameter is a pointer to an ASCIID 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

## System Service Descriptions \$CREATE\_GALAXY\_LOCK (Alpha Only)

The size of the galaxy lock in bytes. Galaxy locks have two legal sizes. These values are returned by SYSSGET\_GALAXY\_LOCK\_SIZE. The value passed to SYSSCREATE\_GALAXY\_LOCK must be equal to the value passed to the call to SYSSCREATE\_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

## System Service Descriptions

### \$CREATE\_GALAXY\_LOCK (Alpha Only)

#### 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\$_NOSHMEN	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 OpenVMS Galaxy system services, refer to the *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: ASCIID string  
access: read  
mechanism: input by reference

The name parameter is a pointer to an ASCIID 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

## System Service Descriptions

### \$CREATE\_GALAXY\_LOCK\_TABLE (Alpha Only)

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:

World				Group				Owner				System			
D	E	W	R	D	E	W	R	D	E	W	R	D	E	W	R
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

ZK-1706-GE

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 SYSS\$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.

## System Service Descriptions

### \$CREATE\_GALAXY\_LOCK\_TABLE (Alpha Only)

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.

## System Service Descriptions

### \$CREATE\_BUFOBJ\_64 (Alpha Only)

---

## \$CREATE\_BUFOBJ\_64 (Alpha Only)

### Create Buffer Object

On Alpha 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 SYSSSTARLET\_C.TLB define the following symbols and their values for the four access modes:



## System Service Descriptions \$CREATE\_BUFOBJ\_64 (Alpha Only)

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
CBOSM_RETSVA	1	If set, returns the system virtual address in the <b>return_va_64</b> argument instead of the process virtual address range. (Valid for inner mode callers only.)
CBOSM_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.

## System Service Descriptions

### \$CREATE\_BUFOBJ\_64 (Alpha Only)

#### **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 the following:

- 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 `$CREATE_BUFOBJ_64` from an inner mode. If calling from user mode, the process must hold the rights identifier `VM$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 pages are charged against a systemwide limit, system parameter `MAXBOBMEM`.

#### **Related Services**

`$CRETVA_64`, `$DELETE_BUFOBJ`, `$EXPREG_64`

## Condition Values Returned

<code>SS\$_NORMAL</code>	The service completed successfully.
<code>SS\$_ACCVIO</code>	The <b>return_va_64</b> , <b>return_length_64</b> , or <b>buffer_handle_64</b> argument cannot be written by the caller.
<code>SS\$_BADPARAM</code>	Invalid flags options specified.
<code>SS\$_EXBUFOBJLM</code>	Buffer object cannot be created because it would bring the total number of buffer object pages above the systemwide limit <code>MAXBOBMEM</code> .
<code>SS\$_INSFMEM</code>	Insufficient dynamic memory.

## System Service Descriptions \$CREATE\_BUFOBJ\_64 (Alpha Only)

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 VM\$BUFFER_OBJECT_USER.
SS\$_NOPRIV	Valid flag options were specified but from user mode.
SS\$_PAGNOTWRITE	A page within the address range is not writable.
SS\$_PAGOWNVIO	The pages could not be 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.

## System Service Descriptions

### \$CREATE\_GDZRO (Alpha Only)

---

## \$CREATE\_GDZRO (Alpha Only)

### Create Permanent Global Demand-Zero Section

On Alpha 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.

## System Service Descriptions \$CREATE\_GDZRO (Alpha Only)

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:

World				Group				Owner				System			
D	E	W	R	D	E	W	R	D	E	W	R	D	E	W	R
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

ZK-1706-GE

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.

## System Service Descriptions

### \$CREATE\_GDZRO (Alpha Only)

#### 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 SYSSSTARLET\_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 <b>rad_mask</b> is used as a mask of RADs from which to allocate memory. See the <b>rad_mask</b> argument description for more information.

## System Service Descriptions \$CREATE\_GDZRO (Alpha Only)

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 for future use by Compaq 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.

Note: OpenVMS support for RADs is available only on the new AlphaServer GS series systems. For more information about using RADs, refer to the *OpenVMS Alpha Partitioning and Galaxy Guide*.

## System Service Descriptions

### \$CREATE\_GDZRO (Alpha Only)

#### 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.



## System Service Descriptions \$CREATE\_GDZRO (Alpha Only)

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, refer to the *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 SSS_CREATED_SHPT.
No	No	No shared page tables created. Returns SSS_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 SSS_CREATED.
Yes	Yes	Shared page tables created. Shared page tables can use GH. Returns SSS_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 SYS1-7 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.

## System Service Descriptions

### \$CREATE\_GDZRO (Alpha Only)

- 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.

Refer to the *OpenVMS Alpha Partitioning and Galaxy Guide* for additional information.

#### 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$M_SYSGBL` is set)
- `PRMGBL` privilege to create a permanent global section
- `VM$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

<code>SS\$_NORMAL</code>	A Galaxy-wide section already existed and has been made available.
<code>SS\$_CREATED</code>	Global section has been created.
<code>SS\$_CREATED_SHPT</code>	Global section has been created with shared page tables.
<code>SS\$_ACCVIO</code>	The <b>gs_name_64</b> descriptor cannot be read by the caller, or the <b>reserved_length_64</b> argument was specified as a nonzero value and cannot be written by the caller.

## System Service Descriptions \$CREATE\_GDZRO (Alpha Only)

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 <b>length_64</b> argument is not a multiple of CPU-specific pages.
SS\$_LOCK_TIMEOUT	An OpenVMS Galaxy lock timed out.
SS\$_MRES_PFN SMALL	Preallocated, contiguous, aligned physical memory specified in the Reserved Memory Registry is smaller than the length specified for the global section by the <b>length_64</b> 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 (VM\$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 <b>gs_name_64</b> argument exceeded the allowed depth of 10.

## System Service Descriptions

### \$CREATE\_GFILE (Alpha Only)

---

## \$CREATE\_GFILE (Alpha Only)

### Create Permanent Global Disk File Section

On Alpha 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.

---

## System Service Descriptions \$CREATE\_GFILE (Alpha Only)

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 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).

## System Service Descriptions

### \$CREATE\_GFILE (Alpha Only)

#### 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 SYSSSTARLET\_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 use by Compaq 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

`SCRMPSC`, `SCRMPSC_GFILE_64`, `$DGBLSC`, `$MGBLSC`, `$MGBLSC_64`

## System Service Descriptions

### \$CREATE\_GFILE (Alpha Only)

#### Condition Values Returned

SS\$_CREATED	The service completed successfully. The specified global section did not previously exist and has been created.
SS\$_ACCVIO	The <b>gs_name_64</b> argument or the <b>return_length_64</b> 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 <b>file_offset_64</b> 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 <b>length_64</b> 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.



## System Service Descriptions \$CREATE\_GFILE (Alpha Only)

SS\$NOWRT	The file is read only, and the flag bit SEC\$M_CRF is not set.
SS\$OFF_NOTBLKALGN	The <b>file_offset_64</b> 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 <b>gs_name_64</b> argument exceeded the allowed depth of 10.

## System Service Descriptions

### \$CREATE\_GPFIL (Alpha Only)

---

## \$CREATE\_GPFIL (Alpha Only)

### Create Permanent Global Page File Section

On Alpha systems, creates a permanent global page file section to which processes can map.

This service accepts 64-bit addresses.

#### Format

```
SYS$CREATE_GPFIL 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.

## System Service Descriptions \$CREATE\_GPFIL (Alpha Only)

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:

World				Group				Owner				System			
D	E	W	R	D	E	W	R	D	E	W	R	D	E	W	R
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

ZK-1706-GE

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:

## System Service Descriptions

### \$CREATE\_GPFIL (Alpha Only)

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\_GPFIL 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 for future use by Compaq and should be specified as 0. The condition value SSS\_IVSECF LG 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.

## System Service Descriptions \$CREATE\_GPFIL (Alpha Only)

### 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\_GPFIL\_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 <b>gs_name_64</b> 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 <b>length_64</b> 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 <b>gs_name_64</b> argument exceeded the allowed depth of 10.

## System Service Descriptions

### \$CREATE\_GPFN (Alpha Only)

---

## \$CREATE\_GPFN (Alpha Only)

### Create Permanent Global Page Frame Section

On Alpha 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_gpfm (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.

## System Service Descriptions \$CREATE\_GPFN (Alpha Only)

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 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:

World				Group				Owner				System			
D	E	W	R	D	E	W	R	D	E	W	R	D	E	W	R
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

ZK-1706-GE

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)  
 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)  
 access: read only  
 mechanism: by value

Length of the page frame section in CPU-specific pages.

## System Service Descriptions

### \$CREATE\_GPFN (Alpha Only)

#### 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 SYSSSTARLET\_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_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_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 by Compaq 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 <b>gs_name_64</b> 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).

**System Service Descriptions**  
**\$CREATE\_GPFN (Alpha Only)**

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:[SYSEXEC] with the file name RIGHTSLIST.DAT. If the database already exists, \$CREATE\_RDB fails with the error RMSS\_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

## System Service Descriptions

### \$CREATE\_RDB

#### Condition Values Returned

SS\$ _NORMAL	The service completed successfully.
SS\$ _ACCVIO	The <b>sysid</b> argument cannot be read by the caller.
SS\$ _INSMEM	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 *OpenVMS Record Management Services Reference Manual*.

## \$CREATE\_REGION\_64 (Alpha Only) Create Virtual Region

On Alpha 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 the flag VASM\_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
VASC_REGION_UCREATE_UOWN	User create mode and user owner mode
VASC_REGION_UCREATE_SOWN	User create mode and supervisor owner mode

## System Service Descriptions

### \$CREATE\_REGION\_64 (Alpha Only)

Symbol	Create and Owner Modes
VASC_REGION_UCREATE_EOWN	User create mode and executive owner mode
VASC_REGION_UCREATE_KOWN	User create mode and kernel owner mode
VASC_REGION_SCREATE_SOWN	Supervisor create mode and supervisor owner mode
VASC_REGION_SCREATE_EOWN	Supervisor create mode and executive owner mode
VASC_REGION_SCREATE_KOWN	Supervisor create mode and kernel owner mode
VASC_REGION_ECREATE_EOWN	Executive create mode and executive owner mode
VASC_REGION_ECREATE_KOWN	Executive create mode and kernel owner mode
VASC_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 flags that are valid for the \$CREATE\_REGION\_64 service:

## System Service Descriptions \$CREATE\_REGION\_64 (Alpha Only)

Flag	Description
VASM_DESCEND	Created region is a descending region; that is, allocation occurs toward decreasing virtual addresses. If VASM_DESCEND is not specified, the region allocation occurs toward increasing virtual addresses.
VASM_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.
VASM_P0_SPACE	Create region in P0 space. This flag cannot be set if VASM_P1_SPACE is set.
VASM_P1_SPACE	Create region in P1 space. This flag cannot be set if VASM_P0_SPACE is set.

All other bits in the **flags** argument are reserved for future use by Compaq. The condition value SSS\_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 VASM\_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

## System Service Descriptions

### \$CREATE\_REGION\_64 (Alpha Only)

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 `VASM_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 `VASM_P0_SPACE` and `VASM_P1_SPACE` are clear)
- P0 space (if the flag `VASM_P0_SPACE` is set and `VASM_P1_SPACE` is clear)
- P1 space (if the flag `VASM_P1_SPACE` is set and `VASM_P0_SPACE` is clear)

If the flag `VASM_SHARED_PTS` is set and this argument is specified, the specified starting address must be aligned to a natural page table page boundary or an error is returned.

## 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 `VASM_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 SYS1-3) are not allowed to create address space into a shared page table region because they have an implicit dependency on process-private page tables.



## System Service Descriptions \$CREATE\_REGION\_64 (Alpha Only)

**Table SYS1–3 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 service 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] <sup>1</sup>	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.

---

<sup>1</sup>\$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 VASM\_SHARED\_PTS flag is set in the **flags** argument, the virtual region is created on a CPU-specific page table page aligned boundary.

Compaq recommends that you not 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 SSS\_VA\_NOTPAGALGN.

## System Service Descriptions

### `$CREATE_REGION_64` (Alpha Only)

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 `$GETSYI` system service and specifying the `SYIS_PAGE_SIZE` item code.
2. Determine the number of CPU-specific pages mapped by a single page table page by using the `$GETSYI` system service and specifying the `SYIS_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. The Alpha architecture provides three granularity hint factors: 8, 64, and 512 pages.

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 SYS1–4 highlights the values `$CREATE_REGION_64` returns for various region lengths. When the `length_64` argument is not an 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 sections 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.

## System Service Descriptions \$CREATE\_REGION\_64 (Alpha Only)

**Table SYS1–4 Sample Returned Values from \$CREATE\_REGION\_64**

<b>length_64</b>	<b>return_va_64</b>	<b>return_length_64</b>	<b>Comments</b>
1,048,576 (1 MB)	FFFFFFFFB00800000 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)	FFFFFFFFBFC000000 at least 26 zero bits	67,108,864 (64 MB)	Returned VA accommodates GH factor of 8 for shared page table pages.
73,400,320 (70 MB)	FFFFFFFFBF8000000 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)	FFFFFFFFBFC000000 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 <b>return_region_id_64</b> argument, the <b>return_va_64</b> argument, or the <b>return_length_64</b> argument cannot be written by the caller.
SS\$_IVREGFLG	One or more of the reserved bits in the <b>flags</b> argument is set, or an illegal combination of <b>flags</b> bits are set.
SS\$_LEN_NOTPAGMULT	The <b>length_64</b> 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.

## System Service Descriptions

### \$CREATE\_REGION\_64 (Alpha Only)

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 VASM\_SHARED\_PTS is set, the **start\_va\_64** argument is not CPU-specific page table page aligned.

## \$CREATE\_USER\_PROFILE

### Create User Profile

Returns an encoded security profile for the specified user.

#### Format

SYS\$CREATE\_USER\_PROFILE usnam ,[itmlst] ,[flags] ,usrpro ,usrprolen ,[contxt]

#### C Prototype

```
int sys$create_user_profile (void *usnam, void *itmlst, unsigned int flags, void
                             *usrpro, unsigned int *usrprolen, unsigned int *contxt);
```

#### Arguments

##### usnam

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 **usnam** 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.

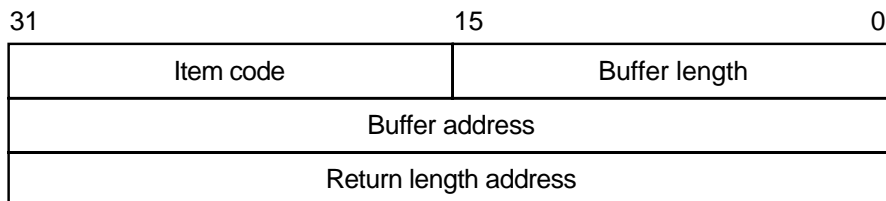
For more information about user names, refer to the *OpenVMS Guide to System Security*.

##### 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 Item Codes section for a list of valid item codes for \$CREATE\_USER\_PROFILE.



ZK-5186A-GE

## System Service Descriptions

### \$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 to control the behavior of the \$CREATE\_USER\_PROFILE service. This argument does not accept ISS\$ mask values; therefore, continue to use the CHP\$ mask values for this argument.

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 set, the service initializes the security profile from the user's default classification instead. This flag is reserved to Compaq.
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 reference

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 SSS\_INSFMEM.

## System Service Descriptions

### **\$CREATE\_USER\_PROFILE**

#### **CHP\$\_CLASS**

The classification to be associated with the created security profile. This item code is reserved to Compaq.

#### **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 SSS\_INSFMEM.

#### **ISS\$\_AUTHPRIV**

Quadword containing the authorized privileges. See \$PRVDEF macro for definitions.

#### **ISS\$\_FLAGS**

Longword containing user flags. The following flag is supported:

ISSM\_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 Compaq. 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 Compaq. See the \$CLSDEF macro for definitions.

#### **ISS\$\_MODE**

Longword containing the access mode. See \$PSLDEF macro for definitions.

#### **ISS\$\_PERMPRIV**

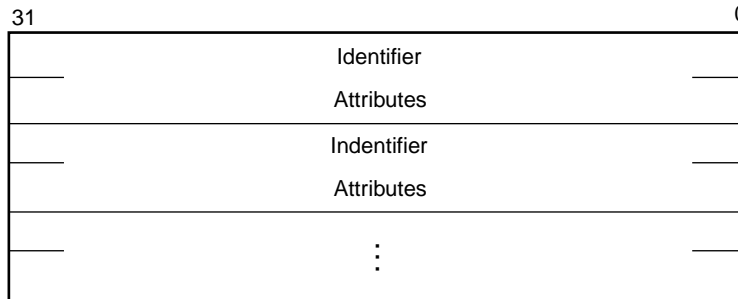
Quadword containing the permanent privileges. See \$PRVDEF macro for definitions.



**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:



VM-0459A-AI

**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 Compaq.

**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.

## System Service Descriptions

### \$CREATE\_USER\_PROFILE

\$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 RIGHTSLLIST.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\$_BADITMCD	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 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 (Alpha)

mechanism: by 32-bit reference (VAX)

Attributes to be associated with the logical name. The **attr** argument is the 32-bit address (on VAX systems) or the 32- or 64-bit address (on Alpha systems) 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
LNMSM_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.
LNMSM_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.

## System Service Descriptions

### \$CRELNM

#### **tabnam**

OpenVMS usage: logical\_name  
type: character-coded text string  
access: read only  
mechanism: by 32- or 64-bit descriptor-fixed-length string descriptor (Alpha)  
mechanism: by 32-bit descriptor-fixed-length string descriptor (VAX)

Name of the table in which to create the logical name. The **tabnam** argument is the 32-bit address (on VAX systems) or the 32- or 64-bit address (on Alpha systems) 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 (Alpha)  
mechanism: by 32-bit descriptor-fixed-length string descriptor (VAX)

Name of the logical name to be created. The **lognam** argument is the 32-bit address (on VAX systems) or the 32- or 64-bit address (on Alpha systems) 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 (\$), 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 (Alpha)  
mechanism: by 32-bit reference (VAX)

Access mode to be associated with the logical name. The **acmode** argument is the 32-bit address (on VAX systems) or the 32- or 64-bit address (on Alpha systems) 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.

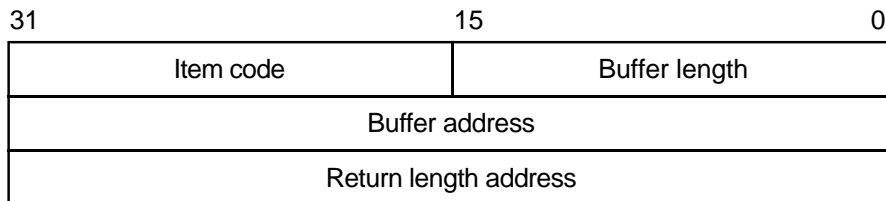
**itmlst**

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 (Alpha)  
 mechanism: by 32-bit reference (VAX)

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-bit address (on VAX systems) or the 32- or 64-bit address (on Alpha systems) 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.

Note that it is possible to create a logical that has no equivalence names. This is done by either omitting the **itmlst** argument to `$CRELNM`, or by not including the `LNMS_STRING` item code to the **itmlst** data structure that is passed into `$CRELNM`. It is not possible to create this kind of logical using `DCL`.

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:

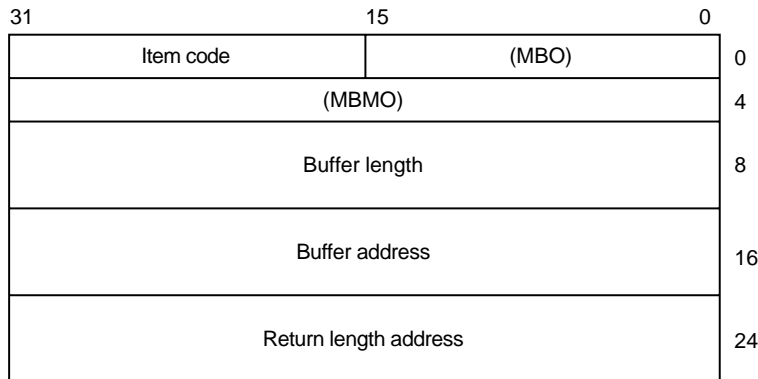
Descriptor Field	Definition
Buffer length	A word specifying the number of bytes in the buffer pointed to by the buffer address field. The length of the buffer needed depends on the item code specified in the item code field of the item descriptor. If the value of buffer length is too small, the service 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.

## System Service Descriptions

### \$CRELNM

Descriptor Field	Definition
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 LNMS_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:



ZK-8782A-GE

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 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.

Descriptor Field	Definition
Buffer address	A quadword containing the 64-bit address of the buffer that receives or passes information.
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 LNMS_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

### LNMS\_ATTRIBUTES

When you specify LNMS\_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 LNMS\_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
LNMSM_CONCEALED	If set, OpenVMS RMS interprets the equivalence name as a device name or logical name with the LNMSM_CONCEALED attribute.
LNMSM_TERMINAL	If set, further iterative logical name translation on the equivalence name is not to be performed.

### LNMS\_CHAIN

When you specify LNMS\_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 LNMS\_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.

### LNMS\_STRING

When you specify LNMS\_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 LNMS\_STRING, it creates an equivalence name entry for the logical name using the most recently specified values for LNMS\_ATTRIBUTES. The equivalence name entry includes the following information:

- Name specified by LNMS\_STRING.

## System Service Descriptions

### \$CRELNM

- 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 LNMS\_ATTRIBUTES (if these are present in the item list).

Therefore, you should construct the item list so that the LNMS\_ATTRIBUTES item codes immediately precede the LNMS\_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 LNMS\_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.

#### LNMS\_TABLE

When you specify LNMS\_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\$INSMEM	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 <b>tabnam</b> argument, the <b>lognam</b> argument, or the equivalence string specifies a string whose length is not in the required range of 1 through 255 characters. The <b>lognam</b> argument specifies a string whose length is not in the required range of 1 to 31 characters for directory table entries.

## System Service Descriptions

### \$CRELNM

SS\$\_IVLOGTAB

The **tabnam** argument does not specify a logical name table.

SS\$\_NOLOGTAB

Either the specified logical name table does not exist or the logical name translation of the table name exceeded the allowable depth of 10 translations.

SS\$\_NOPRIV

The caller lacks the necessary privilege to create the logical name.

## \$CRELNT

### Create Logical Name Table

Creates a process-private or shareable logical name table.

On Alpha 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 (Alpha)  
 mechanism: by 32-bit reference (VAX)

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-bit address (on VAX systems) or the 32- or 64-bit address (on Alpha systems) 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
LNMSM_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 LIBSSPAWN routine. You can specify this attribute only for process-private logical name tables; it is ignored for shareable tables.

## System Service Descriptions

### \$CRELNT

Attribute	Description
	<p>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 <b>attr</b> 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 <b>attr</b> argument. The process-private directory table LNM\$PROCESS_DIRECTORY does not have the LNM\$M_CONFINE attribute.</p>
LNMSM_CREATE_IF	<p>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.</p> <p>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.</p> <p>Regardless of the setting of LNM\$M_CREATE_IF:</p> <ul style="list-style-type: none"> <li>• 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.</li> <li>• 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.</li> </ul>
LNMSM_NO_ALIAS	<p>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.</p>

#### resnam

OpenVMS usage: logical\_name  
 type: character-coded text string  
 access: write only  
 mechanism: by 32- or 64-bit descriptor-fixed-length string descriptor (Alpha)

mechanism: by 32-bit descriptor–fixed-length string descriptor (VAX)

Name of the newly created logical name table, returned by \$CRELNT. The **resnam** argument is the 32-bit address (on VAX systems) or the 32- or 64-bit address (on Alpha systems) 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 (Alpha)  
 mechanism: by 32-bit reference (VAX)

Length in bytes of the name of the newly created logical name table, returned by \$CRELNT. The **reslen** argument is the 32-bit address (on VAX systems) or the 32- or 64-bit address (on Alpha systems) 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 (Alpha)  
 mechanism: by 32-bit reference (VAX)

Maximum number of bytes of memory to be allocated for logical names contained in this logical name table. The **quota** argument is the 32-bit address (on VAX systems) or the 32- or 64-bit address (on Alpha systems) 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 (Alpha)  
 mechanism: by 32-bit reference (VAX)

Protection mask to be associated with the newly created shareable logical name table. The **promsk** argument is the 32-bit address (on VAX systems) or the 32- or 64-bit address (on Alpha systems) 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:

World				Group				Owner				System			
D	C	W	R	D	C	W	R	D	C	W	R	D	C	W	R
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

ZK-5893A-GE

Create access is required to create a shareable table within another shareable table.

## System Service Descriptions

### \$CRELNT

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 (Alpha)  
mechanism: by 32-bit descriptor-fixed-length string descriptor(VAX)

The name of the new logical name table. The **tabnam** argument is the 32-bit address (on VAX systems) or the 32- or 64-bit address (on Alpha systems) 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 (Alpha)  
mechanism: by 32-bit descriptor-fixed-length string descriptor (VAX)

Name string for the parent table name. The **partab** argument is the 32-bit address (on VAX systems) or the 32- or 64-bit address (on Alpha systems) 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 table 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)  
mechanism: by 32-bit reference (VAX)

Access mode to be associated with the newly created logical name table. The **acmode** argument is the 32-bit address (on VAX systems) or the 32- or 64-bit address (on Alpha systems) 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.

## System Service Descriptions

### \$CRELNT

#### Related Services

\$CRELNM, \$DELLNM, \$STRNLNM

#### 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 LNMSM_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 <b>partab</b> argument specifies a string whose length is not within the required range of 1 to 31 characters.
SS\$IVLOGTAB	The <b>tabnam</b> 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: longword (unsigned)  
 access: read only  
 mechanism: by value

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

##### chan

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.

# System Service Descriptions

## \$CREMBX

### 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 word value containing this number. If you do not specify the **bufquo** argument or specify it as 0, the operating system provides a default value.

The maximum value that you can specify with the **bufquo** argument is 60000. For a temporary mailbox, this value must be less than or equal to the process buffer quota.

### 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:

World				Group				Owner				System			
L	P	W	R	L	P	W	R	L	P	W	R	L	P	W	R
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

ZK-1707-GE

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.

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 *OpenVMS Programming Concepts Manual*.

**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

## System Service Descriptions

### \$CREMBX

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, refer to the *OpenVMS I/O User's Reference Manual*.

#### nullarg

OpenVMS usage: null\_arg  
type: longword (unsigned)  
access: read only  
mechanism: by value

Placeholder argument reserved by Compaq.

## 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.

## System Service Descriptions

### \$CREMBX

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 LNMSJOB, 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=LNMPROCESS_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, refer to the *OpenVMS Programming Concepts Manual*.

## System Service Descriptions

### \$CREMBX

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 **prmf** argument is specified as 0; however, a process that has PRMMBX privilege will also meet this requirement.
- PRMMBX privilege whenever the **prmf** 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, \$SENDERR, \$SENDJBC, \$SENDJBCW, \$SENDOPR

#### 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.

## System Service Descriptions \$CREMBX

SS\$_BADPARAM	The <b>bufquo</b> argument specified a value greater than approximately 65324, which is 65535 minus the size of a mailbox unit control block (UCB).
SS\$_EXBYTLM	The process has insufficient buffer I/O byte count (BYTLM) quota to allocate the mailbox UCB or to satisfy buffer requirements.
SS\$_INSFMEM	The system dynamic memory is insufficient for completing the service.
SS\$_INTERLOCK	The bit map lock for allocating mailboxes from the specified shared memory is locked by another process.
SS\$_IVLOGNAM	The logical name string has a length of 0 or has more than 255 characters.
SS\$_IVSTSFLG	The bit set in the <b>prmlflg</b> 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 Compaq support representative.
SS\$_SHMNOTCNCT	The shared memory named in the <b>name</b> 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 <b>lognam</b> 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] ,[privadr] ,[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 *privadr, 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 SYSSYSTEM: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 SYSS\$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 SYSS\$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 SYSS\$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 SYSS\$SYSTEM:LOGINOUT.EXE; in this case, SYSS\$ERROR has the same equivalence name as SYSS\$OUTPUT.

**privadr**

OpenVMS usage: mask\_privileges  
 type: quadword (unsigned)  
 access: read only  
 mechanism: by reference

Privileges to be given to the created process. The **privadr** argument is the address of a quadword bit mask wherein each bit corresponds to a privilege; setting a bit gives the privilege. If the **privadr** 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 SYS1–5 gives the symbolic name and description of each privilege.

**Table SYS1–5 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

(continued on next page)

## System Service Descriptions

### \$CREPRC

Table SYS1–5 (Cont.) User Privileges

Privilege	Symbolic Name	Description
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 <sup>1</sup>	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
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 nonshared 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

<sup>1</sup>This privilege replaces the DETACH privilege; however, the prior mask, PRV\$M\_DETACH, is still valid for existing programs.

(continued on next page)

**Table SYS1–5 (Cont.) User Privileges**

Privilege	Symbolic Name	Description
SYSPRV	PRV\$M_SYSPRV	Access files and other resources as if you have a system UIC
TMPMBX	PRV\$M_TMPMBX	Create temporary mailboxes
UPGRADE	PRV\$M_UPGRADE	Can upgrade classification
VOLPRO	PRV\$M_VOLPRO	Override volume protection
WORLD	PRV\$M_WORLD	World process control

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}};
```

## System Service Descriptions

### \$CREPRC

**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 meaning:

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 system 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\_Mxxxxx, where xxxxx are the characters of the quota name that follow "PQL\$\_" in the quota name.

Default values are named PQL\_Dxxxxx, 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 SSS\_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

## System Service Descriptions

### \$CREPRC

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 pages (on VAX systems) or pagelets (adjusted up or down to represent CPU-specific pages on Alpha systems) that can be used to provide secondary storage in the paging file for the execution of a process.

Minimum: PQL\_MPGFLQUOTA  
Default: PQL\_DPGFLQUOTA  
Pooled

#### **PQL\$\_PRCLM**

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

Minimum: PQL\_MPRCLM  
Default: PQL\_DPRCLM  
Pooled

#### **PQL\$\_TQELM**

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

Minimum: PQL\_MTQELM  
Default: PQL\_DTQELM  
Pooled

#### **PQL\$\_WSDEFAULT**

Default working set size. This quota defines the number of pages (on VAX systems) or pagelets (adjusted up or down to represent CPU-specific pages on Alpha systems) 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 (SLCKPAG) 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 SSS\_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.

## System Service Descriptions

### \$CREPRC

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 VAX range is 0 to 31, where 31 is the highest priority and 0 is the lowest. Usual priorities are in the range 0 to 15, and real-time priorities are in the range 16 to 31.

The OpenVMS Alpha 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 node.

If the **baspri** argument is not specified, the priority defaults to 2 for VAX MACRO and VAX BLISS-32 and 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 SYSSSYSTEM: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 SSS\_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.

## System Service Descriptions

### \$CREPRC

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 22 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.

## System Service Descriptions \$CREPRC

Symbolic Name	Description
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).  Note: OpenVMS support for RADs is available only on the new AlphaServer GS series systems. For more information about using RADs, refer to the <i>OpenVMS Alpha Partitioning and Galaxy Guide</i> .
PRC\$M_IMGDMPI	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 <b>image</b> argument specifies SYSS\$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_NETWORK	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 <i>Username:</i> and <i>Password:</i> prompts if the process is interactive and detached and the image is SYSS\$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, SYSS\$SYSTEM:LOGINOUT.EXE prompts you for the user name and password to be associated with the process. The prompts are displayed at the SYSS\$INPUT device.

## System Service Descriptions

### \$CREPRC

Symbolic Name	Description
PRC\$M_NOUAF	Do not check authorization file if the process is detached and the image is SYSS\$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.
PRC\$M_PARSE_EXPANDED	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 PRC\$M\_BATCH, PRC\$M\_INTER, PRC\$M\_NOUAF, PRC\$M\_NETWORK, and PRC\$M\_NOPASSWORD are intended for use by Compaq software.

#### itmlst

OpenVMS usage: reserved  
 type: longword (unsigned)

The **itmlst** argument is reserved by Compaq.

#### 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 **stsf** 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.

Note: OpenVMS support for RADs is available only on the new AlphaServer GS series systems. For more information about using RADs, refer to the *OpenVMS Alpha Partitioning and Galaxy Guide*.

## 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 `SYSS$INPUT`, `SYSS$OUTPUT`, or `SYSS$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 `SYSS$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 `SYSS$INPUT`, processes it, and reads another command. The presence of the CLI enables the process to execute multiple images. It also enables an image running in the process to use Run-Time Library procedures, such as `LIB$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 **stsf** argument.

## System Service Descriptions

### \$CREPRC

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**

SCANEXH, \$DCLEXH, \$DELPRC, \$EXIT, \$FORCEX, \$GETJPI, \$GETJPIW, \$SHIBER, \$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.

## System Service Descriptions

### \$CREPRC

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: <ul style="list-style-type: none"><li>• The process has exceeded its quota for the creation of subprocesses.</li><li>• A quota value specified for the creation of a subprocess exceeds the creating process's corresponding quota.</li><li>• The quota is deductible and the remaining quota for the creating process would be less than the minimum.</li></ul>
SS\$_INCOMPAT	The remote node is running an incompatible version of the operating system, namely, one that does not support remote process creation.
SS\$_INSMEM	The system dynamic memory is insufficient for the requested operation.
SS\$_INVARG	An invalid argument was specified.
SS\$_IVLOGNAM	At least one of the following two conditions is true: <ul style="list-style-type: none"><li>• The specified process name has a length of 0 or has more than 15 characters.</li><li>• The specified image name, input name, output name, or error name has more than 255 characters.</li></ul>
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 pages (on VAX systems) or pagelets (on Alpha systems) 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 systems, the **retadr** argument should be checked by programs for actual allocation. Because the Alpha architecture defines more than one page size, more space might be created than was specified in the **retadr** 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
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 architecture has a larger page size than the VAX architecture, more space is potentially affected on Alpha 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 Compaq (including Run-Time Library procedures). This system service provides no means to communicate a change in virtual address space with other routines. Compaq 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 *OpenVMS*

## System Service Descriptions

### \$CRETVA

*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 <b>inadr</b> argument cannot be read by the caller, or the <b>retadr</b> 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 Only) Create Virtual Address Space

On Alpha 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

```
SYS$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
VASC_P0	Program region
VASC_P1	Control region
VASC_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 VAS\_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.

## System Service Descriptions

### \$CRETVA\_64 (Alpha Only)

#### 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 SSS\_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
VASM_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 use by Compaq and should be specified as 0. The condition value SSS\_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 SSS\_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.

## System Service Descriptions

### \$CRETVA\_64 (Alpha Only)

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

<code>SS\$_NORMAL</code>	The service completed successfully.
<code>SS\$_ACCVIO</code>	The <b>return_va_64</b> or <b>return_length_64</b> argument cannot be written by the caller.
<code>SS\$_EXPGFLQUOTA</code>	The process has exceeded its paging file quota.
<code>SS\$_INSFWSL</code>	The process's working set limit is not large enough to accommodate the increased virtual address space.
<code>SS\$_IVACMODE</code>	The caller's mode is less privileged than the create mode associated with the region.
<code>SS\$_IVREGID</code>	An invalid region ID was specified.
<code>SS\$_IVVAFLG</code>	An invalid flag, a reserved flag, or an invalid combination of flags and arguments was specified.
<code>SS\$_LEN_NOTPAGMULT</code>	The <b>length_64</b> argument is not a multiple of CPU-specific pages.
<code>SS\$_NOSHPTS</code>	The region ID of a shared page table region was specified.
<code>SS\$_PAGNOTINREG</code>	A page in the specified range is not within the specified region.
<code>SS\$_PAGOWNVIO</code>	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.
<code>SS\$_REGISFULL</code>	The specified virtual region is full.



## System Service Descriptions \$CRETVA\_64 (Alpha Only)

SS\$ VA_IN_USE	A page in the specified range is already mapped, and the VASM_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 <b>start_va_64</b> 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, 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 or a VAX system. The two system types are discussed separately in this section.

##### **Alpha System Usage**

On Alpha systems, if you do *not* set the **SEC\$M\_EXPREG** flag, the **inadr** argument specifies the starting and ending virtual addresses of the region to be mapped. Addresses in system space are not allowed. The addresses must be aligned on CPU-specific pages; no rounding to CPU-specific pages occurs. The lower address of the **inadr** argument must be on a CPU-specific page boundary and the higher address of the **inadr** argument must be 1 less than a CPU-specific boundary, thus forming a range, from lowest to highest, of address bytes. You can use the **SYIS\_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 SECSM\_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 SECSM\_EXPREG flag is set, the second **inadr** longword is ignored, while bit 30 (the second most significant bit) of the first **inadr** longword is used to determine the region of choice. If the bit is clear, P0 is chosen; if the bit is set, P1 is chosen. On Alpha systems, bit 31 (the most significant bit) of the first **inadr** longword *must be* 0. To ensure compatibility between VAX and Alpha systems when you choose a region, Compaq 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 SECSM\_EXPREG flag; however, to ensure compatibility between VAX and Alpha systems, Compaq recommends that the SECSM\_EXPREG flag be clear when the **inadr** argument is omitted.

#### VAX System Usage

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

---

**Note**

---

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

---

If, on the other hand, you *do* set the SECSM\_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.

---

## System Service Descriptions

### \$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 VAX systems, bit 31 (the most significant bit) of the first **inadr** longword *is ignored*. To ensure compatibility between VAX and Alpha systems when you choose a region, Compaq 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. You must also ensure that `SEC$M_EXPREG` is *not* set in the **flags** argument. Omitting the **inadr** argument with `SEC$M_EXPREG` set is interpreted by VAX systems as a request to map with no region preference. This latter combination of argument settings is strongly discouraged, as the chosen region is indeterminate. To ensure compatibility between VAX and Alpha systems, Compaq 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 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_GBL	Pages form a global section. The default is private section.
SEC\$M_CRF	Pages are copy-on-reference. By default, pages are shared.
SEC\$M_DZRO	Pages are demand-zero pages. By default, they are not zeroed when copied. For page file sections, the default is demand zero.
SEC\$M_EXPREG	Pages are mapped into the first available space. By default, pages are mapped into the range specified by the <b>inadr</b> argument. See the <b>inadr</b> argument description for a complete explanation of how to set the SEC\$M_EXPREG flag.
SEC\$M_WRT	Pages form a read/write section. By default, pages form a read-only section. For page file sections, the default is writeable.
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. ‡On Alpha systems, when the SEC\$M_PFNMAP flag is set, the <b>pagcnt</b> and <b>relpag</b> arguments are interpreted in CPU-specific pages, <i>not</i> as pagelets.
SEC\$M_SYSGBL	Pages form a system global section. By default, pages form a group global section.
SEC\$M_PAGFIL	Pages form a global page file section. By default, pages form a disk file section. SEC\$M_PAGFIL also implies SEC\$M_WRT and SEC\$M_DZRO.

---

‡Alpha specific

## System Service Descriptions

### \$CRMPSC

Flag	Description
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_NO_OVERMAP	Pages cannot overmap existing address space. Note that, by default, pages can overmap existing address space.

#### 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 *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 SECSK_MATALL	Match all versions of the section.
1 SECSK_MATEQU	Match only if major and minor identifications match.
2 SECSK_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 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 and VAX systems, 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

## System Service Descriptions

### \$CRMPSC

Number of pages (on VAX systems) or pagelets (on Alpha systems) in the section. The **pagcnt** argument is a longword containing this number.

On Alpha systems, the smallest allocation is an Alpha 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 pages, 16384 bytes.

On Alpha systems, if the SEC\$M\_PFNMAP flag bit is set, the **pagcnt** argument is interpreted as CPU-specific pages, *not* as pagelets.

On Alpha and VAX 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 SYS1-6 shows which arguments are required and which are optional for three different uses of the \$CRMPSC service.

**Table SYS1-6 Required and Optional Arguments for the \$CRMPSC Service**

Argument	Create/Map Global Section	Map Global <sup>1</sup> Section	Create/Map Private Section
<b>inadr</b>	Optional <sup>2</sup>	Required	Required
<b>retadr</b>	Optional	Optional	Optional
<b>acmode</b>	Optional	Optional	Optional
<b>flags</b>			
SEC\$M_GBL	Required	Ignored	Not used
SEC\$M_CRF <sup>3</sup>	Optional	Not used	Optional

<sup>1</sup>The Map Global Section (\$MGBLSC) service maps an existing global section.

<sup>2</sup>See the description of **inadr** for the rules governing the omission of the argument.

<sup>3</sup>For physical page frame sections: **vbn** specifies the starting page frame number; **chan** must be 0; **pf** 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 **pf** not used.

(continued on next page)



**Table SYS1–6 (Cont.) Required and Optional Arguments for the \$CRMPSC Service**

Argument	Create/Map Global Section	Map Global <sup>1</sup> Section	Create/Map Private Section
SEC\$M_DZRO <sup>3</sup>	Optional	Not used	Optional
SEC\$M_EXPREG	Optional	Optional	Optional
SEC\$M_PERM	Optional <sup>2</sup>	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
<b>gsdnam</b>	Required	Required	Not used
<b>ident</b>	Optional	Optional	Not used
<b>relpag</b> <sup>3</sup>	Optional	Optional	Not used
<b>chan</b> <sup>3</sup>	Required		Required
<b>pagcnt</b>	Required		Required
<b>vbn</b> <sup>3</sup>	Optional		Optional
<b>prot</b>	Optional		Not used
<b>pfc</b> <sup>3</sup>	Optional		Optional

<sup>1</sup>The Map Global Section (\$MGBLSC) service maps an existing global section.

<sup>2</sup>See the description of **inadr** for the rules governing the omission of the argument.

<sup>3</sup>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.

## System Service Descriptions

### \$CRMPSC

The following diagram depicts the mask:

World				Group				Owner				System			
D	E	W	R	D	E	W	R	D	E	W	R	D	E	W	R
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

ZK-1706-GE

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 pages (on VAX systems) or pagelets (on Alpha systems) are to be brought into memory when a page fault occurs for a single page.

On Alpha 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 system with an 8KB page size) will be mapped for each physical page. The system cannot map less than one physical page.

On VAX systems, this argument is not used for page file sections or physical page frame sections.

## 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 pages (on VAX systems) or pagelets (on Alpha systems) 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. Refer to the memory management section in the *OpenVMS Programming Concepts Manual*.

Depending on the actual operation requested, certain arguments are required or optional. Table SYS1-6 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
pfcb	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
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 and VAX 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.

## System Service Descriptions

### \$CRMPSC

#### Required Access or Privileges

If \$CRMPSC specifies a global section and the SSS\_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
- SHMEM privilege to create a global section in memory shared by multiple processors (VAX only)

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 <b>inadr</b> argument, <b>gsdnam</b> argument, or name descriptor cannot be read by the caller; the <b>inadr</b> argument was omitted; or the <b>retadr</b> 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 <b>relpag</b> 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.

## System Service Descriptions

### \$CRMPSC

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).

## System Service Descriptions

### \$CRMPSC

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 <b>gsdnam</b> argument exceeded the allowed depth.
SS\$_VA_IN_USE	A page in the specified input address range is already mapped, and the flag <b>SEC\$M_NO_OVERMAP</b> 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 Only) Create and Map Private Disk File Section

On Alpha 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
VASC_P0	Program region
VASC_P1	Control region
VASC_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.

## System Service Descriptions

### \$CRMPSC\_FILE\_64 (Alpha Only)

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



## System Service Descriptions \$CRMPSC\_FILE\_64 (Alpha Only)

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.
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 for future use by Compaq and should be specified as 0. The condition value SSS\_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.

## System Service Descriptions

### \$CRMPSC\_FILE\_64 (Alpha Only)

#### 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 nonzero, the condition value SSS\_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 SSS\_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 SSS\_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.

## System Service Descriptions \$CRMPSC\_FILE\_64 (Alpha Only)

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, \$SULKPAG\_64, \$SULWSET\_64, \$SUPDSEC\_64, \$SUPDSEC\_64W

### Condition Values Returned

SS\$_NORMAL	The service completed successfully.
SS\$_ACCVIO	The <b>return_va_64</b> argument or the <b>return_length_64</b> argument cannot be written by the caller.
SS\$_CHANVIO	The specified channel was assigned from a more privileged access mode.
SS\$_ENDOFFILE	The <b>file_offset_64</b> 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 <b>length_64</b> 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 <b>file_offset_64</b> argument is not a multiple of virtual disk blocks.

## System Service Descriptions

### \$CRMPSC\_FILE\_64 (Alpha Only)

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 <b>start_va_64</b> argument is not CPU-specific page aligned.

---

## \$CRMPSC\_GDZRO\_64 (Alpha Only)

### Create and Map to Global Demand-Zero Section

On Alpha 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:

## System Service Descriptions

### \$CRMPSC\_GDZRO\_64 (Alpha Only)

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 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:

World				Group				Owner				System			
D	E	W	R	D	E	W	R	D	E	W	R	D	E	W	R
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

ZK-1706-GE

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
VASC_P0	Program region
VASC_P1	Control region
VASC_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 number of bytes that can be mapped by a CPU-specific 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. 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).

## System Service Descriptions

### \$CRMPSC\_GDZRO\_64 (Alpha Only)

The \$PSLDEF macro in STARLET.MLB and the file PSLDEF.H in SYSSSTARLET\_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.
SEC\$M_NO_OVERMAP	Pages cannot overmap existing address space.
SEC\$M_PERM	Global section is permanent.



## System Service Descriptions \$CRMPSC\_GDZRO\_64 (Alpha Only)

Flag	Description
SEC\$M_RAD_HINT	When set, the argument <b>rad_mask</b> is used as a mask of RADs from which to allocate memory. See the <b>rad_mask</b> 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 for future use by Compaq 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.

## System Service Descriptions

### \$CRMPSC\_GDZRO\_64 (Alpha Only)

If SECSM\_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 SSS\_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 (8, 64, or 512 pages), then granularity hints are used to map to the global section.

#### **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 SECSM\_RAD\_HINT flag is specified. Otherwise, this argument is ignored.

## System Service Descriptions \$CRMPSC\_GDZRO\_64 (Alpha Only)

On a system that does not support resource affinity domains (RADs), specifying 1 for the **rad\_mask** argument is allowed.

Note: OpenVMS support for RADs is available only on the new AlphaServer GS series systems. For more information about using RADs, refer to the *OpenVMS Alpha Partitioning and Galaxy Guide*.

### 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 **VM\$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,

## System Service Descriptions

### \$CRMPSC\_GDZRO\_64 (Alpha Only)

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 *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 <code>region_id_64</code>	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.

## System Service Descriptions \$CRMPSC\_GDZRO\_64 (Alpha Only)

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:

## System Service Descriptions

### \$CRMPSC\_GDZRO\_64 (Alpha Only)

Table SYS1-7 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. Private page tables will always be used, even if you are specifying a shared page table region into which to map the section.	Set the SEC\$M_WRT flag in the map request. 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 SSS_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 SSS_IVSECFLG is returned.

#### Notes

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 SSS\_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 SSS\_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 SSS\_INSFLPGS or SSS\_MRES\_PFN SMALL, a value is not returned in the **reserved\_length\_64** argument.

## System Service Descriptions \$CRMPSC\_GDZRO\_64 (Alpha Only)

If the service returns a successful condition value or if SSS\_INSFLPGS or SSS\_MRES\_PFN SMALL 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
- VM\$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 <b>gs_name_64</b> argument cannot be read by the caller, or the <b>return_va_64</b> or <b>return_length_64</b> 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.

## System Service Descriptions

### \$CRMPSC\_GDZRO\_64 (Alpha Only)

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\$_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 <b>length_64</b> argument is not a multiple of CPU-specified pages. Or, if a shared page table region is specified by the <b>region_id_64</b> argument, the <b>map_length_64</b> 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 <b>length_64</b> 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 VM\$MEM_RESIDENT_USER.



## System Service Descriptions \$CRMPSC\_GDZRO\_64 (Alpha Only)

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 <b>section_offset_64</b> argument is not CPU-specific page aligned. Or, if a shared page table region is specified by the <b>region_id_64</b> argument, the <b>section_offset_64</b> argument is not CPU-specific page table page aligned.
SS\$ OFFSET_TOO_BIG	The <b>section_offset_64</b> 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 <b>gs_name_64</b> 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 <b>SECSM_NO_OVERMAP</b> 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 <b>start_va_64</b> argument is not CPU-specific page aligned. Or, if a shared page table region is specified by the <b>region_id_64</b> argument, the <b>start_va_64</b> argument is not CPU-specific page table page aligned.

---

## \$CRMPSC\_GFILE\_64 (Alpha Only) Create and Map Global Disk File Section

On Alpha 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	SECSK_MATALL	Match all versions of the section.

---

## System Service Descriptions \$CRMPSC\_GFILE\_64 (Alpha Only)

Value	Symbolic Name	Match Criteria
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.

## System Service Descriptions

### \$CRMPSC\_GFILE\_64 (Alpha Only)

#### 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 SYSSSTARLET\_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
VASC_P0	Program region
VASC_P1	Control region
VASC_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 SYSSSTARLET\_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 S\$\$\_IVACMODE is returned if the caller is less privileged than the create mode for the region.

## System Service Descriptions \$CRMPSC\_GFILE\_64 (Alpha Only)

### 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 for future use by Compaq 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.

## System Service Descriptions

### \$CRMPSC\_GFILE\_64 (Alpha Only)

#### **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 nonzero. 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 SSS\_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

## System Service Descriptions \$CRMPSC\_GFILE\_64 (Alpha Only)

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 SSS\_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 SSS\_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

## System Service Descriptions

### \$CRMPSC\_GFILE\_64 (Alpha Only)

#### 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 <b>gs_name_64</b> argument cannot be read by the caller, or the <b>return_va_64</b> or <b>return_length_64</b> argument cannot be written by the caller.
SS\$_CHANVIO	The specified channel was assigned from a more privileged access mode.
SS\$_ENDOFFILE	The <b>file_offset_64</b> 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\$ _GDLSEC_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.



## System Service Descriptions \$CRMPSC\_GFILE\_64 (Alpha Only)

SS\$_IVSECIDCTL	The match control field of the global section identification is invalid.
SS\$_LEN_NOTBLKMULT	The <b>length_64</b> or the <b>map_length_64</b> 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 SECSM_CRF is not set.
SS\$_OFF_NOTBLKALGN	The <b>file_offset_64</b> or <b>section_offset_64</b> argument is not virtual disk block aligned.
SS\$_OFFSET_TOO_BIG	The <b>section_offset_64</b> 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 <b>gs_name_64</b> 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 SECSM_NO_OVERMAP is set, or the existing underlying page cannot be deleted because it is associated with a buffer object.
SS\$_VA_NOTPAGALGN	The <b>start_va_64</b> argument is not CPU-specific page aligned.

---

## \$CRMPSC\_GPFILE\_64 (Alpha Only)

### Create and Map Global Page File Section

On Alpha 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	SECSK_MATALL	Match all versions of the section.

---

## System Service Descriptions \$CRMPSC\_GPFILE\_64 (Alpha Only)

Value	Symbolic Name	Match Criteria
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:

World				Group				Owner				System			
D	E	W	R	D	E	W	R	D	E	W	R	D	E	W	R
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

ZK-1706-GE

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.

## System Service Descriptions

### \$CRMPSC\_GPFILE\_64 (Alpha Only)

#### 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 SYSS\$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
VASC_P0	Program region
VASC_P1	Control region
VASC_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 SYSS\$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.

## System Service Descriptions \$CRMPSC\_GPFILE\_64 (Alpha Only)

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
<code>SEC\$M_DZRO</code>	Pages are demand-zero pages. By default, this flag is always present in this service and cannot be disabled.
<code>SEC\$M_EXPREG</code>	Pages are mapped into the first available space at the current end of the specified region. <code>SEC\$M_EXPREG</code> cannot be specified with the <code>SEC\$M_NO_OVERMAP</code> flag.
<code>SEC\$M_GBL</code>	Pages form a global section. By default, this flag is always present in this service and cannot be disabled.
<code>SEC\$M_NO_OVERMAP</code>	Pages cannot overmap existing address space. By default, pages can overmap existing address space. <code>SEC\$M_NO_OVERMAP</code> cannot be specified with the <code>SEC\$M_EXPREG</code> flag.
<code>SEC\$M_PAGFIL</code>	Pages form a global page file section. By default, this flag is always present in this service and cannot be disabled.
<code>SEC\$M_PERM</code>	Global section is permanent. By default, global sections are temporary.
<code>SEC\$M_SYSGBL</code>	Pages form a system global section. By default, pages form a group global section.
<code>SEC\$M_WRT</code>	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 for future use by Compaq 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

## System Service Descriptions

### \$CRMPSC\_GPFILE\_64 (Alpha Only)

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 nonzero, the condition value SSS\_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 SSS\_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 SSS\_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.

## System Service Descriptions \$CRMPSC\_GPFILE\_64 (Alpha Only)

### 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

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 <b>gs_name_64</b> argument cannot be read by the caller, or the <b>return_va_64</b> or <b>return_length_64</b> 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.

## System Service Descriptions

### \$CRMPSC\_GPFILE\_64 (Alpha Only)

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 <b>length_64</b> 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 <b>section_offset_64</b> 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 <b>section_offset_64</b> 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 <b>gs_name_64</b> 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 <b>start_va_64</b> argument is not CPU-specific page aligned.



## \$CRMPSC\_GPFN\_64 (Alpha Only) Create and Map Global Page Frame Section

On Alpha 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_gpfm_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.

## System Service Descriptions

### \$CRMPSC\_GPFN\_64 (Alpha Only)

Value	Symbolic Name	Match Criteria
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:

World				Group				Owner				System			
D	E	W	R	D	E	W	R	D	E	W	R	D	E	W	R
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

ZK-1706-GE

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)  
 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)  
 access: read only  
 mechanism: by value

Length of the page frame section in CPU-specific pages.

## System Service Descriptions \$CRMPSC\_GPFN\_64 (Alpha Only)

### 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 SYSSSTARLET\_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
VASC_P0	Program region
VASC_P1	Control region
VASC_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 SYSSSTARLET\_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 SSS\_IVACMODE is returned if the caller is less privileged than the create mode for the region.

## System Service Descriptions

### \$CRMPSC\_GPFN\_64 (Alpha Only)

#### 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 \$CRMPSC\_GPFN\_64:

Flag	Description
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 for future use by Compaq and should be specified as 0. The condition value SSS\_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 nonzero, the condition value SSS\_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 SSS\_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 SSS\_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.

## System Service Descriptions

### \$CRMPSC\_GPFN\_64 (Alpha Only)

#### Related Services

\$CREATE\_GPFN, \$CREATE\_REGION\_64, \$CRMPSC, \$CRMPSC\_FILE\_64, \$CRMPSC\_GFILE\_64, \$CRMPSC\_GPFIL\_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 <b>gs_name_64</b> argument cannot be read by the caller, or the <b>return_va_64</b> or <b>return_length_64</b> 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).

## System Service Descriptions \$CRMPSC\_GPFN\_64 (Alpha Only)

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 <b>gs_name_64</b> 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 <b>SECSM_NO_OVERMAP</b> is set, or the existing underlying page cannot be deleted because it is associated with a buffer object.
SS\$_VA_NOTPAGALGN	The <b>start_va_64</b> argument is not CPU-specific page-aligned.

## System Service Descriptions

### \$CRMPSC\_PFN\_64 (Alpha Only)

---

## \$CRMPSC\_PFN\_64 (Alpha Only)

### Create and Map Private Page Frame Section

On Alpha 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,...);
```

#### 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
VASC_P0	Program region
VASC_P1	Control region
VASC_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.



## System Service Descriptions \$CRMPSC\_PFN\_64 (Alpha Only)

### page\_count

OpenVMS usage: CPU-specific page count  
type: longword (unsigned)  
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 SSS\_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:

## System Service Descriptions

### \$CRMPSC\_PFN\_64 (Alpha Only)

Flag	Description
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_PFNMAP	Pages form a page frame 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.

All other bits in the **flags** argument are reserved for future use by Compaq and should be specified as 0. The condition value SSS\_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 nonzero, the condition value SSS\_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.

## System Service Descriptions \$CRMPSC\_PFN\_64 (Alpha Only)

All global page frame sections are permanent. Pages mapped by SECSM\_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 SSS\_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 SSS\_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

SSS_NORMAL	The service completed successfully.
SSS_ACCVIO	The <b>return_va_64</b> argument or the <b>return_length_64</b> argument cannot be written by the caller.
SSS_INSFWSL	The process's working set limit is not large enough to accommodate the increased virtual address space.
SSS_IVACMODE	The caller's mode is less privileged than the create mode associated with the region.
SSS_IVREGID	Invalid region ID specified.
SSS_IVSECFLG	An invalid flag, a reserved flag, or an invalid combination of flags and arguments was specified.
SSS_NOPFNMAP	The process does not have the privilege to create a section starting at a specific physical page frame number (PFNMAP).
SSS_NOSHPTS	The region ID of a shared page table region was specified.
SSS_PAGNOTINREG	A page in the specified range is not within the specified region.
SSS_PAGOWNVIO	A page in the specified input address range is owned by a more privileged access mode.

## System Service Descriptions

### \$CRMPSC\_PFN\_64 (Alpha Only)

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 <b>start_va_64</b> argument is not CPU-specific page aligned.

## \$CVT\_FILENAME (Alpha Only) 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.

## System Service Descriptions

### \$CVT\_FILENAME (Alpha Only)

For conversion from RMS format to ACP-QIO format, only the CVTFNMSV\_NO\_DELIMITERS flag is valid.

For conversion from ACP-QIO format to RMS format, legal flags are CVTFNMSV\_WORD\_CHARS and CVTFNMSV\_NO\_DELIMITERS (defined by the \$CVTFNMDEF macro).

Flag	Description
CVTFNMSV_WORD_CHARS	Input source string contains word-width UCS-2 characters (ACPQIO_TO_RMS conversion only).
CVTFNMSV_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.
CVTFNMSV_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. (Refer to the *Guide to OpenVMS File Applications*.)

For ACPQIO\_TO\_RMS conversion, if the output string contains word-width characters, the CVTFNMSV\_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, SYSSCVT\_FILENAME sets the bit corresponding to CVTFNMSV\_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 CVTFNMSV\_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. (Refer to the *Guide to OpenVMS File Applications* for details on ODS-5 file specifications.)

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 CVTFNMSV\_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 CVTFNMSV\_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.

## System Service Descriptions

### \$CVT\_FILENAME (Alpha Only)

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.

Question 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, refer to: [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 <b>srcstr</b> argument. Either the input string has word-width characters but odd byte-length (ACPQIO_TO_RMS only), or the CVTFNMSV_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. Refer to the *OpenVMS Programming Concepts Manual* for additional information.

**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, \$SENDERR, \$SENDJBC, \$SENDJBCW, \$SENDOPR

**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 systems, you specified an invalid channel number that your process never assigned.
SS\$_NOICHAN	On Alpha 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.

## System Service Descriptions

### \$DCLAST

---

## \$DCLAST

### Declare AST

Queues an asynchronous system trap (AST) for the calling access mode or for a less privileged access mode.

On Alpha 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 (Alpha)  
mechanism: by 32-bit reference (VAX)

AST service routine to be executed. On Alpha systems, the **astadr** argument is the 32- or 64-bit address of this routine. On VAX systems, the **astadr** argument is the 32-bit address of this routine.

#### **astprm**

OpenVMS usage: `user_arg`  
type: quadword (unsigned)  
access: read only  
mechanism: by 64-bit value (Alpha)  
mechanism: by 32-bit value (VAX)

AST parameter to be passed to the AST routine specified by the **astadr** argument. On Alpha systems, the **astprm** argument is a quadword value containing this parameter. On VAX systems, the **astprm** argument is a longword 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

On Alpha systems, specifies the address of a routine to receive control when a Change Mode to User or Change Mode to Supervisor instruction trap occurs.

On VAX systems, specifies the address of a routine to receive control when a Change Mode to User or Change Mode to Supervisor instruction trap occurs, or when a compatibility mode fault occurs.

#### Format

```
SYS$DCLCMH  address ,[prvhnd] ,[type]
```

#### C Prototype

```
int sys$dclcmh (int (*address)(__unknown_params), void *(*(prvhnd)), char type);
```

#### Arguments

##### address

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 **address** argument is the exception handling code in the address space of the calling process.

If you specify the **address** 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 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 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.

On VAX systems, the \$DCLCMH service specifies the address of a routine to receive control when (1) a Change Mode to User or Change Mode to Supervisor instruction trap occurs, or (2) a compatibility mode fault occurs. 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

## System Service Descriptions

### \$DCLCMH

#### 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.
‡SS\$_IVSSRQ	The call to the service is invalid because it attempted to declare a compatibility mode handler on Alpha systems.

---

‡Alpha specific

## \$DCLEXH Declare Exit Handler

Declares an exit handling routine that receives control when an image exits.

### Format

SYS\$DCLEXH desblk

### C Prototype

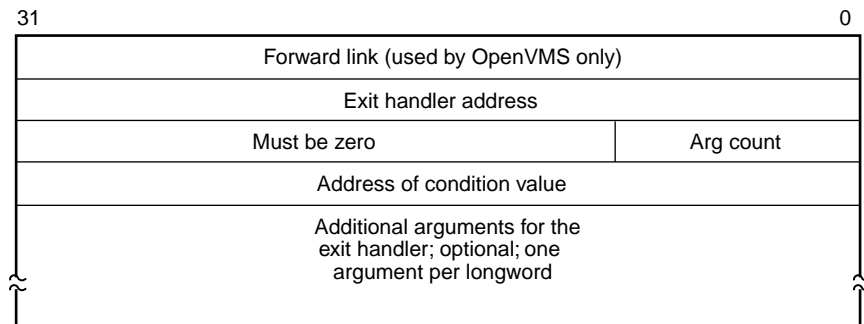
```
int sys$dclcxh (void *desblk);
```

### 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:



VM-0537A-AI

### 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.

## System Service Descriptions

### \$DCLEXH

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, refer to the *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**

\$SCANEXH, \$CREPRC, \$DELPRC, \$EXIT, \$FORCEX, \$GETJPI, \$GETJPIW,  
\$HIBER, \$PROCESS\_SCAN, \$RESUME, \$SETPRI, \$SETPRN, \$SETPRV,  
\$SETRWM, \$SUSPND, \$WAKE

The Cancel Exit Handler (\$SCANEXH) 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.

## System Service Descriptions

### \$DELETE\_BUFOBJ (Alpha Only)

---

## \$DELETE\_BUFOBJ (Alpha Only)

### Delete Buffer Object

On Alpha 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

SSS_NORMAL	The service completed successfully.
SSS_ACCVIO	The <b>buffer_handle_64</b> argument cannot be read by the caller.
SSS_BADPARAM	The <b>buffer_handle_64</b> argument is not a valid buffer handle.



**System Service Descriptions**  
**\$DELETE\_BUF OBJ (Alpha Only)**

SS\$\_NOPRIV

The buffer object was created by a more privileged access mode than the caller's access mode.

## System Service Descriptions

### \$DELETE\_GALAXY\_LOCK (Alpha Only)

---

### \$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, refer to the *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

<code>SS\$NORMAL</code>	Normal completion.
<code>SS\$BADLCKTBL</code>	Galaxy lock table is corrupt.
<code>SS\$IVLOCKID</code>	Invalid lock id.
<code>SS\$IVLOCKTBL</code>	Invalid lock table.
<code>SS\$LOCKINUSE</code>	Invalid operation; lock is in use.

**System Service Descriptions**  
**\$DELETE\_GALAXY\_LOCK (Alpha Only)**

SS\$\_NOCMKRNL  
SS\$\_NOSHMEM

Operation requires CMKRNL privilege.  
Operation requires SHMEM privilege.

## System Service Descriptions

### \$DELETE\_GALAXY\_LOCK\_TABLE (Alpha Only)

---

## \$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, refer to the *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, refer to the *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 ITMSL\_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.

## System Service Descriptions

### \$DELETE\_INTRUSION

Item	Description
CIA\$_USER_CRITERIAL	Address of a buffer, 1-1058 bytes long, containing the intruder or suspect.

If the CIA\$\_SCSNODE\_LIST item is present, it is the address of a list of 8-character null-padded SCS nodenames for which intrusions are to be deleted. If this item is absent, the service deletes the specified intrusion records for all nodes in the cluster. Multiple CIA\$\_SCSNODE\_LIST items are permitted.

#### 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 names 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.
CIA\$M_ITEMLIST	If FALSE, the <b>user_criteria</b> 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 <b>user_criteria</b> argument cannot be read.
SS\$ BADBUFLN	The length of the <b>user_criteria</b> argument is out of range.
SS\$ BADPARAM	An invalid flag was specified in the <b>flags</b> 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 <b>rem_node</b> 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_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.

## System Service Descriptions

### \$DELETE\_PROXY

#### 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 <b>rem_node</b> , <b>rem_user</b> , <b>local_user</b> , or <b>flags</b> argument cannot be read by the service.
SS\$_BADBUFLEN	The length of the <b>rem_node</b> , <b>rem_user</b> , or <b>local_user</b> argument was out of range.
SS\$_BADPARAM	An invalid flag was specified in the <b>flags</b> 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 <b>rem_node</b> and <b>rem_user</b> 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 Only) Delete a Virtual Region

On Alpha 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 VA\$C\_P0, VA\$C\_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 SYSS\$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.

## System Service Descriptions

### \$DELETE\_REGION\_64 (Alpha Only)

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.

## System Service Descriptions \$DELETE\_REGION\_64 (Alpha Only)

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

<code>SS\$_NORMAL</code>	Successful completion. All pages within the region have been deleted as well as the region itself.
<code>SS\$_ACCVIO</code>	The <b>return_va_64</b> argument or the <b>return_length_64</b> argument cannot be written by the caller.
<code>SS\$_IVREGID</code>	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.
<code>SS\$_REGOWNVIO</code>	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.
<code>SS\$_PAGOWNVIO</code>	A page within the specified region is owned by a more privileged access mode than the access mode of the caller.
<code>SS\$_VA_IN_USE</code>	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 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 (Alpha)

mechanism: by 32-bit descriptor-fixed-length string descriptor (VAX)

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-bit address (on VAX systems) or the 32- or 64-bit address (on Alpha systems) 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 (Alpha)

mechanism: by 32-bit descriptor-fixed-length string descriptor (VAX)

Logical name to be deleted. The **lognam** argument is the 32-bit address (on VAX systems) or the 32- or 64-bit address (on Alpha systems) 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 (Alpha)  
mechanism: by 32-bit reference (VAX)

Access mode to be used in the delete operation. The **acmode** argument is the 32-bit address (on VAX systems) or the 32- or 64-bit address (on Alpha systems) 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

## System Service Descriptions

### \$DELLNM

#### 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 <b>lognam</b> argument specifies a string whose length is not in the required range of 1 through 255 characters.
SS\$ _IVLOGTAB	The <b>tabnam</b> 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 occurs 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

## System Service Descriptions

### \$DELMBX

#### 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]

### 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.

### 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.

## System Service Descriptions

### \$DELPRC

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

\$SCANEXH, \$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.

## System Service Descriptions \$DELPRC

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 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 (SECRETVA) 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.

## System Service Descriptions

### \$DELTVA\_64 (Alpha Only)

---

## \$DELTVA\_64 (Alpha Only)

### Delete Virtual Address Space

On Alpha 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
VASC_P0	Program region
VASC_P1	Control region
VASC_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 VAS\_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 \$DELTV\_A\_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 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 \$DELTV\_A\_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 \$DELTV\_A\_64 service returns the length in bytes of the virtual address range deleted.

## System Service Descriptions

### \$DELTVA\_64 (Alpha Only)

#### 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

<code>SS\$_NORMAL</code>	The service completed successfully.
<code>SS\$_ACCVIO</code>	The <b>return_va_64</b> argument or the <b>return_length_64</b> argument cannot be written by the caller.
<code>SS\$_IVREGID</code>	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.
<code>SS\$_LEN_NOTPAGMULT</code>	The <b>length_64</b> 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.
<code>SS\$_PAGNOTINREG</code>	A page in the specified range is not within the specified region.
<code>SS\$_PAGOWNVIO</code>	A page in the specified range is owned by an access mode more privileged than the access mode of the caller.
<code>SS\$_VA_NOTPAGALGN</code>	The <b>start_va_64</b> argument is not a CPU-specific page table page aligned address; or, for shared page table regions, is not page table page aligned.

**System Service Descriptions**  
**\$DELTVA\_64 (Alpha Only)**

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 (SENQ) service.

On Alpha 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 LCKSM\_DEQALL flag in the **flags** argument.

When you specify the LCKSM\_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 LCKSM\_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 LCKSM\_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 LCKSM\_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 SENQ 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 LCKSM\_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 (Alpha)  
 mechanism: by 32-bit reference (VAX)

Lock value block for the resource associated with the lock to be dequeued. The **valblk** argument is the 32- or 64-bit address (on Alpha systems) or the 32-bit (on VAX systems) of the 16-byte lock value block. When you specify the LCKSM\_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 LCKSM\_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 LCKSM\_DEQALL flag in the **flags** argument.

## System Service Descriptions

### \$DEQ

A symbolic name for each flag bit is defined by the `$LCKDEF` macro. The following table describes each flag:

Flag	Description
LCK\$M_DEQALL	<p>When you specify this flag, \$DEQ dequeues multiple locks, depending on the value of the <b>lkid</b> argument. Refer to the description of the <b>lkid</b> argument for details. The <b>acmode</b> 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.</p>
LCK\$M_CANCEL	<p>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 <code>SS\$_NORMAL</code>.</p> <p>If you attempt to cancel a granted lock, the request fails and \$DEQ returns the condition value <code>SS\$_CANCELGRANT</code>. There are two types of waiting requests that can be canceled:</p> <ul style="list-style-type: none"><li>• A request for a new lock</li><li>• A request to convert an existing lock</li></ul> <p>When canceling a new lock request, the following action is taken:</p> <ul style="list-style-type: none"><li>• If a completion asynchronous system trap (AST) was requested, the AST is queued for delivery and <code>SS\$_ABORT</code> is stored in the lock status block.</li></ul> <p>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:</p> <ul style="list-style-type: none"><li>• 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.</li><li>• If a completion AST was specified by the conversion request, the completion AST is queued for delivery with <code>SS\$_CANCEL</code> status stored in the lock status block that was specified by the conversion request.</li></ul> <p>If you specify the LCK\$M_DEQALL flag, the LCK\$M_CANCEL flag is ignored.</p>

Flag	Description
LCKSM_INVVALBLK	<p>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.</p> <p>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 LCKSM_CANCEL flag.</p>

## 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 SSS\_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 SSS\_NORMAL and the following specific action is taken:

- If a completion AST was requested, the completion AST is queued for delivery with SSS\_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 SSS\_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 SSS\_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 LCKSM\_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.

## System Service Descriptions

### \$DEQ

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 *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 <b>valblk</b> argument cannot be accessed by the caller.
SS\$ _CANCELGRANT	The LCK\$M_CANCEL flag in the <b>flags</b> 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

### Scan for Device Paths

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**.

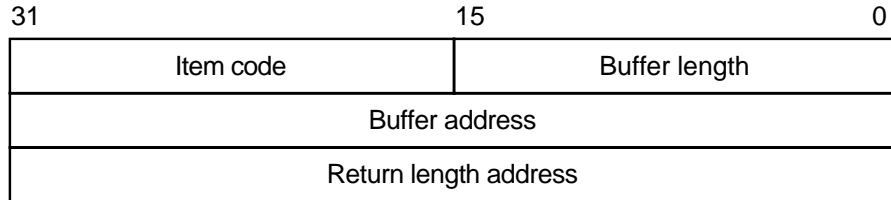
## System Service Descriptions

### \$DEVICE\_PATH\_SCAN

#### 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:



ZK-5186A-GE

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

Placelholding argument reserved to Compaq.

## 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 SSS\_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 <b>itmlst</b> 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 <b>devnam</b> nor <b>chan</b> argument was specified.

## System Service Descriptions

### \$DEVICE\_PATH\_SCAN

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 (DVIS\_ALLDEVNAM). Only uppercase characters are accepted.

## System Service Descriptions

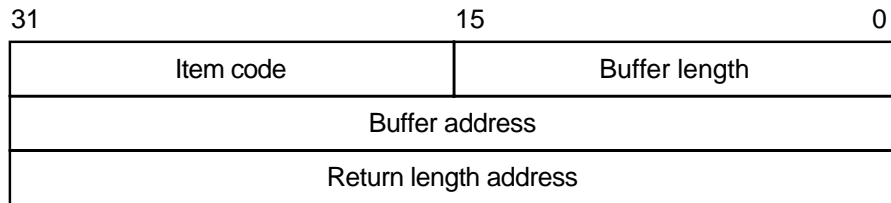
### \$DEVICE\_SCAN

#### 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:



ZK-5186A-GE

The following table defines the item descriptor fields:

Descriptor Field	Definition
Buffer length	A word containing a user-supplied integer specifying the length (in bytes) of the longword from which \$DEVICE_SCAN 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-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.
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. Compaq 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

## System Service Descriptions

### \$DEVICE\_SCAN

#### Condition Values Returned

SS\$_NORMAL	The service completed successfully.
SS\$_ACCVIO	The <b>search_devnam</b> , <b>itmlst</b> , or <b>ctxt</b> argument cannot be read by the caller, or the <b>retlen</b> , <b>return_devnam</b> , or <b>ctxt</b> argument cannot be written by the caller.
SS\$_BADPARAM	The <b>ctxt</b> 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 systems, this service accepts 64-bit addresses.

### Format

```
SYS$DGBLSC [flags] ,gsdnam ,[ident]
```

### C Prototype

```
int sys$dgbpsc (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 (Alpha)  
 mechanism: by 32-bit descriptor–fixed-length string descriptor (VAX)

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.

## System Service Descriptions

### \$DGBLSC

Names are first subject to a logical name translation, after the application of the prefix `GBLS` 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 *OpenVMS Programming Concepts Manual*.

#### **ident**

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

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 (on Alpha systems) or the 32-bit address (on VAX systems) 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	SECSK_MATALL	Match all versions of the section.
1	SECSK_MATEQU	Match only if major and minor identifications match.
2	SECSK_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.

On VAX systems, a section located in memory that is shared by multiple processors can be marked for deletion only by a process running on the same processor that created the section.

**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, \$SEXPREG, \$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.

## System Service Descriptions

### \$DGBLSC

SS\$_NOTCREATOR	The section is in memory shared by multiple processors and was created by a process on another processor.
†SS\$_SHMNOTCNCT	The shared memory named in the <b>name</b> 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 <b>gsdnam</b> string exceeded the allowed depth of 10.

---

†VAX specific

## \$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.

## System Service Descriptions

### \$DISMOU

Flag	Meaning
DMTSM_CLUSTER	<p>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.</p> <p>DMTSM_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.</p> <p>DMTSM_CLUSTER has no effect if the system is not a member of a cluster. DMTSM_CLUSTER applies only to disks.</p>
DMTSM_FORCE	<p>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.</p>
DMTSM_MINICOPY_OPTIONAL	<p>\$DISMOU takes place, regardless of whether minicopy is enabled on the disk.</p>
DMTSM_MINICOPY_REQUIRED	<p>\$DISMOU fails if minicopy has not been enabled on the disk.</p>
DMTSM_NOUNLOAD	<p>Specifies that the volume is not to be physically unloaded after the dismount. If both the DMTSM_UNLOAD and DMTSM_NOUNLOAD flags are specified, the DMTSM_NOUNLOAD flag is ignored. If neither flag is specified, the volume is physically unloaded, unless the DMTSM_NOUNLOAD flag was specified on the \$MOUNT system service or the /NOUNLOAD qualifier was specified on the MOUNT command when the volume was mounted.</p>
DMTSM_OVR_CHECKS	<p>Specifies that the volume should be dismounted without checking for open files, spooled devices, installed images, or installed swap and page files.</p>
DMTSM_UNIT	<p>The specified device, rather than the entire volume set, is dismounted.</p>

Flag	Meaning
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 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 SSS\_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.

## System Service Descriptions

### \$DISMOU

#### 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\_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.

## System Service Descriptions

### \$DISPLAY\_PROXY

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:

31			0
	Proxy node length	Proxy user length	
	Default user length	Local users count	

ZK-6169A-GE

The following table defines the **buffer\_sizes** fields:

Descriptor Field	Definition
Proxy user length	Return length (in bytes) of the <b>rem_user</b> 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 <b>rem_user</b> argument.
Proxy node length	Return length (in bytes) of the <b>rem_node</b> 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 <b>rem_node</b> 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 <b>default_user</b> 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.



**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 SSS\_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 SSS\_NOMOREITEMS when information about all of the matching proxies has been returned. No proxy information is returned from the call that returns the SSS\_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

SSS_NORMAL	The service completed successfully.
------------	-------------------------------------

## System Service Descriptions

### \$DISPLAY\_PROXY

SS\$_ACCVIO	The <b>rem_node</b> or <b>rem_user</b> argument cannot be read by the service; or the <b>buffer_sizes</b> , <b>proxy_node</b> , <b>proxy_user</b> , <b>default_user</b> , or <b>local_users</b> argument cannot be written by the service; or the <b>context</b> argument cannot be read or written by the service.
SS\$_BADBUFLEN	The length of the <b>rem_node</b> , <b>rem_user</b> , <b>proxy_node</b> , <b>proxy_user</b> , <b>default_user</b> , or <b>local_users</b> argument was out of range.
SS\$_BADCONTEXT	The <b>context</b> argument did not contain a 0 on the first call to the service, or the <b>context</b> argument's value changed between consecutive calls to the service.
SS\$_NOMOREITEMS	Information about all proxies matching the specification of the <b>rem_node</b> and <b>rem_user</b> 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 <b>rem_node</b> and <b>rem_user</b> 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 SSS\_NORMAL.

#### Required Access or Privileges

Delete access is required.

#### Required Quota

None

#### Related Services

\$ASCEFC, \$CLREF, \$DACEFC, \$READEF, \$SETEF, \$WAITFR, \$WFLAND, \$WFLOR

## System Service Descriptions

### \$DLCEFC

#### 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.



## \$DNS (VAX Only) Distributed Name Service Clerk

On VAX systems, the DIGITAL Distributed Name Service (DECdns) clerk allows client applications to store resource names and addresses.

The \$DNS system service completes asynchronously; that is, it returns to the client immediately after making a name service call. The status returned to the client call indicates whether a request was successfully queued to the name service.

The DIGITAL Distributed Name Service Clerk Wait (\$DNSW) system service is the synchronous equivalent of \$DNS. \$DNSW is identical to \$DNS in every way except that \$DNSW returns to the caller after the operation completes.

### Format

```
SYS$DNS [efn] ,func ,itmlst ,[dnsb] ,[astadr] ,[astprm]
```

### Arguments

#### **efn**

OpenVMS usage: ef\_number  
type: longword (unsigned)  
access: read only  
mechanism: by value

Number of the event flag to be set when \$DNS completes. The **efn** argument is a longword containing this number. The **efn** argument is optional; if not specified, event flag 0 is set.

When \$DNS begins execution, it clears the event flag. Even if the service encounters an error and completes without queuing a name service request, the specified event flag is set.

#### **func**

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

Function code specifying the action that \$DNS is to perform. The **func** argument is a longword containing this function code.

A single call to \$DNS can specify one function code. Most function codes require or allow for additional information to be passed in the call with the **itmlst** 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.

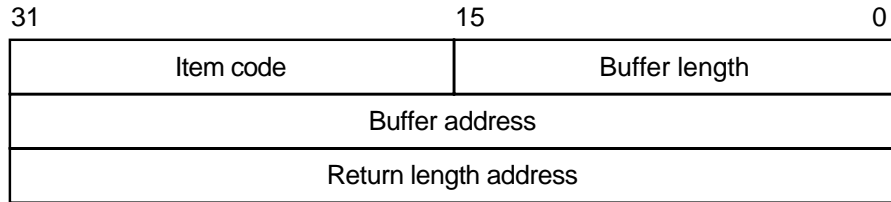
## System Service Descriptions

### \$DNS (VAX Only)

The item list consists of one or more item descriptors, each of which is three longwords.

The descriptors can be in any order in the item list. Each item descriptor specifies an item code. Item codes are specified as either input or output parameters. Input parameters modify functions, set context, or describe the information to be returned. Output parameters return the requested information. The item list is terminated by a longword of 0.

The item list is a standard format item list. The following figure depicts the general structure of an item descriptor:



ZK-5186A-GE

The following table defines the item descriptor fields:

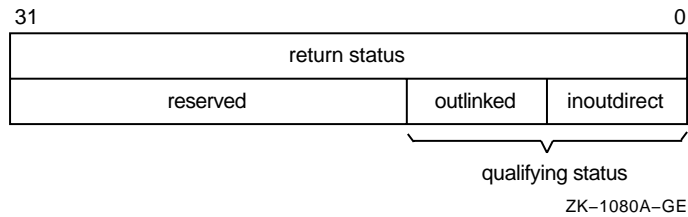
Descriptor Field	Definition
Buffer length	A word specifying the length of the buffer; the buffer either supplies information to be used by \$DNS or receives information from \$DNS. The required length of the buffer varies, depending on the item code specified. Each item code description specifies the 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.
Buffer address	A longword containing the address of the buffer that specifies or receives the information.
Return length address	A longword containing the address of a word specifying the actual length (in bytes) of the information returned by \$DNS. The information resides in a buffer identified by the buffer address field. The field applies to output item list entries only and must be 0 for input entries. If the return length address is 0, it is ignored.

#### **dnsb**

OpenVMS usage: dns\_status\_block  
 type: quadword (unsigned)  
 access: write only  
 mechanism: by reference

Status block to receive the final completion status of the \$DNS operation. The **dnsb** argument is the address of the quadword \$DNS status block.

The following figure depicts the structure of a \$DNS status block:



The following table defines the status block fields:

Status Block Field	Definition
Return status	Set on completion of a DECdns clerk request to indicate the success or failure of the operation. Check the qualifying status word for additional information about a request marked as successful.
Qualifying status	This field consists of two flags that provide additional information about a successful request to the DECdns server.

The two qualifying status flags, `DNSSV_DNSB_INOUTDIRECT` and `DNSSV_DNSB_OUTLINKED`, are defined as follows:

- `DNSSV_DNSB_INOUTDIRECT`—Indicates whether the members were found in the top-level group or in one of the subgroups. The values are defined as follows:
  - 1: The member was found in the top-level group.
  - 0: The member was found in one of the subgroups of the top-level group.
- `DNSSV_DNSB_OUTLINKED`—If set, indicates that one or more soft links were encountered while resolving the name specified in a call.

Functions that access the DECdns server return a qualifying status. Name conversion functions do not return qualifying status.

**astadr**

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

Asynchronous system trap (AST) routine to be executed when I/O completes. The **astadr** argument is the address of the AST routine.

The AST routine executes in the access mode of the caller of \$DNS.

**astprm**

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

Asynchronous system trap parameter passed to the AST service routine. The **astprm** argument is a longword value containing the AST parameter.

## System Service Descriptions

### \$DNS (VAX Only)

#### Function Codes

##### **DNS\$\_ADD\_REPLICA**

This request adds a directory replica in the specified clearinghouse. Specify the item code `DNS$_REPLICATYPE` as either a secondary directory (`DNS$_SECONDARY`) or a read-only directory (`DNS$_READONLY`).

You must have control access to the directory being replicated and write access to the new replica's clearinghouse.

You must specify the following input value item codes:

`DNS$_CLEARINGHOUSE`  
`DNS$_DIRECTORY`  
`DNS$_REPLICATYPE`

You can specify the following input value item codes:

`DNS$_CONF`  
`DNS$_WAIT`

\$DNS returns the following qualifying status:

`DNS$_DNSB_OUTLINKED`

##### **DNS\$\_ALLOW\_CH**

This request permits a directory to store clearinghouse objects. This request takes as input the name of a directory (`DNS$_DIRECTORY`).

You must have control access to the parent directory.

You must specify the following input value item code:

`DNS$_DIRECTORY`

You can specify the following input value item codes:

`DNS$_CONF`  
`DNS$_WAIT`

##### **DNS\$\_CREATE\_DIRECTORY**

This request creates a master directory in the specified clearinghouse.

You must have write or control access to the parent directory and write access to the master replica's clearinghouse.

You must specify the following input value item code:

`DNS$_DIRECTORY`

You can specify the following input value item codes:

`DNS$_CLEARINGHOUSE`  
`DNS$_WAIT`

You can specify the following output value item code:

`DNS$_OUTCTS`

##### **DNS\$\_CREATE\_LINK**

This request creates a soft link to a directory, object, soft link, or clearinghouse in the namespace. Specify the target to which the soft link points in the `DNS$_TARGETNAME` item code. Use the `DNS$_RESOLVE_NAME` function code to check the existence of the target.

## System Service Descriptions \$DNS (VAX Only)

You must have write or control access to the directory in which the soft link is being created.

You must specify the following input value item codes:

DNS\$\_LINKNAME  
DNS\$\_TARGETNAME

You can specify the following input value item codes:

DNS\$\_CONF  
DNS\$\_EXPIRETIME  
DNS\$\_EXTENDTIME  
DNS\$\_WAIT

You can specify the following output value item code:

DNS\$\_OUTCTS

### **DNS\$\_CREATE\_OBJECT**

This request creates an object in the namespace. Initially, the object has the attributes of DNS\$CTS, DNS\$UTS, DNS\$Class, DNS\$ClassVersion, and DNS\$ACS. The name service creates the DNS\$CTS, DNS\$UTS, and DNS\$ACS attributes. The client application supplies the DNS\$Class and DNS\$ClassVersion attributes. You can add attributes using the DNS\$\_MODIFY\_ATTRIBUTE function.

The DECdns clerk cannot guarantee that an object has been created. Another DNS\$\_CREATE\_OBJECT request could supersede the object created by your call. To verify an object creation, wait until the directory is skulked and then check to see if the requested object is present. If the value of the directory's DNS\$ALLUPTO attribute is greater than the DNS\$CTS of the object, your object has been successfully created.

If specified, DNS\$\_OUTCTS holds the creation timestamp of the newly created object.

This function code returns the following:

SS\$\_NORMAL  
DNS\$\_ENTRYEXISTS  
DNS\$\_INVALID\_OBJECTNAME  
DNS\$\_INVALID\_CLASSNAME  
Any condition listed in the section Condition Values Returned

You must have write access to the directory where the object will reside.

You must specify the following input value item codes:

DNS\$\_CLASS  
DNS\$\_OBJECTNAME  
DNS\$\_VERSION

You can specify the following input value item codes:

DNS\$\_CONF  
DNS\$\_WAIT

You can specify the following output value item code:

DNS\$\_OUTCTS

## System Service Descriptions

### \$DNS (VAX Only)

#### **DNS\$\_DELETE\_DIRECTORY**

This request removes a directory from the namespace.

You must have delete access to the directory being deleted and write, control, or delete access to the parent directory.

You must specify the following input value item code:

DNS\$\_DIRECTORY

You can specify the following input value item codes:

DNS\$\_CONF

DNS\$\_WAIT

#### **DNS\$\_DELETE\_OBJECT**

This request removes the specified object from the namespace.

This function code returns the following:

SS\$\_NORMAL

DNS\$\_INVALID\_OBJECTNAME

Any condition listed in the section Condition Values Returned

You must have delete access to the object.

You must specify the following input value item code:

DNS\$\_OBJECTNAME

You can specify the following input value item codes:

DNS\$\_CONF

DNS\$\_WAIT

\$DNS returns the following qualifying status:

DNS\$\_V\_DNSB\_OUTLINKED

#### **DNS\$\_DISALLOW\_CH**

This request prevents a directory from storing clearinghouse objects. This request takes as input the name of a directory (DNS\$\_DIRECTORY).

You must have control access to the parent directory, and read or control access to any child directories.

You must specify the following input value item code:

DNS\$\_DIRECTORY

You can specify the following input value item codes:

DNS\$\_CONF

DNS\$\_WAIT

#### **DNS\$\_ENUMERATE\_ATTRIBUTES**

This request returns a set of attribute names in DNS\$\_OUTATTRIBUTESET that are associated with the directory, object, soft link, or clearinghouse. Specify the entry type in the DNS\$\_LOOKINGFOR item code. The function returns either DNS\$\_K\_SET or DNS\$\_K\_SINGLE along with the set of attribute names.

To manipulate the attribute names returned by this call, you should use the DNS\$\_REMOVE\_FIRST\_SET\_VALUE run-time library routine.

## System Service Descriptions \$DNS (VAX Only)

The DECdns clerk enumerates attributes in alphabetical order. A return status of DNS\$\_MOREDATA implies that not all attributes have been enumerated. You should make further calls, setting DNS\$\_CONTEXTVARNAME to the last attribute in the set returned, until the procedure returns S\$\$\_NORMAL.

This function code returns the following:

- SS\$\_NORMAL
- DNS\$\_MOREDATA
- DNS\$\_INVALID\_ENTRYNAME
- DNS\$\_INVALID\_CONTEXTNAME

Any condition listed in the section Condition Values Returned

You must have read access to the directory, object, soft link, or clearinghouse.

You must specify the following input value item codes:

- DNS\$\_ENTRY
- DNS\$\_LOOKINGFOR

You must specify the following output value item code:

- DNS\$\_OUTATTRIBUTESET

You can specify the following input value item codes:

- DNS\$\_CONF
- DNS\$\_CONTEXTVARNAME
- DNS\$\_WAIT

You can specify the following output value item code:

- DNS\$\_CONTEXTVARNAME

\$DNS returns the following qualifying status:

- DNS\$V\_DNSB\_OUTLINKED

### **DNS\$\_ENUMERATE\_CHILDREN**

This request takes as input a directory name with an optional simple name that uses a wildcard. The DECdns clerk matches the input against child directory entries in the specified directory.

The DECdns clerk returns a set of simple names of child directories in the target directory that match the name with the wildcard. A null set is returned when there is no match or the directory has no child directories.

To manipulate the values returned by this call, you should use the DNS\$REMOVE\_FIRST\_SET\_VALUE run-time routine. The value returned is a simple name.

The clerk enumerates child directories in alphabetical order. If the call returns DNS\$\_MOREDATA, not all child directories have been enumerated and the client should make further calls, setting DNS\$\_CONTEXTVARNAME to the last child directory in the set returned, until the procedure returns S\$\$\_NORMAL. Subsequent calls return the child directories, starting with the directory specified in DNS\$\_CONTEXTVARNAME and continuing in alphabetical order.

## System Service Descriptions

### \$DNS (VAX Only)

This function code returns the following:

SS\$\_NORMAL  
DNS\$\_MOREDATA  
DNS\$\_INVALID\_DIRECTORYNAME  
DNS\$\_INVALID\_CONTEXTNAME  
DNS\$\_INVALID\_WILDCARDNAME

You must have read access to the parent directory.

You must specify the following input value item code:

DNS\$\_DIRECTORY

You must specify the following output value item code:

DNS\$\_OUTCHILDREN

You can specify the following input value item codes:

DNS\$\_CONF  
DNS\$\_CONTEXTVARNAME  
DNS\$\_WAIT  
DNS\$\_WILDCARD

You can specify the following output value item code:

DNS\$\_CONTEXTVARNAME

\$DNS returns the following qualifying status:

DNS\$\_DNSB\_OUTLINKED

#### **DNS\$\_ENUMERATE\_OBJECTS**

This request takes as input the directory name, a simple name that can use a wildcard, and a class name that uses a wildcard. The DECdns clerk matches these against objects in the directory. If a wildcard and class filter are not specified, all objects in the directory are returned.

The function returns (in DNS\$\_OUTOBJECTS) a set of simple names of object entries in the directory that match the name with the wildcard. The function also returns the class of the object entries, if specified with DNS\$\_RETURNCLASS. If no object entries match the wildcard or the directory contains no object entries, a null set is returned.

To manipulate the values returned by this call, you should use the DNS\$\_REMOVE\_FIRST\_SET\_VALUE run-time routine. The value returned is a simple name structure.

The clerk enumerates objects in alphabetical order. If the call returns DNS\$\_MOREDATA, not all objects have been enumerated and the client should make further calls, setting DNS\$\_CONTEXTVARNAME to the last object in the set returned, until the procedure returns SS\$\_NORMAL. If the class filter is specified, only those objects of the specified classes are returned.



This function code returns the following:

SS\$ \_NORMAL  
DNS\$ \_MOREDATA  
DNS\$ \_INVALID\_DIRECTORYNAME  
DNS\$ \_INVALID\_CONTEXTNAME  
DNS\$ \_INVALID\_WILDCARDNAME  
DNS\$ \_INVALID\_CLASSNAME

You must have read access to the directory.

You must specify the following input value item code:

DNS\$ \_DIRECTORY

You must specify the following output value item code:

DNS\$ \_OUTOBJECTS

You can specify the following input value item codes:

DNS\$ \_CLASSFILTER  
DNS\$ \_CONF  
DNS\$ \_CONTEXTVARNAME  
DNS\$ \_RETURNCLASS  
DNS\$ \_WAIT  
DNS\$ \_WILDCARD

You can specify the following output value item code:

DNS\$ \_CONTEXTVARNAME

\$DNS returns the following qualifying status:

DNS\$V\_DNSB\_OUTLINKED

#### **DNS\$ \_ENUMERATE\_SOFTLINKS**

This request takes as input the name of a directory and a wildcarded simple name. The DECdns clerk matches these against soft links in the directory. It returns (in DNS\$ \_OUTSOFTLINKS) a set consisting of simple names of soft links in the directory that match the wildcarded name. If no soft link entries match the wildcard or the directory contains no soft links, a null set is returned.

If no wildcard is specified, then all soft links in the directory are returned.

To manipulate the values returned by this call, use the DNS\$REMOVE\_FIRST\_SET\_VALUE run-time library routine. The value returned is a simple name.

The clerk enumerates soft links in alphabetical order. If the call returns DNS\$ \_MOREDATA, not all matching soft links have been enumerated and the client should make further calls, setting DNS\$ \_CONTEXTVARNAME to the last soft link in the set returned, until the procedure returns SSS\$ \_NORMAL.

This function code returns the following:

SS\$ \_NORMAL  
DNS\$ \_INVALID\_DIRECTORYNAME  
DNS\$ \_INVALID\_CONTEXTNAME  
DNS\$ \_INVALID\_WILDCARDNAME

You must have read access to the directory.

## System Service Descriptions

### \$DNS (VAX Only)

You must specify the following input value item code:

DNS\$\_DIRECTORY

You must specify the following output value item code:

DNS\$\_OUTSOFTLINKS

You can specify the following input value item codes:

DNS\$\_CONF

DNS\$\_CONTEXTVARNAME

DNS\$\_WAIT

DNS\$\_WILDCARD

You can specify the following output value item code:

DNS\$\_CONTEXTVARNAME

\$DNS returns the following qualifying status:

DNS\$\_DNSB\_OUTLINKED

#### **DNS\$\_FULL\_OPAQUE\_TO\_STRING**

This request converts a full name in opaque format to its equivalent in string format. To prevent the namespace nickname from being included in the string name, set the byte referred to by DNS\$\_SUPPRESS\_NSNAME to 1.

This function code returns the following:

SS\$\_NORMAL

DNS\$\_INVALIDNAME

You must specify the following input value item code:

DNS\$\_FROMFULLNAME

You must specify the following output value item code:

DNS\$\_TOSTRINGNAME

You can specify the following input value item code:

DNS\$\_SUPPRESS\_NSNAME

#### **DNS\$\_MODIFY\_ATTRIBUTE**

This request applies one update to the specified entry in the namespace. The update operations are as follows:

- Add or remove an attribute.
- Add or remove an attribute value from either a single-valued attribute or a set-valued attribute.

To add a value to a single-valued or set-valued attribute, specify a value in the DNS\$\_MODVALUE item code. If you do not specify a value for a single-valued attribute, you receive the error DNS\$\_INVALIDUPDATE. Single-valued attributes cannot exist without a value.

If you do not specify a value for a set-valued attribute, the clerk creates the attribute with an empty set.

To delete an attribute value, use the DNS\$\_MODVALUE item code to remove the specified value from an attribute set. If you do not specify the item code, the name service removes the attribute and all its values.

This function code returns the following:

SS\$ \_NORMAL  
DNS\$ \_WRONGATTRIBUTETYPE  
DNS\$ \_INVALIDUPDATE  
DNS\$ \_INVALID\_ENTRYNAME  
DNS\$ \_INVALID\_ATTRIBUTENAME

You must have write or delete access to the directory, object, soft link, or clearinghouse whose attribute is being modified, depending on whether the operation adds or removes the attribute.

You must specify the following input value item codes:

DNS\$ \_ATTRIBUTENAME  
DNS\$ \_ATTRIBUTETYPE  
DNS\$ \_ENTRY  
DNS\$ \_LOOKINGFOR  
DNS\$ \_MODOPERATION

You can specify the following input value item codes:

DNS\$ \_CONF  
DNS\$ \_MODVALUE  
DNS\$ \_WAIT

\$DNS returns the following qualifying status:

DNS\$V \_DNSB \_OUTLINKED

#### **DNS\$ \_NEW\_EPOCH**

This request reconstructs an entire replica set of a directory and synchronizes the copies to recover as much of the original directory state as possible. You can also use the function to change a replica type for configuration management purposes.

This request takes as input the full name of a clearinghouse (DNS\$ \_CLEARINGHOUSE) and directory (DNS\$ \_DIRECTORY). Specify, optionally, the full names of clearinghouses in which to store secondary and read-only replicas (DNS\$ \_SECCHSET and DNS\$ \_READCHSET).

You must have control access to the parent directory and write access to each clearinghouse for which the replica type will be changed from its current value to a new value.

You must specify the following input value item codes:

DNS\$ \_CLEARINGHOUSE  
DNS\$ \_DIRECTORY

You can specify the following input value item codes:

DNS\$ \_READCHSET  
DNS\$ \_SECCHSET

#### **DNS\$ \_PARSE\_FULLNAME\_STRING**

This request takes a full name in string format and converts it to its equivalent in opaque format. If you specify the DNS\$ \_NEXTCHAR\_PTR item code, the clerk examines the name specified in DNS\$ \_FROMSTRINGNAME for invalid characters. The buffer returns the address of the character in the name that immediately follows a valid DECdns name.

## System Service Descriptions

### \$DNS (VAX Only)

This function code returns the following:

SS\$\_NORMAL  
DNS\$\_INVALIDNAME

You must specify the following input value item code:

DNS\$\_FROMSTRINGNAME

You must specify the following output value item code:

DNS\$\_TOFULLNAME

You can specify the following input value item code:

DNS\$\_NEXTCHAR\_PTR

#### **DNS\$\_PARSE\_SIMPLENAME\_STRING**

This request takes a simple name in string format and converts it to its equivalent in opaque format. If you specify the DNS\$\_NEXTCHAR\_PTR item code, the clerk examines the name specified in DNS\$\_FROMSTRINGNAME for invalid characters. The buffer returns the address of the character in that name that immediately follows a valid DECdns name.

This function code returns the following:

SS\$\_NORMAL  
DNS\$\_INVALIDNAME

You must specify the following input value item code:

DNS\$\_FROMSTRINGNAME

You must specify the following output value item code:

DNS\$\_TOSIMPLENAME

You can specify the following input value item code:

DNS\$\_NEXTCHAR\_PTR

#### **DNS\$\_READ\_ATTRIBUTE**

This request returns (in DNS\$\_OUTVALSET) a set whose members are the values of the specified attribute.

To manipulate the values returned by this call, use the DNS\$REMOVE\_FIRST\_SET\_VALUE run-time library routine. The run-time library routine returns the value of a single-valued attribute or the first value from a set-valued attribute. The contents of DNS\$\_OUTVALSET are passed to DNS\$REMOVE\_FIRST\_SET\_VALUE, and the routine returns the value of the attribute.

The attribute values are returned in the order in which they were created. If the call returns DNS\$\_MOREDATA, not all of the set members have been returned. The client application can make further calls, setting DNS\$\_CONTEXTVARTIME to the timestamp of the last attribute in the set returned, until the procedure returns SS\$\_NORMAL.

If the client sets the DNS\$\_MAYBEMORE item code to 1, the name service attempts to make subsequent DNS\$\_READ\_ATTRIBUTE calls for the same value more efficient.

This function code returns the following:

SS\$\_NORMAL  
DNS\$\_MOREDATA

DNS\$\_INVALID\_ENTRYNAME  
DNS\$\_INVALID\_ATTRIBUTENAME

You must have read access to the object whose attribute is to be read.

You must specify the following input value item codes:

DNS\$\_ATTRIBUTENAME  
DNS\$\_ENTRY  
DNS\$\_LOOKINGFOR

You must specify the following output value item code:

DNS\$\_OUTVALSET

You can specify the following input value item codes:

DNS\$\_CONF  
DNS\$\_CONTEXTVARTIME  
DNS\$\_MAYBEMORE  
DNS\$\_WAIT

You can specify the following output value item code:

DNS\$\_CONTEXTVARTIME

\$DNS returns the following qualifying status:

DNS\$\_DNSB\_OUTLINKED

#### **DNS\$\_REMOVE\_LINK**

This request deletes a soft link from the namespace. Only the soft link is deleted. Any DECdns name that is referenced by the soft link remains unaffected by the operation.

You must have delete access to the soft link, or delete or control access to its parent directory.

You must specify the following input value item code:

DNS\$\_LINKNAME

You can specify the following input value item codes:

DNS\$\_CONF  
DNS\$\_WAIT

#### **DNS\$\_REMOVE\_REPLICA**

This request removes the specified replica of a directory.

You must have control access to the replica being removed and write access to the replica's clearinghouse.

You must specify the following input value item codes:

DNS\$\_CLEARINGHOUSE  
DNS\$\_DIRECTORY

You can specify the following input value item codes:

DNS\$\_CONF  
DNS\$\_WAIT

## System Service Descriptions

### \$DNS (VAX Only)

#### **DNS\$\_RESOLVE\_NAME**

This request follows a chain of soft links to its target. The function returns the full name of the target.

Applications that maintain their own databases of opaque DECdns names should use `DNS$_RESOLVE_NAME` any time they receive the qualifying status `DNS$_DNSB_OUTLINKED`. The qualifying status indicates that a soft link was followed to make the request to the DECdns server. After receiving the resolved name, the application should store it, so future references to the name do not incur the overhead of following a soft link.

If the application provides a name that does not contain any soft links, `DNS$_NOTLINKED` status is returned. If the target of any of the chain of soft links followed does not exist, the `DNS$_DANGLINGLINK` status is returned. To obtain the target of any particular soft link, use the `DNS$_READ_ATTRIBUTE` function with `DNS$_LOOKINGFOR` set to `DNS$_SOFTLINK` and request the attribute `DNS$_LINKTARGET`. This can be useful in discovering which link in a chain does not point to an existing target. If the DECdns clerk detects a loop, it returns `DNS$_POSSIBLECYCLE` status.

This function code returns the following:

- SS\$\_NORMAL
- DNS\$\_INVALID\_LINKNAME
- DNS\$\_NOTLINKED
- DNS\$\_POSSIBLECYCLE

You must have read access to each of the soft links in the chain.

You must specify the following input value item code:

- DNS\$\_LINKNAME

You must specify the following output value item code:

- DNS\$\_OUTNAME

You can specify the following input value item codes:

- DNS\$\_CONF
- DNS\$\_WAIT

\$DNS returns the following qualifying status:

- DNS\$\_DNSB\_OUTLINKED

#### **DNS\$\_SIMPLE\_OPAQUE\_TO\_STRING**

This request takes a simple name in opaque format and converts it to its equivalent in string format.

This function code returns the following:

- SS\$\_NORMAL
- DNS\$\_INVALIDNAME

You must specify the following input value item code:

- DNS\$\_FROMSIMPLENAME

You must specify the following output value item code:

- DNS\$\_TOSTRINGNAME

### **DNS\$\_SKULK**

This request attempts to ensure that all replicas of the specified directory have absorbed all updates applied to any replica prior to the time the skulk began. Successful update of the replica set requires all replicas to be available for an extended time.

You must have control access to the directory being skulked.

You must specify the following input value item code:

DNS\$\_DIRECTORY

### **DNS\$\_TEST\_ATTRIBUTE**

This request tests an object for the presence of a particular attribute value. This function returns DNS\$\_TRUE in the \$DNS status block if the specified attribute has one of the following characteristics:

- It is a single-valued attribute and its value matches the specified value.
- It is a set-valued attribute and the attribute contains the specified value as one of its members.

If the attribute is not present or if the specified attribute does not exist, the function returns DNS\$\_FALSE in the \$DNS status block.

This function code returns the following:

DNS\$\_INVALID\_ENTRYNAME  
DNS\$\_INVALID\_ATTRIBUTENAME

You must have test or read access to the directory, object, soft link, or clearinghouse whose attribute is to be tested.

You must specify the following input value item codes:

DNS\$\_ATTRIBUTENAME  
DNS\$\_ENTRY  
DNS\$\_LOOKINGFOR  
DNS\$\_VALUE

You can specify the following input value item codes:

DNS\$\_CONF  
DNS\$\_WAIT

\$DNS returns the following qualifying status:

DNS\$\_DNSB\_OUTLINKED

### **DNS\$\_TEST\_GROUP**

This request tests a group object for a particular member. It returns DNS\$\_TRUE in the \$DNS status block if the specified member is a member of the specified group (or a subgroup thereof), and DNS\$\_FALSE otherwise. If the clerk searches a subgroup and one or more of the subgroups is unavailable, the clerk returns the status encountered in trying to access that group.

The DNS\$\_INOUTDIRECT argument, on input, controls the scope of the search. If you set this item code to 1, the clerk searches only the top-level group. If you set it to 0, the clerk searches all of the subgroups. On output, the clerk returns a 1 in the DNS\$\_DNSB\_INOUTDIRECT qualifying status if the member was found in the top-level group; it returns a 0 if the member was found in a subgroup.

## System Service Descriptions

### \$DNS (VAX Only)

This function code returns the following:

```
SS$_NORMAL
DNS$_NOTAGROUP
DNS$_INVALID_GROUPNAME
DNS$_INVALID_MEMBERNAME
```

You must have test or read access to each of the groups being tested or control access to their respective directories.

You must specify the following input value item codes:

```
DNS$_GROUP
DNS$_MEMBER
```

You can specify the following input value item codes:

```
DNS$_CONF
DNS$_INOUTDIRECT
DNS$_WAIT
```

\$DNS returns the following qualifying status:

```
DNS$_DNSB_INOUTDIRECT
DNS$_DNSB_OUTLINKED
```

## Item Codes

Table SYS1-8 provides a summary of item codes that are valid as an item descriptor in the **itmlst** argument. The table lists the item codes and their data types. Complete descriptions of each item code are provided after the table.

**Table SYS1-8 \$DNS Item Codes and Their Data Types**

Item Code	Data Type
DNS\$_ATTRIBUTENAME	An opaque simple name, which is limited to 31 ISO Latin-1 characters.
DNS\$_ATTRIBUTETYPE	A single byte, indicating whether the attribute is a set (DNSK\$_SET) or a single value (DNSK\$_SINGLE), followed by an opaque simple name.
DNS\$_CLASS	An opaque simple name, limited to 31 ISO Latin-1 characters.
DNS\$_CLASSFILTER	An opaque simple name that can contain a wildcard.
DNS\$_CLEARINGHOUSE	An opaque simple name of a clearinghouse.
DNS\$_CONF	The confidence setting, which is a 1-byte field with the value DNSK\$_LOW, DNSK\$_MEDIUM, or DNSK\$_HIGH.
DNS\$_CONTEXTVARNAME	An opaque simple name.
DNS\$_CONTEXTVARTIME	A creation timestamp (CTS).
DNS\$_DIRECTORY	An opaque full name of a directory.

(continued on next page)



## System Service Descriptions \$DNS (VAX Only)

**Table SYS1-8 (Cont.) \$DNS Item Codes and Their Data Types**

Item Code	Data Type
DNS\$_ENTRY	An opaque full name of a directory, soft link, group, or clearinghouse.
DNS\$_EXPIRETIME	A quadword absolute time representation.
DNS\$_EXTENDTIME	A quadword relative time representation.
DNS\$_FROMFULLNAME	An opaque full name.
DNS\$_FROMSIMPLENAME	An opaque simple name.
DNS\$_FROMSTRINGNAME	A full or simple name consisting of a string of ISO-1 Latin characters. The length of the name is length stored separately in an item list.
DNS\$_GROUP	An opaque full name.
DNS\$_INOUTDIRECT	A 1-byte Boolean field. Valid values are 0 and 1.
DNS\$_LINKNAME	An opaque full name of a soft link.
DNS\$_LOOKINGFOR	A 1-byte field. Valid values are DNS\$_K_OBJECT, DNS\$_K_SOFTLINK, DNS\$_K_CHILDDIRECTORY, DNS\$_K_DIRECTORY, or DNS\$_K_CLEARINGHOUSE.
DNS\$_MAYBEMORE	A 1-byte Boolean field. Valid values are DNS\$_FALSE and DNS\$_TRUE.
DNS\$_MEMBER	A single byte, indicating whether the member is a principal (DNS\$_K_GRPMEM_NOT_GROUP) or another group (DNS\$_K_GRPMEM_IS_GROUP), followed by the opaque full name of the member.
DNS\$_MODOPERATION	A value indicating that an attribute is being added (DNS\$_K_PRESENT) or deleted (DNS\$_K_ABSENT).
DNS\$_MODVALUE	The structure of this value is dependent on the application.
DNS\$_NEXTCHAR_PTR	The address of an invalid character following a valid full or simple name.
DNS\$_OBJECTNAME	An opaque full name.
DNS\$_OUTATTRIBUTESET	DNS\$_K_SET or DNS\$_K_SINGLE in the first byte followed by a single or set of attribute names.
DNS\$_OUTCHILDREN	A set of opaque simple names of the child directories found in the parent directory.
DNS\$_OUTCTS	A timestamp.
DNS\$_OUTNAME	An opaque full name.

(continued on next page)

## System Service Descriptions

### \$DNS (VAX Only)

Table SYS1–8 (Cont.) \$DNS Item Codes and Their Data Types

Item Code	Data Type
DNS\$_OUTOBJECTS	A set of opaque simple names. Optionally, each simple name can be followed by the value of the DNS\$Class attribute.
DNS\$_OUTSOFTLINKS	A set of opaque simple names of the soft links for an object.
DNS\$_OUTVALSET	A set of attribute values.
DNS\$_READCHSET	An opaque full name of a read-only directory.
DNS\$_REPLICATYPE	The type of directory replica. Valid values are secondary replica (DNS\$K_SECONDARY) and read-only replica (DNS\$K_READONLY).
DNS\$_RETURNCLASS	A flag indicating that the value of DNS\$Class is returned in DNS\$_OUTOBJECTS.
DNS\$_SECCHSET	An opaque full name of a secondary directory.
DNS\$_SUPPRESS_NSNAME	A 1-byte value: a value of DNS\$_TRUE suppresses the namespace name, and a value of DNS\$_FALSE returns the namespace name.
DNS\$_TARGETNAME	The opaque full name of an entry in the namespace to which a soft link will point.
DNS\$_TOFULLNAME	The opaque full name of an object. The maximum output of DNS\$PARSE_FULLNAME_STRING is 402 bytes.
DNS\$_TOSIMPLENAME	An opaque simple name. It can be no longer than 257 bytes.
DNS\$_TOSTRINGNAME	A name string of ISO-1 Latin characters. The name length is stored separately in an item list.
DNS\$_VALUE	An attribute value in string format.
DNS\$_VERSION	A 2-byte field: the first byte contains the major version number, the second contains the minor version number.
DNS\$_WAIT	A quadword time representation.
DNS\$_WILDCARD	An opaque simple name containing a wildcard character.

This section describes each item code.

#### DNS\$\_ATTRIBUTENAME

The DNS\$\_ATTRIBUTENAME item code specifies the opaque simple name of an attribute. An attribute name cannot be longer than 31 characters.

#### DNS\$\_ATTRIBUTETYPE

The DNS\$\_ATTRIBUTETYPE item code specifies whether an attribute is set valued (DNS\$K\_SET) or single valued (DNS\$K\_SINGLE).

**DNS\$\_CLASS**

The DNS\$\_CLASS item code specifies the DNS\$Class attribute of an object for the \$DNS function DNS\$\_CREATE\_OBJECT. DNS\$\_CLASS is an opaque simple name.

**DNS\$\_CLASSFILTER**

DNS\$\_CLASSFILTER specifies a filter that limits the scope of an enumeration to those objects belonging to a certain class or group of classes. DNS\$\_CLASSFILTER is used by the \$DNS function DNS\$\_ENUMERATE\_OBJECTS. DNS\$\_CLASSFILTER is an opaque simple name, which can contain a wildcard (either the asterisk or question mark).

DNS\$\_CLASSFILTER is optional. A wildcard simple name using an asterisk (\*) is used by default, meaning that objects of all classes are enumerated.

**DNS\$\_CLEARINGHOUSE**

DNS\$\_CLEARINGHOUSE specifies the clearinghouse in which the directory will be added or removed. DNS\$\_CLEARINGHOUSE is an opaque full name.

**DNS\$\_CONF**

DNS\$\_CONF specifies for \$DNS whether to use the clerk's cache or a DECdns server to complete the request. DNS\$\_CONF is 1 byte long and can take one of the following values:

Confidence Level	Description
DNS\$K_LOW	On read requests, services the DECdns request from the clerk's cache. On create or modify requests, services the request from a master or secondary directory.
DNS\$K_MEDIUM	Bypasses any cached information and services the request directly from a DECdns server.
DNS\$K_HIGH	Services the request from the master directory.

DNS\$\_CONF is optional; if it is not specified, the DECdns clerk assumes a value of DNS\$K\_LOW.

**DNS\$\_CONTEXTVARNAME**

DNS\$\_CONTEXTVARNAME specifies and returns a context for the enumeration functions. On input, specify null to set the initial context. On output, DNS\$\_CONTEXTVARNAME returns the opaque simple name of the last item enumerated.

DNS\$\_CONTEXTVARNAME is optional. If you do not specify or you specify a null value for the context variable item, the clerk returns the results from the beginning of the set. To restart an enumeration where it left off, specify the last value returned in DNS\$\_CONTEXTVARNAME.

**DNS\$\_CONTEXTVARTIME**

DNS\$\_CONTEXTVARTIME specifies and returns a timestamp for the DNS\$\_READ\_ATTRIBUTE function. On input, specify a timestamp to set up the context for reading attributes. On output, DNS\$\_CONTEXTVARNAME returns the timestamp of the last item read.

## System Service Descriptions

### \$DNS (VAX Only)

DNS\$\_CONTEXTVARTIME is optional. If you do not specify or you specify a null value for the context variable item, the clerk returns the results from the beginning of the set. To restart a read operation where it left off, specify the last value returned in DNS\$\_CONTEXTVARTIME.

#### **DNS\$\_DIRECTORY**

DNS\$\_DIRECTORY specifies the directory in which the child directories, soft links, or objects to be enumerated reside. DNS\$\_DIRECTORY is an opaque full name.

#### **DNS\$\_ENTRY**

DNS\$\_ENTRY specifies the opaque full name of an object, soft link, directory, or clearinghouse in the namespace.

#### **DNS\$\_EXPIRETIME**

DNS\$\_EXPIRETIME specifies the absolute time when the soft link will expire. The clerk deletes the soft link at the expiration time. If this item code is a null value, the clerk neither checks nor deletes the link.

#### **DNS\$\_EXTENDTIME**

DNS\$\_EXTENDTIME specifies an extension factor to be added to the absolute time if the soft link still exists. A new expiration time is created by adding the expiration time and the extend time together.

#### **DNS\$\_FROMFULLNAME**

DNS\$\_FROMFULLNAME specifies for the DNS\$\_FULL\_OPAQUE\_TO\_STRING function the opaque full name that is to be converted into string format.

#### **DNS\$\_FROMSIMPLENAME**

DNS\$\_FROMSIMPLENAME specifies for the DNS\$\_SIMPLE\_OPAQUE\_TO\_STRING function the opaque simple name that is to be converted into string format.

#### **DNS\$\_FROMSTRINGNAME**

DNS\$\_FROMSTRINGNAME specifies, in string format, a simple or full name that is to be converted to opaque format for the parse functions DNS\$\_PARSE\_FULLNAME\_STRING and DNS\$\_PARSE\_SIMPLENAME\_STRING.

#### **DNS\$\_GROUP**

DNS\$\_GROUP specifies for the DNS\$\_TEST\_GROUP function the opaque full name of the group that is to be tested. DNS\$\_GROUP must be the name of a group object.

#### **DNS\$\_INOUTDIRECT**

DNS\$\_INOUTDIRECT specifies a value that controls the scope of a test for group membership.

Value	Definition
1	Tests the top-level group specified by the DNS\$_GROUP item (the default)
0	Tests all subgroups of the group named in DNS\$_GROUP

DNS\$\_INOUTDIRECT is a single-byte value.

**DNS\$\_LINKNAME**

DNS\$\_LINKNAME specifies the opaque full name of a soft link.

**DNS\$\_LOOKINGFOR**

DNS\$\_LOOKINGFOR specifies the type of entry in the namespace on which the call is to operate. DNS\$\_LOOKINGFOR can take one of the following values:

- DNS\$\_K\_DIRECTORY
- DNS\$\_K\_OBJECT
- DNS\$\_K\_CHILDDIRECTORY
- DNS\$\_K\_SOFTLINK
- DNS\$\_K\_CLEARINGHOUSE

**DNS\$\_MAYBEMORE**

DNS\$\_MAYBEMORE is used with the DNS\$\_READ\_ATTRIBUTE function to indicate that the results of the read operation are to be cached. This is a single-byte item.

When this item is set to 1, the clerk returns all of the entry's attributes in the return buffer. The clerk caches all of this information to make later lookups of attribute information for the same entry quicker and more efficient.

If you do not specify this item, only the requested information is returned.

**DNS\$\_MEMBER**

DNS\$\_MEMBER specifies for the DNS\$\_TEST\_GROUP function of \$DNS the opaque full name of a member that is to be tested for inclusion within a given group.

**DNS\$\_MODOPERATION**

DNS\$\_MODOPERATION specifies for the DNS\$\_MODIFY\_ATTRIBUTE function the type of operation that is to take place. There are two types of modifications: adding an attribute or deleting an attribute. To add an attribute, specify DNS\$\_K\_PRESENT. To delete an attribute, specify DNS\$\_K\_ABSENT.

**DNS\$\_MODVALUE**

DNS\$\_MODVALUE specifies for the DNS\$\_MODIFY\_ATTRIBUTE function the value that is to be added to or deleted from an attribute. The structure of this value is dependent on the application.

DNS\$\_MODVALUE is an optional argument that affects the overall operation of the DNS\$\_MODIFY\_ATTRIBUTE function. Note that the DNS\$\_MODVALUE item code must be specified to add a single-valued attribute. You can specify a null value for a set-valued attribute. (See the DNS\$\_MODIFY\_ATTRIBUTE item code description for more information.)

**DNS\$\_NEXTCHAR\_PTR**

DNS\$\_NEXTCHAR\_PTR is an optional item code that can be used with the parse functions DNS\$\_PARSE\_FULLNAME\_STRING and DNS\$\_PARSE\_SIMPLENAME\_STRING to return the address of an invalid character that immediately follows a valid DECdns name. This option is most useful when applications are parsing command line strings.

Without this item code, the parse functions return an error if any portion of the name string is invalid.

## System Service Descriptions

### \$DNS (VAX Only)

#### **DNS\$\_OBJECTNAME**

DNS\$\_OBJECTNAME specifies the opaque full name of an object.

#### **DNS\$\_OUTATTRIBUTESET**

DNS\$\_OUTATTRIBUTESET returns a set of enumerated attribute names. This item code is used with the DNS\$\_ENUMERATE\_ATTRIBUTES functions. The item code returns either DNS\$\_K\_SET or DNS\$\_K\_SINGLE along with the set of attribute names.

The names returned in this set can be extracted from the buffer with the DNS\$REMOVE\_FIRST\_SET\_VALUE routine. The resulting values are contained in the \$DNSATTRSPECDEF structure. This 1-byte structure indicates whether an attribute is set-valued or single-valued followed by an opaque simple name.

#### **DNS\$\_OUTCHILDREN**

DNS\$\_OUTCHILDREN returns the set of opaque simple names enumerated by the DNS\$\_ENUMERATE\_CHILDREN function.

You can extract the values resulting from the enumeration using the DNS\$REMOVE\_FIRST\_SET\_VALUE run-time library routine. These values are the opaque simple names of the child directories found in the parent directory.

#### **DNS\$\_OUTCTS**

DNS\$\_OUTCTS returns the creation timestamp (CTS) that the specified entry received when it was created. This item code is optional and can be used by the \$DNS create functions.

#### **DNS\$\_OUTNAME**

DNS\$\_OUTNAME returns the opaque full name of the target pointed to by a soft link. This item code is used with the DNS\$\_RESOLVE\_NAME function.

#### **DNS\$\_OUTOBJECTS**

DNS\$\_OUTOBJECTS returns the set of opaque simple names enumerated by the DNS\$\_ENUMERATE\_OBJECTS function.

Each object name is followed by the object's class if you specify the DNS\$\_RETURNCLASS item code on input. The object's class is the value of the DNS\$Class attribute.

You can extract the values resulting from the enumeration using the DNS\$REMOVE\_FIRST\_SET\_VALUE run-time library routine. The resulting values are the opaque simple names of the objects found in the directory.

#### **DNS\$\_OUTSOFTLINKS**

DNS\$\_OUTSOFTLINKS returns the set of opaque simple names enumerated by the DNS\$\_ENUMERATE\_SOFTLINKS function.

You can extract the values resulting from the enumeration using the DNS\$REMOVE\_FIRST\_SET\_VALUE run-time library routine. The resulting values are the opaque simple names of the soft links found in the directory.

#### **DNS\$\_OUTVALSET**

DNS\$\_OUTVALSET returns for the DNS\$\_READ\_ATTRIBUTE function a set of values for the given attribute.

You can extract the values resulting from the enumeration using the DNS\$REMOVE\_FIRST\_SET\_VALUE run-time library routine. The extracted values are the values of the attribute.

**DNS\$\_READCHSET**

DNS\$\_READCHSET specifies the names of clearinghouses that contain read-only replicas of the directory being reconstructed with DNS\$\_NEW\_EPOCH.

**DNS\$\_REPLICATYPE**

DNS\$\_REPLICATYPE specifies the type of directory replica being added in the specified clearinghouse. You can add a secondary replica (DNS\$\_SECONDARY) or a read-only replica (DNS\$\_READONLY).

**DNS\$\_RETURNCLASS**

DNS\$\_RETURNCLASS specifies that the class of object entries enumerated with the DNS\$\_ENUMERATE\_OBJECTS function should be returned along with the object names in the DNS\$\_OUTOBJECTS item code. The object's class is the value of the DNS\$\_Class attribute.

**DNS\$\_SECCHSET**

DNS\$\_SECCHSET specifies the names of clearinghouses that contain secondary replicas of the directory being reconstructed with DNS\$\_NEW\_EPOCH.

**DNS\$\_SUPPRESS\_NSNAME**

DNS\$\_SUPPRESS\_NSNAME specifies that the leading namespace name should not be returned in the converted full name string. This item code is used by the DNS\$\_FULL\_OPAQUE\_TO\_STRING function. This is an optional single-byte value.

A value of 1 suppresses the leading namespace name in the resulting full name string.

**DNS\$\_TARGETNAME**

DNS\$\_TARGETNAME specifies the name of an existing entry in the namespace to which the soft link will point. This item code is used by the DNS\$\_CREATE\_LINK function.

**DNS\$\_TOFULLNAME**

DNS\$\_TOFULLNAME returns for the DNS\$\_PARSE\_FULLNAME\_STRING function the address of a buffer that contains the resulting opaque full name.

**DNS\$\_TOSIMPLENAME**

DNS\$\_TOSIMPLENAME specifies for the DNS\$\_PARSE\_SIMPLENAME\_STRING function the address of a buffer that will contain the resulting opaque simple name.

**DNS\$\_TOSTRINGNAME**

DNS\$\_TOSTRINGNAME returns the string name resulting from one of the conversion functions: DNS\$\_FULL\_OPAQUE\_TO\_STRING or DNS\$\_SIMPLE\_OPAQUE\_TO\_STRING. DNS\$\_TOSTRINGNAME has the following structure:

[NS\_name:] [.] Namestring [.Namestring]

- *NS\_name*, if present, is a local system representation of the namespace creation timestamp (NSCTS), the unique identifier of the DECdns server. The DECdns clerk supplies a namespace name (*node-name\_NS*) if the value is omitted.
- *Namestring* represents a simple name component. Multiple simple names are separated by periods.

## System Service Descriptions

### \$DNS (VAX Only)

#### **DNS\$\_VALUE**

DNS\$\_VALUE specifies for the DNS\$\_TEST\_ATTRIBUTE function the value that is to be tested. This item contains the address of a buffer holding the value.

#### **DNS\$\_VERSION**

DNS\$\_VERSION specifies the DNS\$ClassVersion attribute for the DNS\$\_CREATE\_OBJECT function. This is a 2-byte structure: the first byte contains the major version number, the second contains the minor version number.

#### **DNS\$\_WAIT**

DNS\$\_WAIT enables the client to specify a timeout value to wait for a call to complete. If the timeout expires, the call returns either DNS\$K\_TIMEOUTNOTDONE or DNS\$K\_TIMEOUTMAYBEDONE, depending on whether the namespace was updated by the incomplete operation.

The parameter is optional; if it is not specified, a default timeout value of 30 seconds is assumed.

#### **DNS\$\_WILDCARD**

DNS\$\_WILDCARD is an optional item code that specifies to the enumeration functions of \$DNS the opaque simple name used to limit the scope of the enumeration. (The simple name does not have to use a wildcard.) Only those simple names that match the wildcard are returned by the enumeration.

Table SYS1-8 provides a summary of the data types for \$DNS item codes. The data types define the encoding of each item list element.

## Description

The \$DNS system service provides a low-level interface between an application (client) and DECdns. The DECdns clerk interface is used to create, delete, modify, and retrieve DECdns names in a namespace.

A single system service call supports the DECdns clerk. It has two main parameters:

- A function code identifying the specific service to perform
- An item list specifying all the parameters for the required function

The use of this item list is similar to that of other system services that use a single item list for both input and output operations.

The \$DNS system service performs DECnet I/O on behalf of the DECdns client. It requires system dynamic memory to construct a database to queue the I/O request and might require additional memory on a device-dependent basis.

In addition to the system services, DECdns provides a set of callable run-time library routines. You can use the clerk run-time library routines to manipulate output from the system service and to write data that can be specified in a system service function code.

For further information, refer to the *OpenVMS Programming Concepts Manual*.

#### **Required Access or Privileges**

None



**Required Quota**

- The buffered I/O byte count (BYTLM) quota for the process
- The quota for buffered I/O limit (BIOLM) or direct I/O limit (DIOLM) for the process
- The AST limit (ASTLM) quota, if an AST service routine is specified, for the process

**Related Services**

\$DNSW

**Condition Values Returned**

SS\$_NORMAL	Normal completion of the request.
SS\$_BADPARAM	Either an item code in the item list is out of range or the item list contains more than the maximum allowable number of items.

**Condition Values Returned in the \$DNS Status Block**

DNS\$_ACCESSDENIED	Caller does not have required access to the entry in question. This error is returned only if the client has some access to the entry; otherwise, the unknown entry status is returned.
DNS\$_BADCLOCK	The clock at the name server has a value outside the permissible range.
DNS\$_BADEPOCH	Copies of directories are not synchronized.
DNS\$_BADITEMBUFFER	Invalid output item buffer detected. (This normally indicates that the buffer has been modified during the call.)
DNS\$_CACHELOCKED	Global client cache locked.
DNS\$_CLEARINGHOUSEDOWN	Clearinghouse is not available.
DNS\$_CLERKBUG	Internal clerk error detected.
DNS\$_CONFLICTINGARGUMENTS	Two or more optional arguments conflict; they cannot be specified in the same function code.
DNS\$_DANGLINGLINK	Soft link points to nonexistent target.
DNS\$_DATACORRUPTION	An error occurred in accessing the data stored at a clearinghouse. The clearinghouse might be corrupted.
DNS\$_ENTRYEXISTS	An entry with the same full name already exists in the namespace.
DNS\$_FALSE	Unsuccessful test operation.
DNS\$_INVALIDARGUMENT	A syntactically incorrect, out of range, or otherwise inappropriate argument was specified in the call.

## System Service Descriptions

### \$DNS (VAX Only)

DNS\$ _INVALID_ATTRIBUTENAME	The name given for function is not a valid DECdns attribute name.
DNS\$ _INVALID_CLASSNAME	The name given for function is not a valid DECdns class name.
DNS\$ _INVALID_CLEARINGHOUSENAME	The name given for function is not a valid DECdns clearinghouse name.
DNS\$ _INVALID_CONTEXTNAME	The name given for function is not a valid DECdns context name.
DNS\$ _INVALID_DIRECTORYNAME	The name given for function is not a valid DECdns directory name.
DNS\$ _INVALID_ENTRYNAME	The name given for function is not a valid DECdns entry name.
DNS\$ _INVALIDFUNCTION	Invalid function specified.
DNS\$ _INVALID_GROUPNAME	The name given for function is not a valid DECdns group name.
DNS\$ _INVALIDITEM	Invalid item code was specified in the item list.
DNS\$ _INVALID_LINKNAME	The name given for function is not a valid DECdns soft link name.
DNS\$ _INVALID_MEMBERNAME	The name given for function is not a valid DECdns member name.
DNS\$ _INVALIDNAME	A name containing invalid characters was specified in the call.
DNS\$ _INVALID_NSNAME	Namespace name given in name string is not a valid DECdns name.
DNS\$ _INVALID_OBJECTNAME	The name given for function is not a valid DECdns object name.
DNS\$ _INVALID_TARGETNAME	The name given for function is not a valid DECdns target name.
DNS\$ _INVALIDUPDATE	An update was attempted to an attribute that cannot be directly modified by the client.
DNS\$ _INVALID_WILDCARDNAME	The name given for function is not a valid DECdns wildcard name.
DNS\$ _LOGICAL_ERROR	Error translating logical name in given string.
DNS\$ _MISSINGITEM	Required item code is missing from the item list.
DNS\$ _MOREDATA	More output data to be returned.
DNS\$ _NAMESERVERBUG	A name server encountered an implementation bug. Please contact your Compaq support representative.
DNS\$ _NOCACHE	Client cache file not initialized.
DNS\$ _NOCOMMUNICATION	No communication was possible with any name server capable of processing the request. Check NCP event 353.5 for the DECnet error.

## System Service Descriptions \$DNS (VAX Only)

DNS\$_NONSNAME	Unknown namespace name specified.
DNS\$_NONSRESOURCES	The call could not be performed due to lack of memory or communication resources at the local node to process the request.
DNS\$_NOTAGROUP	The full name given is not the name of a group.
DNS\$_NOTIMPLEMENTED	This function is defined by the architecture as optional and is not available in this implementation.
DNS\$_NOTLINKED	A soft link is not contained in the name.
DNS\$_NOTNAMESERVER	The node contacted by the clerk does not have a DECdns server running. This can happen when the application supplies the clerk with inaccurate replica information.
DNS\$_NOTSUPPORTED	This version of the architecture does not support the requested function.
DNS\$_POSSIBLECYCLE	Loop detected in soft link or group.
DNS\$_RESOURCEERROR	Failure to obtain system resource.
DNS\$_TIMEOUTMAYBEDONE	The operation did not complete in the time allotted. Modifications might or might not have been made to the namespace.
DNS\$_TIMEOUTNOTDONE	The operation did not complete in the time allotted. No modifications have been performed even if the operation requested them.
DNS\$_TRUE	Successful test operation.
DNS\$_UNKNOWNCLEARINGHOUSE	The clearinghouse does not exist.
DNS\$_UNKNOWNENTRY	Either the requested entry does not exist or the client does not have access to the entry.
DNS\$_UNTRUSTEDCH	A DECdns server is not included in the object's access control set.
DNS\$_WRONGATTRIBUTETYPE	The caller specified an attribute type that did not match the actual type of the attribute.

## System Service Descriptions

### \$DNSW (VAX Only)

---

### \$DNSW (VAX Only)

#### Distributed Name Service Clerk and Wait

On VAX systems, is the client interface to the DIGITAL Distributed Name Service.

The \$DNSW service completes synchronously; that is, it returns to the caller after the operation completes.

For asynchronous completion, use the \$DNS service, which returns to the caller immediately after making a name service call. The return status to the client call indicates whether a request was successfully queued to the name service.

In all other respects, \$DNSW is identical to \$DNS. Refer to the \$DNS description for complete information about the \$DNSW service.

#### Format

```
SYS$DNSW [efn] ,func ,itmlst [,dnsb] [,astadr] [,astprm]
```

## \$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.

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

The flag currently defined is shown in the following table:

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.

#### 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 Values Returned section.

## System Service Descriptions

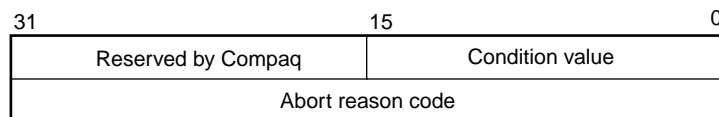
### \$END\_TRANS

- The outcome of the transaction.  
If the service returns SSS\_NORMAL, the outcome of the transaction is commit. If the service returns SSS\_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 SSS\_ABORT.

The \$DDTMMSGDEF macro defines symbolic names for these abort reason codes; those currently defined are shown in the following table:

Symbolic Name	Description
DDTMS_ABORTED	The application aborted the transaction.
DDTMS_COMM_FAIL	A communications link failed.
DDTMS_INTEGRITY	A resource manager integrity constraint check failed.
DDTMS_LOG_FAIL	A write operation to the transaction log failed.
DDTMS_PART_SERIAL	A resource manager serialization check failed.
DDTMS_PART_TIMEOUT	The timeout specified by a resource manager expired.
DDTMS_SEG_FAIL	A process or image terminated.
DDTMS_SERIALIZATION	A DECdtm transaction manager serialization check failed.
DDTMS_SYNC_FAIL	The transaction was not globally synchronized.
DDTMS_TIMEOUT	The timeout specified on \$START_TRANS expired.
DDTMS_UNKNOWN	The reason is unknown.
DDTMS_VETOED	A resource manager was unable to commit the transaction.

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



VM-0456A-AI

#### **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: transaction\_id  
type: octaword (unsigned)  
access: read only  
mechanism: by reference

Identifier 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.

\$END\_TRANS 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 operations are still in progress. If there are any of these operations in progress when \$END\_TRANS is called, an unintended set of operations could be committed, invalidating the data managed by the resource managers participating in the transaction.

---

\$END\_TRANS returns the outcome of the transaction. If it completes successfully, the outcome of the transaction is commit. If it returns the SSS\_ABORT error, the outcome is abort, and the I/O status block contains one reason why the transaction aborted.

\$END\_TRANS must be called from the process that started the transaction. The access mode of the caller must be the same as or more privileged than that specified in the call to \$START\_TRANS that started the transaction.

\$END\_TRANS does not complete either successfully or with the SSS\_ABORT error until all quotas allocated for the transaction by calls on the local node to DECdtm services have been returned.

\$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.

## System Service Descriptions

### **\$END\_TRANS**

- 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.

#### **Required Access or Privileges**

None

#### **Required Quotas**

ASTLM

#### **Related Services**

\$ABORT\_TRANS, \$ABORT\_TRANSW, \$END\_TRANSW, \$START\_TRANS, \$START\_TRANSW

### **Condition Values Returned**

SS\$_NORMAL	If this was returned in R0, the request was successfully queued. If it was returned in the I/O status block, the service completed successfully.
SS\$_SYNCH	The service completed successfully and synchronously (returned only if the DDTM\$M_SYNC flag is set).
SS\$_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 by the caller.
SS\$_BADPARAM	The options flags were invalid.
SS\$_CURTIDCHANGE	The <b>tid</b> 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	Not enough arguments were supplied.
SS\$_INSFMEM	There was insufficient system dynamic memory for the operation.
SS\$_NOCURTID	An attempt was made to end the default transaction (the <b>tid</b> 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\$_NOSUCHTID	A transaction with the specified transaction identifier does not exist.
SS\$_NOTORIGIN	The calling process did not start the transaction.
SS\$_TPDISABLED	The TP_SERVER process was not running on the local node.
SS\$_WRONGACMODE	The access mode of the caller was less privileged than the mode specified in the call to \$START_TRANS.



**System Service Descriptions**  
**\$END\_TRANS**

SS\$\_WRONGSTATE

The calling process had already called either \$ABORT\_TRANS 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. Refer to the descriptions of these other services for additional information about lock management.

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

For additional information about system service completion, see the Synchronize (\$SYNCH) service.

### Format

```
SYS$ENQ  [efn] ,lkmode ,lkspb ,[flags] ,[resnam] ,[parid] ,[astadr] ,[astprm] ,[blkast]
         ,[acmode] ,[rsdm_id] ,[nullarg]
```

### C Prototype

```
int sys$enq (unsigned int efn, unsigned int lkmode, struct _lkspb *lkspb,
            unsigned int flags, void *resnam, unsigned int parid, void
            (*astadr)(__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.

## System Service Descriptions

### \$ENQ

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 <i>share lock</i> .
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 <i>update lock</i> .
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 <i>exclusive lock</i> .

**Table SYS1-9 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
CW	Yes	Yes	Yes	No	No	No
PR	Yes	Yes	No	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

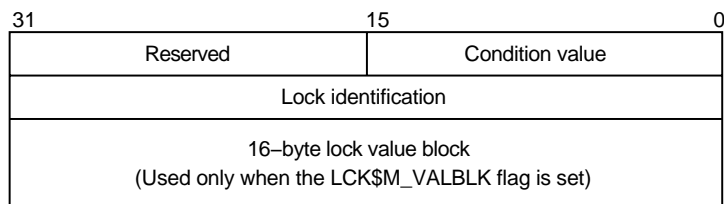
**lksb**

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

Lock status block in which \$ENQ writes the final completion status of the operation. The **lksb** argument is the 32- or 64-bit address (on Alpha systems) or the 32-bit address (on VAX systems) of the 8-byte lock status block.

The lock status block can optionally contain a 16-byte lock value block. The initial value of the lock value block is zero (0). When you specify the LCKSM\_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.

The following diagram shows the format of the lock status block and the optional lock value block:



ZK-1708-GE

The following table defines the status block fields:

## System Service Descriptions

### \$ENQ

Status Block Field	Definition
Condition value	<p>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.</p>
Reserved	<p>A word reserved by Compaq.</p>
Lock identification	<p>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.</p> <p>For a lock conversion on an existing lock, you must supply the lock identification of the existing lock in this field.</p>
Lock value block	<p>A user-defined, 16-byte structure containing information about the resource. This information is interpreted only by the user program.</p> <p>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 <b>flags</b> argument. The copy provided to the process is a copy of the lock value block stored in the lock manager's database.</p> <p>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.</p> <p>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.</p> <p>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 <i>OpenVMS Programming Concepts Manual</i> in the table entitled, "Effect of Lock Conversion on Lock Value Block".</p>

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 SSS_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 SSS_NORMAL in R0; then when the request is granted, \$ENQ sets the event flag and queues an AST if the <b>astadr</b> 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 <b>lksb</b> argument for more information.
LCK\$M_CONVERT	When you specify this flag, \$ENQ performs a lock conversion. In this case, the caller must supply (in the second longword of the lock status block) the lock identification of the lock to be converted.

## System Service Descriptions

### \$ENQ

Flag	Description
LCKSM_NODLCKWT	<p>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.</p> <p>This flag helps to prevent false deadlocks by providing the lock management services with additional information about the process issuing the lock request. When you set this flag, the lock management services do not consider this lock when trying to detect deadlock conditions.</p> <p>A process should specify the LCKSM_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 LCKSM_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.</p> <p>The lock management services make use of the LCKSM_NODLCKWT flag only when the lock specified by the call to \$ENQ is in either the waiting or the conversion queue.</p> <p>Improper use of the LCKSM_NODLCKWT flag can result in the lock management services ignoring genuine deadlocks.</p>



Flag	Description
LCK\$M_NODLCKBLK	<p>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.</p> <p>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.</p> <p>To enable blocking ASTs, the <b>blkast</b> 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.</p> <p>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.</p> <p>Improper use of the LCK\$M_NODLCKBLK flag can result in the lock management services ignoring genuine deadlocks.</p>
LCK\$M_NOQUOTA	<p>This flag is reserved by Compaq. When you set this flag, the calling process is not charged Enqueue Limit (ENQLM) quota for this new lock. The calling process must be running in executive or kernel mode to set this flag. This flag is ignored for lock conversions.</p>
LCK\$M_CVTSYS	<p>This flag is reserved by Compaq. 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.</p>

## System Service Descriptions

### \$ENQ

Flag	Description
LCKSM_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 SSS_UNSUPPORTED will be returned.
LCKSM_QUECVT	<p>This flag is valid only for conversion operations. A conversion request with the LCKSM_QUECVT flag set will be forced to wait behind any already queued conversions.</p> <p>The conversion request is granted immediately, if there are no already queued conversions.</p> <p>The QUECVT behavior is valid only for a subset of all possible conversions. Table SYS1-10 defines the legal set of conversion requests for LCKSM_QUECVT. Illegal conversion requests are failed with SSS_BADPARAM returned.</p>

**Table SYS1-10 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 (Alpha)  
 mechanism: by 32-bit descriptor-fixed-length string descriptor (VAX)

Name of the resource to be locked by this lock. The **resnam** argument is the 32- or 64-bit address (on Alpha systems) or the 32-bit address (on VAX systems) 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 (Alpha)  
mechanism: by 32-bit reference (VAX)

AST service routine to be executed when the lock is either granted or converted. The **astadr** argument is the 32- or 64-bit address (on Alpha systems) or the 32-bit address (on VAX systems) 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 (Alpha)  
mechanism: by 32-bit reference (VAX)

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 (on Alpha systems) or the 32-bit address (on VAX systems) 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.

## System Service Descriptions

### \$ENQ

#### **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
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

Placeholder argument reserved by Compaq.

## 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 SSS\_VALNOTVALID in the lock status block, provided the caller has specified the flag LCKSM\_VALBLK in the **flags** argument.

The SSS\_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, the lock value block becomes 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.
- 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 LCKSM\_INVVALBLK in the **flags** argument, the lock value block maintained in the lock database is marked invalid.

### 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.

## System Service Descriptions

### \$ENQ

#### 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 <b>flags</b> 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 <b>lkmode</b> 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 <b>flags</b> 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 <b>flags</b> argument was specified, and \$ENQ was not able to grant the lock request immediately.
SS\$_PARNOTGRANT	The parent lock specified in the <b>parid</b> 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 <b>flags</b> argument. Note that the lock has been successfully granted despite the return of this warning message. Refer to the Description section for a complete discussion of lock value block invalidation.

## \$ENQW Enqueue Lock Request and Wait

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 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,...);
```



## \$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	ERASK_MEMORY
Disk	ERASK_DISK
Tape	ERASK_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 SSS\_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.

# System Service Descriptions

## \$ERAPAT

### 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 SSS\_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 SSS\_NOTRAN is made. If SSS\_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 SSS\_NOTRAN is returned, at which point the security erase procedure is complete.

```
#include <ssdef.h>
#include <eradef.h>
#include <starlet.h>

/*
** This function takes a pointer to an array of integers and the
** number of elements in the array, and erases the memory used
** by the array. The function returns SSS_NORMAL upon success,
** or the error code from $ERAPAT for any failures.
*/

int ERASE_MEMORY(int *ptr, int items)
{
    int loop,                /* Loop counter for erasing buffer */
        status,             /* Status of system calls */
        pattern,            /* Place to store erase pattern */
        count = 1;         /* Count parameter for $ERAPAT */

    /* Get pattern from $ERAPAT, erase memory, repeat... */
    status = sys$erapat(ERA$K_MEMORY, count++, &pattern);
    while (status == SSS_NORMAL)
    {
        for (loop = 0; loop < items; loop++)
            ptr[loop] = pattern;
        status = sys$erapat(ERA$K_MEMORY, count++, &pattern);
    }

    if (status == SSS_NOTRAN) /* Check for expected status */
        status = SSS_NORMAL; /* Change to SSS_NORMAL if all's well */

    return (status);        /* Return success of failure indication */
}
```

### 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, \$SIDTOASC, \$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 <b>patadr</b> argument cannot be written by the caller.
SS\$_BADPARAM	The <b>type</b> argument or <b>count</b> argument is invalid.

## System Service Descriptions

### \$EXIT

---

### \$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

\$SCANEXH, \$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 pages (on VAX systems) or pagelets (on Alpha systems) to add to the current end of the program or control region. The **pagcnt** argument is a longword value containing this number.

On Alpha 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.

## System Service Descriptions

### \$EXPREG

#### 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.

## System Service Descriptions \$EXPREG

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 Only)

### Expand Virtual Address Space

On Alpha 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
VASC_P0	Program region
VASC_P1	Control region
VASC_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 SSS\_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 for future use by Compaq and should be specified as 0. The condition value SSS\_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 in bytes of the virtual address range created.

## System Service Descriptions

### \$EXPREG\_64 (Alpha Only)

#### 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 SSS\_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 SSS\_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

SSS_NORMAL	The service completed successfully.
SSS_ACCVIO	The <b>return_va_64</b> argument or the <b>return_length_64</b> argument cannot be written by the caller.
SSS_EXPGFLQUOTA	The process exceeded its paging file quota.
SSS_INSFWSL	The process's working set limit is not large enough to accommodate the increased virtual address space.
SSS_IVACMODE	The caller's mode is less privileged than the create mode associated with the region.
SSS_IVREGID	An invalid region ID was specified.

## System Service Descriptions \$EXPREG\_64 (Alpha Only)

SS\$_IVVAFLG	An invalid flag, a reserved flag, or an invalid combination of flags and arguments was specified.
SS\$_LEN_NOTPAGMULT	The <b>length_64</b> 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.

## \$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 (SFAOL) service provides an alternate method for specifying input parameters when calling the SFAO system service.

The formats for both services are shown in the Format section.

On Alpha 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 (Alpha)

mechanism: by 32-bit descriptor-fixed-length string descriptor (VAX)

Control string passed to SFAO that contains the text to be output together with one or more SFAO directives. SFAO 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 (on Alpha systems) or the 32-bit address (on VAX systems) of a character string descriptor pointing to the control string. The formatting of the SFAO directives is described in the Description section.

There is no restriction on the length of the control string or on the number of SFAO 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 SFAO that the next characters are to be interpreted as FAO directives.

When SFAO processes the control string, it writes to the output buffer each character that is not part of an SFAO directive.

If the SFAO directive is valid, SFAO processes it. If the directive requires a parameter, SFAO processes the next consecutive parameter in the specified parameter list. If the SFAO directive is not valid, SFAO terminates and returns a condition value in R0.

Table SYS1–11 lists and describes the \$FAO directives. Table SYS1–12 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 (Alpha)  
mechanism: by 32-bit reference (VAX)

Length in bytes of the fully formatted output string returned by \$FAO. The **outlen** argument is the 32- or 64-bit address (on Alpha systems) or the 32-bit address (on VAX systems) 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 (Alpha)  
mechanism: by 32-bit descriptor–fixed-length string descriptor (VAX)

Output buffer into which \$FAO writes the fully formatted output string. The **outbuf** argument is the 32- or 64-bit address (on Alpha systems) or the 32-bit address (on VAX systems) 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 (Alpha)

## System Service Descriptions

### \$FAO/\$FAOL

mechanism: by 32-bit reference (VAX)

List of \$FAO directive parameters to be passed to \$FAOL. The **prmlst** argument is the 32- or 64-bit address (on Alpha systems) or the 32-bit address (on VAX systems) 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 PUSHHL 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 SSS\_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 !xL (where *x* is O, X, Z, U, or S).
- On Alpha systems, identify quadword data as !xQ for \$FAO and \$FAOL\_64 or !@xQ 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 (32 bits on VAX and 64-bits on Alpha systems), identify the address as !xA for \$FAO and \$FAOL\_64 or !@xA for \$FAOL (where *x* is O, X, Z, U, or S).
- If the size of an address is determined by the hardware architecture (32 bits on VAX, but 64 bits on Alpha), identify the address as !xH for \$FAO and \$FAOL\_64 or !@xH 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.

## System Service Descriptions

### \$FAO/\$FAOL

- If the size of an integer is determined by operating system software (32 bits on both VAX and Alpha systems), identify the integer as !xI for \$FAO and \$FAOL\_64 or !@xI for \$FAOL (where *x* is O, X, Z, U, or S).
- If the size of an integer is determined by the hardware architecture (32 bits on VAX, but 64 bits on Alpha), identify the address as !xJ for \$FAO and \$FAOL\_64 or !@xJ 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 SYS1–11 lists \$FAO directives.

**Table SYS1–11 \$FAO Directives**

Directive	Description
<b>Directives for Character String Substitution</b>	
!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 ASCIID 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.
<b>Directives for Zero-Filled Numeric Conversion</b>	
!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.

(continued on next page)



**Table SYS1–11 (Cont.) \$FAO Directives**

Directive	Description
<b>Directives for Zero-Filled Numeric Conversion</b>	
!OQ	Converts on Alpha 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. <sup>1</sup>
!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. <sup>1</sup>
!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. <sup>2</sup>
!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. <sup>2</sup>
!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.

<sup>1</sup>Determined by the operating system. On VAX and Alpha systems, this is 32 bits.

<sup>2</sup>Determined by the hardware architecture. On VAX systems, this is 32 bits; on Alpha systems, this is 64 bits.

(continued on next page)

## System Service Descriptions

### \$FAO/\$FAOL

Table SYS1–11 (Cont.) \$FAO Directives

Directive	Description
<b>Directives for Zero-Filled Numeric Conversion</b>	
!XQ	Converts on Alpha systems 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. It receives one parameter: the value to be converted. This directive cannot be used from DCL.
!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. <sup>1</sup>
!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. <sup>1</sup>
!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. <sup>2</sup>
!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. <sup>2</sup>
!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.

<sup>1</sup>Determined by the operating system. On VAX and Alpha systems, this is 32 bits.

<sup>2</sup>Determined by the hardware architecture. On VAX systems, this is 32 bits; on Alpha systems, this is 64 bits.

(continued on next page)

**Table SYS1–11 (Cont.) \$FAO Directives**

Directive	Description
<b>Directives for Zero-Filled Numeric Conversion</b>	
!ZQ	Converts on Alpha systems 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 value to be converted. This directive cannot be used from DCL.
!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, a 32-bit value is written to the output buffer. It receives one parameter: the address of the value to be converted. <sup>1</sup>
!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. <sup>1</sup>
!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. <sup>2</sup>
!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 value to be converted. <sup>2</sup>
<b>Directives for Blank-Filled Numeric Conversion</b>	
!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.

<sup>1</sup>Determined by the operating system. On VAX and Alpha systems, this is 32 bits.

<sup>2</sup>Determined by the hardware architecture. On VAX systems, this is 32 bits; on Alpha systems, this is 64 bits.

(continued on next page)

## System Service Descriptions

### \$FAO/\$FAOL

Table SYS1–11 (Cont.) \$FAO Directives

Directive	Description
<b>Directives for Blank-Filled Numeric Conversion</b>	
!UQ	Converts on Alpha systems 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. This directive cannot be used from DCL.
!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. <sup>1</sup>
!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. <sup>1</sup>
!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. <sup>2</sup>
!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. <sup>2</sup>
!SB	Converts a signed byte value to the ASCII representation of the value's decimal equivalent. It requires one parameter: the value to be converted. \$FAO uses only the low-order byte of the longword parameter.
!SW	Converts a signed word value to the ASCII representation of the value's decimal equivalent. It requires one parameter: the value to be converted. \$FAO uses only the low-order word of the longword parameter.
!SL	Converts a signed longword value to the ASCII representation of the value's decimal equivalent. It requires one parameter: the value to be converted.

<sup>1</sup>Determined by the operating system. On VAX and Alpha systems, this is 32 bits.

<sup>2</sup>Determined by the hardware architecture. On VAX systems, this is 32 bits; on Alpha systems, this is 64 bits.

(continued on next page)

**Table SYS1–11 (Cont.) \$FAO Directives**

Directive	Description
<b>Directives for Blank-Filled Numeric Conversion</b>	
!SQ	Converts on Alpha 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. <sup>1</sup>
!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. <sup>1</sup>
!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. <sup>2</sup>
!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. <sup>2</sup>
<b>Directives for Output String Formatting</b>	
!/	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 <i>S</i> if the most recently converted numeric value is not 1. An uppercase <i>S</i> is inserted if the character before the !%S directive is an uppercase character; a lowercase <i>s</i> 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.

<sup>1</sup>Determined by the operating system. On VAX and Alpha systems, this is 32 bits.

<sup>2</sup>Determined by the hardware architecture. On VAX systems, this is 32 bits; on Alpha systems, this is 64 bits.

(continued on next page)

## System Service Descriptions

### \$FAO/\$FAOL

Table SYS1–11 (Cont.) \$FAO Directives

Directive	Description
<b>Directives for Output String Formatting</b>	
!%U	Converts a longword integer UIC to a standard UIC specification in the format [xxx,yyy], where <i>xxx</i> is the group number and <i>yyy</i> is the member number. It takes one parameter: a longword integer. The directive inserts the surrounding brackets ( [ ] ) and comma ( , ).
!%I	Converts a longword to the appropriate alphanumeric identifier. If the longword represents a UIC, surrounding brackets ( [ ] ) and comma ( , ) are added as necessary. If no identifier exists and the longword represents a UIC, the longword is formatted as in !%U. Otherwise it is formatted as in !XL with a preceding !%X added to the formatted result.
!%D	Inserts the system date and time. It takes one parameter: the address of a quadword time value to be converted to ASCII. If you specify 0, the current system date and time is inserted.
!n%C	Inserts a character string when the most recently evaluated argument has the value <i>n</i> . (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 <i>n</i> 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 <i>c</i> in the output string <i>n</i> times.
<b>Directives for Parameter Interpretation</b>	
!–	Causes \$FAO to reuse the most recently used parameter in the list. It takes no parameters.
!+	Causes \$FAO to skip the next parameter in the list. It takes no parameters.

Table SYS1–12 shows the \$FAO output field lengths and their fill characters.

**Table SYS1–12 \$FAO Output Field Lengths and Fill Characters**

<b>Conversion/Substitution Type</b>	<b>Default Length of Output Field</b>	<b>Action When Explicit Output Field Length Is Longer Than Default</b>	<b>Action When Explicit Output Field Length Is Shorter Than Default</b>
<b>Hexadecimal</b>			
Byte	2 (zero-filled)	ASCII result is right-justified and blank-filled to the specified length.	ASCII result is truncated on the left.
Word	4 (zero-filled)		
Longword	8 (zero-filled)		
Quadword	16 (zero-filled)		
<b>Octal</b>			
Byte	3 (zero-filled)	Hexadecimal or octal output is always zero-filled to the default output field length, then blank-filled to specified length.	
Word	6 (zero-filled)		
Longword	11 (zero-filled)		
Quadword	22 (zero-filled)		
<b>Signed or unsigned decimal</b>			
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 (*).
<b>Unsigned zero-filled decimal</b>			
Unsigned zero-filled decimal	As many characters as necessary	ASCII result is right-justified and zero-filled to the specified length.	
<b>ASCII string substitution</b>			
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

## System Service Descriptions

### \$FAO/\$FAOL

#### 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, \$SENDERR, \$SENDJBC, \$SENDJBCW, \$SENDOPR

#### 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 <b>ctrstr</b> , <b>p1</b> through <b>pn</b> , or <b>prmlst</b> arguments cannot be read, or the <b>outlen</b> 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 Compaq 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)!!");
```

## System Service Descriptions

### \$FAO/\$FAOL

```
        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.

## System Service Descriptions \$FAO/\$FAOL

```
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 */
}
```

**SFAOL 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 SFAOL 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 */
}
```

**SFAO 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. SFAO 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)");
```

## System Service Descriptions

### \$FAO/\$FAOL

```
        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 */
}
```

## System Service Descriptions \$FAO/\$FAOL

```
        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 $DESCRIPTOR(fao_desc, "!5*> The time is now: !%D");
    return(sys$faol(&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.

## System Service Descriptions

### \$FAO/\$FAOL

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 */
}
```

## System Service Descriptions \$FAO/\$FAOL

```

        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 ASCII_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
```

In 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))

```

## System Service Descriptions

### \$FAO/\$FAOL

```
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 Compaq Fortran for OpenVMS program used to output the following string:

```
FILE [BOELITZ]TESTING.DAT ABORTED AT ERROR 25
```



---

## \$FAOL\_64 (Alpha Only) Formatted ASCII Output with List Parameter for 64-Bit Virtual Addresses

On Alpha 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.

## System Service Descriptions

### \$FAOL\_64 (Alpha Only)

#### quad\_prmlst\_64

OpenVMS usage: vector\_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 ,value1st ,[fldflags] ,[auxout] ,[retlen]

#### C Prototype

```
int sys$filesan (void *srcstr, void *value1st, 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

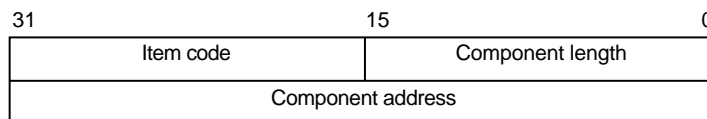
String to be searched for the file specification. The **srcstr** argument is the address of a descriptor pointing to this string.

##### value1st

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:



ZK-5185A-GE

## System Service Descriptions

### \$FILESCAN

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 SSS_ NORMAL condition value.
Item code	A user-supplied, word-length symbolic code that specifies the component desired. The \$FSCNDEF macro defines the item codes.
Component address	A longword in which \$FILESCAN writes the starting address of the component. This address points to a location in the input string itself. 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 SSS_ 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\$_DEVICE	Device name
FSCN\$_DIRECTORY	Directory name
FSCN\$_NAME	File name
FSCN\$_NODE	Node name
FSCN\$_NODE_ACS	Access control string of primary node
FSCN\$_NODE_PRIMARY	Primary (first) node name
FSCN\$_NODE_SECONDARY	Secondary (additional) node information

Symbolic Name	Corresponding Component
FSCN\$V_ROOT	Root directory name string
FSCN\$V_TYPE	File type
FSCN\$V_VERSION	Version number

The **fdflags** 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 **fdflags** 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**

Returns the length and starting address of the device name. The device name includes the single colon (:).

**FSCN\$\_DIRECTORY**

Returns the length and starting address of the directory name. The directory name includes the brackets ([ ]) or angle brackets (< >).

**FSCN\$\_FILESPEC**

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**

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.

## System Service Descriptions

### \$FILESCAN

If there is no name component but there is either a type or version component, the flags argument for FSCN\$V\_NAME is set and the address field contains a nonzero pointer (pointing to the period for the type component); however, the length field does contain zero.

#### **FSCN\$\_NODE**

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**

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**

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**

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**

Returns the length and starting address of the root directory string. The root directory name string includes the brackets ([ ]) or angle brackets (< >).

#### **FSCN\$\_TYPE**

Returns the length and starting address of the file type. The file type includes the preceding period (.).

#### **FSCN\$\_VERSION**

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.

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 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

## System Service Descriptions

### \$FILESCAN

#### 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, \$SENDERR, \$SENDJBC, \$SENDJBCW, \$SENDOPR

#### Condition Values Returned

SS\$_NORMAL	The service completed successfully.
SS\$_ACCVIO	The service could not read the string pointed to by the <b>srcstr</b> argument; or it could not write to an item descriptor in the item list specified by the <b>valuelst</b> argument; or it could not write to the specified auxiliary output buffer; or the <b>retlen</b> argument could not be written.
SS\$_BADPARAM	The item list contains an invalid item code.



---

## \$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:

## System Service Descriptions

### \$FIND\_HELD

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 SSS\_NOSUCHID. When SSS\_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 SSS\_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 <b>id</b> argument cannot be written by the service, or the <b>holder</b> , <b>attrib</b> , or <b>ctxt</b> argument cannot be read by the service.
SS\$IVCHAN	The contents of the <b>ctxt</b> longword are not valid.
SS\$INSMEM	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 *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, refer to the <i>OpenVMS DCL Dictionary</i> .
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

## System Service Descriptions

### \$FIND\_HOLDER

#### Condition Values Returned

SS\$NORMAL	The service completed successfully.
SS\$ACCVIO	The <b>id</b> argument cannot be read by the caller, or the <b>holder</b> , <b>attrib</b> , or <b>ctxt</b> argument cannot be written by the caller.
SS\$IVCHAN	The contents of the <b>ctxt</b> 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 *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 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 (Alpha)

mechanism: by 32-bit reference (VAX)

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 SSS\_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

## System Service Descriptions

### \$FINISH\_RDB

#### Condition Values Returned

SS\$_NORMAL	The service completed successfully.
SS\$_ACCVIO	The <b>ctxt</b> argument cannot be written by the caller.
SS\$_IVCHAN	The contents of the <b>ctxt</b> 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 *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

## System Service Descriptions

### \$FORCEX

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

\$SCANEXH, \$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.)

## \$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

# System Service Descriptions

## \$FORMAT\_ACL

Bit	Name
Bit 6	BIT_6
.	.
.	.
Bit 31	BIT_31

### nullarg

OpenVMS usage: null\_arg  
 type: longword (unsigned)  
 access: read only  
 mechanism: by value

Placelholding argument reserved to Compaq.

## 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:

Flags	Type	Length
Access		
Alarm name		

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

Field	Symbol Name	Description
Flags	ACESW_FLAGS	Word containing Alarm ACE information and ACE type-independent information
Access	ACESL_ACCESS	Longword containing a mask indicating the access modes to be watched
Alarm name	ACEST_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
ACESV_SUCCESS	Indicates that the alarm is raised when access is successful
ACESV_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
ACESV_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.
ACESV_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.
ACESV_NOPROPAGATE	This ACE is not propagated among versions of the same file.
ACESV_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 ACESM rather than ACESV:

Bit	Meaning When Set
ACESV_READ	Read access is monitored.
ACESV_WRITE	Write access is monitored.
ACESV_EXECUTE	Execute access is monitored.

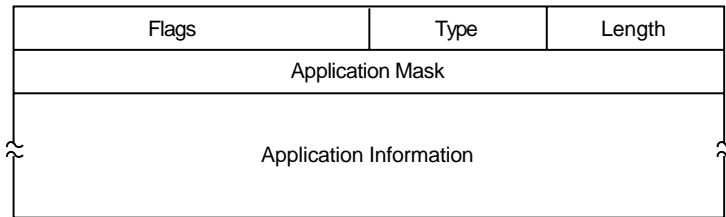
## System Service Descriptions

### \$FORMAT\_ACL

Bit	Meaning When Set
ACESV_DELETE	Delete access is monitored.
ACESV_CONTROL	Modification of the access field is monitored.

#### Application ACE

The Application ACE contains application-dependent information. Its format is as follows:



ZK-1711-GE

The following table describes the ACE fields and lists the symbol name for each:

Field	Symbol Name	Description
Length	ACESB_SIZE	Byte containing the length in bytes of the ACE buffer.
Type	ACESB_TYPE	Byte containing the type value ACESC_INFO
Flags	ACESW_FLAGS	Word containing Application ACE information and ACE type-independent information
Application mask	ACESL_INFO_FLAGS	Longword containing a mask defined and used by the application
Application information	ACEST_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
ACESV_INFO_TYPE	Four-bit field containing a value indicating whether the application is a CSS application (ACESC_CSS) or a customer application (ACESC_CUST).

The following symbols are bit offsets to ACE information that is independent of ACE type:



Bit	Meaning When Set
ACESV_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.
ACESV_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.
ACESV_NOPROPAGATE	This ACE is not propagated among versions of the same file.
ACESV_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:

Flags	Type	Length
Access		
Alarm name		

ZK-1710-GE

The following table describes the ACE fields and lists the symbol name for each:

Field	Symbol Name	Description
Length	ACESB_SIZE	Byte containing the length in bytes of the ACE buffer
Type	ACESB_TYPE	Byte containing the type value ACE\$C_AUDIT
Flags	ACESW_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
ACESV_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.

## System Service Descriptions

### \$FORMAT\_ACL

Bit Position	Meaning When Set
ACESV_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.
ACESV_NOPROPAGATE	This ACE is not propagated among versions of the same file.
ACESV_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 ACESM rather than ACESV.

Bit	Meaning When Set
ACESV_READ	Read access is monitored.
ACESV_WRITE	Write access is monitored.
ACESV_EXECUTE	Execute access is monitored.
ACESV_DELETE	Delete access is monitored.
ACESV_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:

Flags	Type	Length
Access		

ZK-5488A-GE

The following table describes the ACE fields and lists the symbol name for each:

Field	Symbol Name	Description
Length	ACESB_SIZE	Byte containing the length in bytes of the ACE buffer.
Type	ACESB_TYPE	Byte containing the type value ACESC_NEW_OWNER.
Flags	ACESW_FLAGS	Word containing Creator ACE information and ACE type-independent information.
Access	ACESL_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
ACESV_NOPROPAGATE	This ACE is not propagated among versions of the same file.
ACESV_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
ACESV_READ	Read access is granted.
ACESV_WRITE	Write access is granted.
ACESV_EXECUTE	Execute access is granted.
ACESV_DELETE	Delete access is granted.
ACESV_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:

Flags	Type	Length
	Spare	
	System	
	Owner	
	Group	
	World	

ZK-1712-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_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 Service Descriptions

### \$FORMAT\_ACL

Field	Symbol Name	Description
System	ACESL_SYS_PROT	Longword containing a mask indicating the access mode granted to system users. Each bit represents one type of access.
Owner	ACESL_OWN_PROT	Longword containing a mask indicating the access mode granted to the owner. Each bit represents one type of access.
Group	ACESL_GRP_PROT	Longword containing a mask indicating the access mode granted to group users. Each bit represents one type of access.
World	ACESL_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
ACESV_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.
ACESV_NOPROPAGATE	This ACE is not propagated among versions of the same file.
ACESV_PROTECTED	This ACE is not deleted if the entire ACL is deleted; instead, you must delete this ACE explicitly.

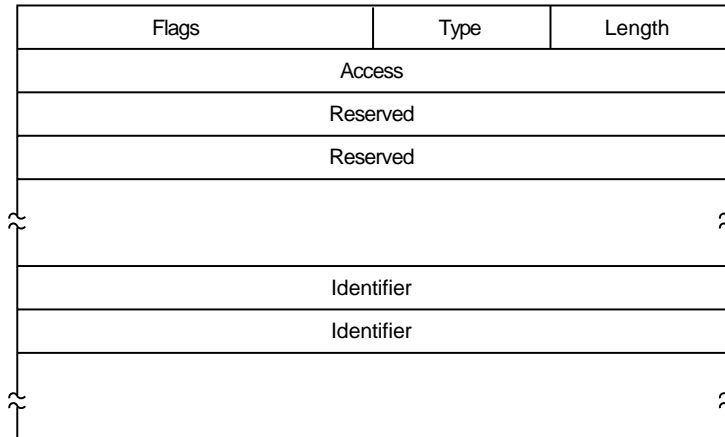
The system interprets the bits within the access mask as shown in the following table. The following symbol values are offsets to bits within the mask indicating the access mode granted in the system, owner, group, and world fields:

Bit Position	Meaning When Bit Is Set
ACESV_READ	Read access is denied.
ACESV_WRITE	Write access is denied.
ACESV_EXECUTE	Execute access is denied.
ACESV_DELETE	Delete access is denied.
ACESV_CONTROL	Delete access is denied.

You can also obtain the symbol values as masks with the appropriate bit set by using the prefix ACE\$M rather than ACE\$V.

#### Identifier ACE

The Identifier ACE controls access to an object based on identifiers. Its format is as follows:



ZK-1713-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_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:

## System Service Descriptions

### \$FORMAT\_ACL

Bit	Meaning When Set
ACESV_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
ACESV_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.
ACESV_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.
ACESV_NOPROPAGATE	This ACE is not propagated among versions of the same file.
ACESV_PROTECTED	This ACE is not deleted if the entire ACL is deleted; instead, you must delete this ACE explicitly.

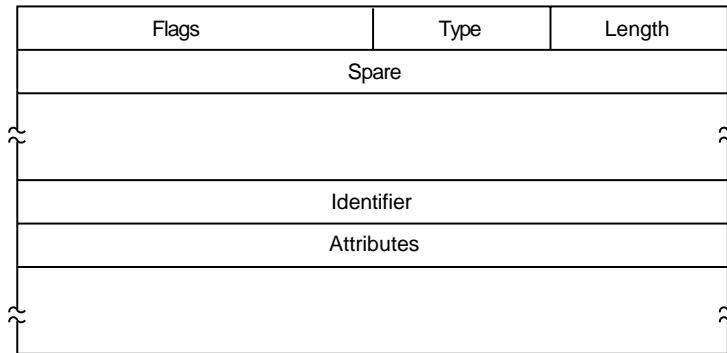
The following symbol values are offsets to bits within the mask indicating the access mode granted in the system, owner, group, and world fields:

Bit Position	Meaning When Set
ACESV_READ	Read access is granted.
ACESV_WRITE	Write access is granted.
ACESV_EXECUTE	Execute access is granted.
ACESV_DELETE	Delete access is granted.
ACESV_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 ACESM rather than ACESV.

#### Subsystem ACE

The Subsystem ACE maintains protected subsystems. Its format is as follows:



ZK-5489A-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_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:

## System Service Descriptions

### \$FORMAT\_ACL

Bit Position	Meaning When Set
ACESV_READ	Read access is granted.
ACESV_WRITE	Write access is granted.
ACESV_EXECUTE	Execute access is granted.
ACESV_DELETE	Delete access is granted.
ACESV_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 ACESV.

#### 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. Refer to the *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.

## System Service Descriptions

### \$FORMAT\_AUDIT

#### **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 (NSASC\_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 <b>fmmtyp</b> of brief and a <b>routin</b> argument.)
	0	Do not build column titles.

**Description**

The Format Security Audit Event Message 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.

## System Service Descriptions

### \$FORMAT\_AUDIT

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\_USER\_CAPABILITY (Alpha Only) Release a Reserved User Capability

On Alpha systems, releases a user capability, indicating to other processes that the resource is now available.

This service accepts 64-bit addresses.

### Format

```
SYS$FREE_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 Compaq and must be 0.

## System Service Descriptions

### \$FREE\_USER\_CAPABILITY (Alpha Only)

#### 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.

## \$GETDVI

### Get Device/Volume Information

Returns information related to the primary and secondary device characteristics of an I/O device.

For synchronous completion, use the Get Device/Volume Information and Wait (\$GETDVIW) service. The \$GETDVIW service is identical to the \$GETDVI service in every way except that \$GETDVIW returns to the caller with the requested information.

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

#### Format

SYS\$GETDVI [efn] [,chan] [,devnam] ,itmlst [,iosb] [,astadr] [,astprm] [,nullarg]

#### 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);
```

#### 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).

##### 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**.

## System Service Descriptions

### \$GETDVI

#### 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.

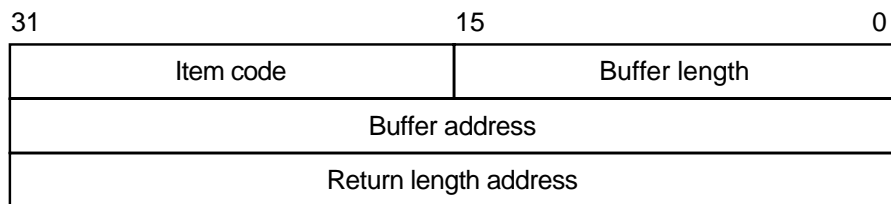
To identify a device to \$GETDVI, you can specify either the **chan** or **devnam** argument, but you should not specify both. If both arguments are specified, the **chan** argument is used.

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:



ZK-5186A-GE



The following table defines the item descriptor fields:

Descriptor Field	Definition
Buffer length	A word containing a user-supplied integer specifying the length (in bytes) of the buffer in which \$GETDVI 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 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 Compaq.

Though this argument is optional, Compaq 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.

## System Service Descriptions

### \$GETDVI

#### **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

Placeholder argument reserved to Compaq.

## Item Codes

#### **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_F11V5	Files-11 Level 5
DVI\$C_ACP_MTA	Magnetic tape
DVI\$C_ACP_NET	Networks
DVI\$C_ACP_REM	Remote I/O

#### **DVI\$\_ALL**

Returns a string that is either TRUE or FALSE to indicate whether the device is allocated or not allocated.

#### **DVI\$\_ALLDEVNAM**

Returns the allocation-class device name, which is a 64-byte string. The allocation-class device name uniquely identifies each device that is currently connected to any node in an OpenVMS Cluster system or to a single-node system.

This item code generates a single unique name for a device even if the device is dual ported.

One use for the allocation-class device name might be in an application wherein processes need to coordinate their access to devices (not volumes) using the 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 allocation-class device name of the device.

Note that the name returned by the `DVIS_DEVLOCKNAM` item code should be used to coordinate access to volumes.

#### **DVIS\_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, refer to *OpenVMS Cluster Systems*.

#### **DVIS\_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 either a VAX 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.

You should not be concerned with which host provides access to the device. When accessing a device, you specify the allocation class of the desired device, not the name of the host that serves it.

If the host serving the primary path fails, the system automatically creates a path to the device through the alternate host.

#### **DVIS\_ALT\_HOST\_NAME**

Returns the name of the host serving the alternate path as a 64-byte zero-filled string.

For more information about hosts, dual-pathed devices, and primary and alternate paths, refer to the description of the `DVIS_ALT_HOST_AVAIL` item code.

## System Service Descriptions

### \$GETDVI

#### 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.

Compaq suggests this item code be used solely for display purposes, and should not be used during an attempt to determine the particular cluster hardware configuration.

#### 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\$_REC	Device is record oriented.
DEV\$_CCL	Device is a carriage control device.
DEV\$_TRM	Device is a terminal.
DEV\$_DIR	Device is directory structured.
DEV\$_SDI	Device is single-directory structured.
DEV\$_SQD	Device is sequential and block oriented.
DEV\$_SPL	Device is being spooled.
DEV\$_OPR	Device is an operator.
DEV\$_RCT	Disk contains Revector Cache Table (RCT). This bit is set for every DAA disk.
DEV\$_NET	Device is a network device.

Symbol	Description
DEV\$V_FOD	Device is files oriented.
DEV\$V_DUA	Device is dual ported.
DEV\$V_SHR	Device is shareable.
DEV\$V_GEN	Device is a generic device.
DEV\$V_AVL	Device is available for use.
DEV\$V_MNT	Device is mounted.
DEV\$V_MBX	Device is a mailbox.
DEV\$V_DMT	Device is marked for dismount.
DEV\$V_ELG	Device has error logging enabled.
DEV\$V_ALL	Device is allocated.
DEV\$V_FOR	Device is mounted foreign.
DEV\$V_SWL	Device is software write locked.
DEV\$V_IDV	Device can provide input.
DEV\$V_ODV	Device can provide output.
DEV\$V_RND	Device allows random access.
DEV\$V_RTM	Device is a real-time device.
DEV\$V_RCK	Device has read-checking enabled.
DEV\$V_WCK	Device has write-checking enabled.

Note that each device characteristic has its own individual \$GETDVI item code with the format DVI\$\_xxxx, where xxxx are the characters following the underscore character in the symbolic name for that device characteristic.

For example, when you specify the item code DVI\$\_REC, \$GETDVI returns a longword value that is interpreted as Boolean. If the value is 0, the device is not record oriented; if the value is 1, it is record oriented. This information is identical to that returned in the DEV\$V\_REC bit of the longword vector specified by the DVI\$\_DEVCHAR item code.

The buffer must specify a longword for all of these device-characteristic item codes.

#### **DVI\$\_DEVCHAR2**

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.

## System Service Descriptions

### \$GETDVI

Symbol	Description
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.
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

### **DVI\$\_DEVDEPEND**

Returns device-dependent characteristics as a 4-byte bit vector. To determine what information is returned for a particular device, refer to the *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. Refer to the *OpenVMS I/O User's Reference Manual* to determine what information is returned for a particular device.

Note that, for terminals only, individual \$GETDVI item codes are provided for most of the informational items returned in the DVI\$\_DEVDEPEND2 longword bit vector. As with DVI\$\_DEVDEPEND, the same characteristic name appears in the item code as appears in the symbolic name defined for each bit in the DVI\$\_DEVDEPEND2 longword, except that in the case of DVI\$\_DEVDEPEND2, the symbolic names for bits are defined by the \$TT2DEF macro.

The list of these terminal-specific item codes follows this list of item codes.

### **DVI\$\_DEVICE\_TYPE\_NAME**

Returns a string identifying the type of device about which information was requested.

### **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.

## System Service Descriptions

### \$GETDVI

#### **DVI\$\_DEVNAM**

Returns the device name as a 64-byte, zero-filled string. The node name is also returned.

#### **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, refer to the *OpenVMS I/O User's Reference Manual*.

#### **DVI\$\_DEVTYPE**

Returns the device type as an unsigned integer longword. The \$TTDEF 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 the preferred device name for user displays as a 256-byte zero-filled string. The DVI\$\_DISPLAY\_DEVNAM item code is not recommended for use with the \$ASSIGN service. Use the DVI\$\_ALLDEVNAM item code to return an allocation class device name that is usable as input to a program.

#### **DVI\$\_ERRCNT**

Returns the device's error count as an unsigned integer longword.

#### **DVI\$\_FC\_NODE\_NAME**

#### **DVI\$\_FC\_PORT\_NAME**

Returns the Fibre Channel Node or Port name for the Fibre Channel Host Bus Adapter as a zero-filled string for each of these values. If a Fibre Channel Disk or Tape is directly attached to the system, then the codes also work for these devices.

The maximum length of this string may change in the future; therefore, Compaq recommends that you pass a 32-byte buffer to this function. A return length address should also be passed with the call and examined when the function completes.

If the return length is equal to the size of the buffer, then call \$GETDVI again with a larger buffer to ensure that the complete name has been read.

#### **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 the node name and device name as a 64-byte, zero-filled string.

The DVI\$\_FULLDEVNAM item code is useful in an OpenVMS Cluster environment because, unlike DVI\$\_DEVNAM, DVI\$\_FULLDEVNAM returns the name of the node on which the device resides.



One use for the DVI\$\_FULLDEVNAM item code might be to retrieve the name of a device in order to have that name displayed on a terminal. However, you should not use this name as a resource name as input to the lock manager; use the name returned by the DVI\$\_DEVLOCKNAM item code for locking volumes and the name returned by DVI\$\_ALLDEVNAM for locking devices.

**DVI\$\_HOST\_AVAIL**

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, refer to the description of the DVI\$\_ALT\_HOST\_AVAIL item code.

**DVI\$\_HOST\_COUNT**

Returns, as a longword integer, the number of hosts that make the device available to other nodes in the OpenVMS Cluster system. 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, refer to 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 64-byte, zero-filled string.

For more information about hosts, dual-pathed devices, and primary and alternate paths, refer to the description of the DVI\$\_ALT\_HOST\_AVAIL item code.

**DVI\$\_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.

Compaq suggests this item code be used solely for display purposes, and should not be used during an attempt to determine the particular cluster hardware configuration.

**DVI\$\_LOCKID**

Returns the lock ID of the lock on a disk. 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

## System Service Descriptions

### \$GETDVI

example, it is served by an HSC controller or MSCP server or if it is a dual-ported MASSBUS disk.

The buffer must specify a longword into which \$GETDVI is to return the 4-byte hexadecimal lock ID.

#### **DVI\$\_LOGVOLNAM**

Returns the logical name of the volume or volume set as a 64-byte string.

#### **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 64-byte, zero-filled 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 64-byte, zero-filled string. This item code is applicable only to disks and tapes.

#### **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\$\_MSCP\_UNIT\_NUMBER**

Returns the internal coded value for MSCP unit numbers as a longword integer. This item code is reserved to Compaq.

#### **DVI\$\_MT3\_DENSITY**

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\$\_MT3\_SUPPORTED**

Returns the density of the device. Valid for tapes only. This code is an unsigned longword integer.

#### **DVI\$\_NEXTDEVNAM**

Returns the device name of the next volume in the volume set as a 64-byte, zero-filled string. The node name is also returned. This item code is applicable only to disks.

#### **DVI\$\_OPCNT**

Returns the operation count for the volume as an unsigned integer longword.

**DVI\$\_OWNUIC**

Returns the user identification code (UIC) of the owner of the device as a standard 4-byte UIC.

**DVI\$\_PID**

Returns the process identification (PID) of the owner of the device as an unsigned integer longword.

**DVI\$\_PREFERRED\_CPU**

The return argument is a 32-bit CPU bit mask with a bit set 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 bit mask input argument to the \$PROCESS\_AFFINITY system service. The argument can be used to assign an application process to the optimal preferred CPU.

**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 64-byte, zero-filled string. This item code is applicable only to disks.

**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\_FAILED\_MEMBER**

Returns a longword, which is interpreted as Boolean. The value 1 indicates that the device is a member that has been removed from the shadow set by the remote server. The DVI\$\_SHDW\_FAILED\_MEMBER item code is for use only with VAX Volume Shadowing (phase I).

## System Service Descriptions

### \$GETDVI

#### DVI\$\_SHDW\_MASTER

Returns a longword, which is interpreted as Boolean. The value 1 indicates that the device is a virtual unit.

#### DVI\$\_SHDW\_MASTER\_NAME

When the specified device is a shadow set member, \$GETDVI returns the device name of the virtual unit 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.

---

#### Note

---

Shadow set members must have a nonzero allocation class to operate in an OpenVMS Cluster system. Refer to *Volume Shadowing for OpenVMS* for more information.

---

Because the shadow set virtual unit name can include up to 64 characters, the buffer length field of the item descriptor should specify 64 (bytes).

#### 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

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\_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 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 DUS10, \$GETDVI returns the string \$255\$DUS10.

---

#### Note

---

Shadow set members must have a nonzero allocation class to operate in an OpenVMS Cluster system. Refer to *Volume Shadowing for OpenVMS* for more information.

---

Because a device name can include up to 64 characters, the buffer length field of the item descriptor should specify 64 (bytes).

**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\$_TIM	Timeout is enabled.
UCB\$_INT	Interrupt is expected.
UCB\$_ERLOGIP	Error log is in progress on unit.
UCB\$_CANCEL	I/O on unit is canceled.
UCB\$_ONLINE	Unit is on line.
UCB\$_POWER	Power failed while unit busy.
UCB\$_TIMOUT	Unit timed out.
UCB\$_INTTYPE	Receiver interrupt.
UCB\$_BSY	Unit is busy.
UCB\$_MOUNTING	Device is being mounted.
UCB\$_DEADMO	Deallocate at dismount.
UCB\$_VALID	Volume is software valid.
UCB\$_UNLOAD	Unload volume at dismount.
UCB\$_TEMPLATE	Template UCB from which other UCBs for this device type are made.
UCB\$_MNTVERIP	Mount verification is in progress.
UCB\$_WRONGVOL	Wrong volume detected during mount verification.
UCB\$_DELETEUCB	Delete this UCB when reference count equals 0.

**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\$\_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 64-byte zero-filled string.

The \$GETDVI service returns the name in the format of the remote system. If the remote system is a LAT terminal server, \$GETDVI returns the name as *server\_name/port\_name*. The names are separated by the slash (/) character. If the remote system is an X.29 terminal, the name is returned as *network.remote\_DTE*.

When writing applications, you should use the string returned by DVI\$\_ACCPORNAM, instead of the physical device name, to identify remote terminals.

## System Service Descriptions

### \$GETDVI

#### 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
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 64-byte zero-filled string.

#### DVI\$\_UNIT

Returns the unit number as an unsigned integer longword.

**DVI\$\_VOLCOUNT**

Returns the number of volumes in the volume set as an unsigned integer longword. This item code is applicable only to disks.

**DVI\$\_VOLNAM**

Returns the volume name as a 12-byte zero-filled 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 value, which is interpreted as Boolean. A value of 1 indicates that the device is part of a volume set; a value of 0 indicates that it is not. This item code is applicable only to disks.

**DVI\$\_VPROT**

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 *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

## System Service Descriptions

### \$GETDVI

DVIS_TT_MODHANGUP	DVIS_TT_BRDCSTMBX
DVIS_TT_DMA	DVIS_TT_ALTYPEAHD
DVIS_TT_ANSICRT	DVIS_TT_REGIS
DVIS_TT_AVO	DVIS_TT_EDIT
DVIS_TT_BLOCK	DVIS_TT_DECCRT
DVIS_TT_EDITING	DVIS_TT_INSERT
DVIS_TT_DIALUP	DVIS_TT_SECURE
DVIS_TT_FALLBACK	DVIS_TT_DISCONNECT
DVIS_TT_PASTHRU	DVIS_TT_SIXEL
DVIS_TT_PRINTER	DVIS_TT_APP_KEYPAD
DVIS_TT_DRCS	DVIS_TT_SYSPWD
DVIS_TT_DECCRT2	
DVIS_TT_DECCRT3	
DVIS_TT_DECCRT4	

#### DVIS\_WWID

Returns the World Wide Identifier (WWID) of Fibre Channel Disk and Tape devices as a zero-filled string of indeterminate length.

The maximum length of this string may change with new devices; therefore, Compaq recommends that a 380-byte buffer be passed to this function.

A return length address should also be passed with the call and examined when the function completes. If the return length is equal to the size of the buffer, then call \$GETDVI again with a larger buffer to ensure that the complete name has been read.

#### DVIS\_yyyy

DVIS\_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 DVIS\_DEVCHAR item code.

In the description of the DVIS\_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 DVIS\_REC item code returns the same information as the DEV\$V\_REC bit in the DVIS\_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.

## 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\_SCAN, \$DISMOU, \$GETDVIW, \$GETMSG, \$GETQUI, \$GETQUIW, \$INIT\_VOL, \$IO\_FASTPATH, \$MOUNT, \$PUTMSG, \$QIO, \$QIOW, \$SENDERR, \$SENDJBC, \$SENDJBCW, \$SENDOPR

**Condition Values Returned**

SS\$_NORMAL	The service completed successfully.
SS\$_ACCVIO	The device name string descriptor, device name string, or <b>itmlst</b> 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.

## System Service Descriptions

### \$GETDVI

SS\$_IVDEVNAM	The device name string contains invalid characters, or neither the <b>devnam</b> nor <b>chan</b> argument was specified.
SS\$_IVLOGNAM	The device name string has a length of 0 or has more than 63 characters.
SS\$_NONLOCAL	The device is on a remote system.
SS\$_NOPRIV	The specified channel is not assigned or was assigned from a more privileged access mode.
SS\$_NOSUCHDEV	The specified device does not exist on the host system.

### Condition Values Returned in the I/O Status Block

Same as those returned in R0.

---

## \$GETDVIW

### Get Device/Volume Information and Wait

Returns information about an I/O device; this information consists of primary and secondary device characteristics.

The \$GETDVIW service completes synchronously; that is, it returns to the caller with the requested information. Compaq 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, refer to the description of \$GETDVI.

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

#### Format

```
SYS$GETDVIW [efn] ,[chan] ,[devnam] ,itmlst [,iosb] [,astadr] [,astprm] [,nullarg]
```

#### 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, unsigned __int64 *nullarg);
```

# System Service Descriptions

## \$GETENV (Alpha Only)

### \$GETENV (Alpha Only)

#### Get Environment

Returns the value(s) of the specified console environment variable(s).

#### Format

`SY$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:

63	31	0	
Buffer length	Item code		STENVDEF\$L_ITEM_CODE/STENVDEF\$L_BUF_LEN
Buffer address			STENVDEF\$Q_BUF_ADDR
Buffer length address			STENVDEF\$Q_RET_ADDR

VM-0460A-AI

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.

## \$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 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).

#### **pidadr**

OpenVMS usage: process\_id  
type: longword (unsigned)  
access: modify  
mechanism: by 32- or 64-bit reference (Alpha)  
mechanism: by 32-bit reference (VAX)

Process identification (PID) of the process about which \$GETJPI is to return information. The **pidadr** argument is the 32-bit address (on VAX systems) or the 32- or 64-bit address (on Alpha systems) 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 SSS\_NOMOREPROC after each call and exiting from the loop when SSS\_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 (Alpha)  
 mechanism: by 32-bit descriptor–fixed-length string descriptor (VAX)

Name of the process about which \$GETJPI is to return information. The **prcnam** argument is the 32-bit address (on VAX systems) or the 32- or 64-bit address (on Alpha systems) 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.

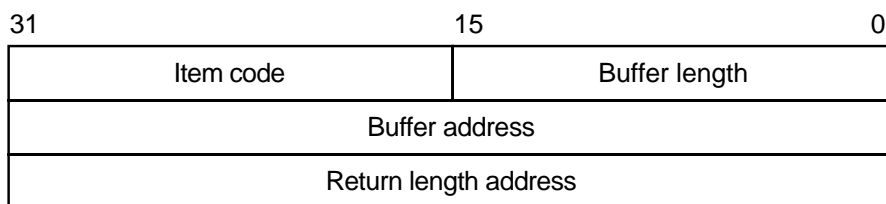
You can use the **prcnam** argument only if the process identified by **prcnam** has the same UIC group number as the calling process. If the process has a different group number, \$GETJPI returns no information. To obtain information about processes in other groups, you must use the **pidadr** argument.

**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 (Alpha)  
 mechanism: by 32-bit reference (VAX)

Item list specifying which information about the process or processes is to be returned. The **itmlst** argument is the 32-bit address (on VAX systems) or the 32- or 64-bit address (on Alpha systems) 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

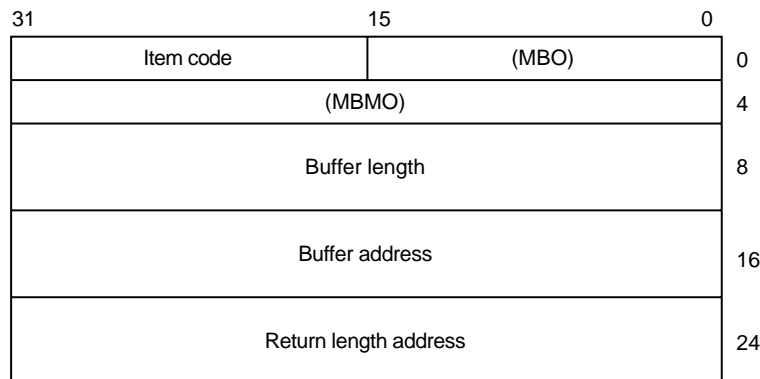
The following table defines the item descriptor fields for 32-bit item list entries:

## System Service Descriptions

### \$GETJPI

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:



ZK-8782A-GE

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.



Descriptor Field	Definition
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 (Alpha)  
 mechanism: by 32-bit reference (VAX)

I/O status block that is to receive the final completion status. The **iosb** argument is the 32-bit address (on VAX systems) or the 32- or 64-bit address (on Alpha systems) 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, Compaq 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.

## System Service Descriptions

### \$GETJPI

#### **astadr**

OpenVMS usage: ast\_procedure  
type: procedure value  
access: call without stack unwinding  
mechanism: by 32- or 64-bit reference (Alpha)  
mechanism: by 32-bit reference (VAX)

AST service routine to be executed when \$GETJPI completes. The **astadr** argument is the 32-bit address (on VAX systems) or the 32- or 64-bit address (on Alpha systems) 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 pages (on VAX systems) or pagelets (on Alpha systems), 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.

Note that this item code is only valid for the current process. If the service is called with a process name or PID other than the current process, it returns the value of zero (0) in the RETLEN output parameter.

#### **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 `SPRVDEF` 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\$\_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\$\_CLASS\_NAME**

Returns the name of the scheduling class (as a character string) that this process belongs to. 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\$\_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.

## System Service Descriptions

### \$GETJPI

#### **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 systems, returns the current explicit affinity mask for the associated kernel thread.

#### **JPI\$\_CURRENT\_USERCAP\_MASK**

On Alpha systems, returns the current user capability mask for the associated kernel thread.

#### **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 pages (on VAX systems) or pagelets (on Alpha systems).

#### **JPI\$\_DFWSCNT**

Returns, in pages (on VAX systems) or pagelets (on Alpha systems), 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\$\_FAST\_VP\_SWITCH**

Returns an unsigned longword containing the number of times this process has issued a vector instruction that resulted in an inactive vector processor being enabled without the expense of a vector context switch. In other words, this count reflects those instances where the process has reenabled a vector processor on which the process's vector context has remained intact.

**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\$\_FREPOVA**

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\$\_FREPTCNT**

Returns the number of pages (on VAX systems) or pagelets (on Alpha systems) that the process has available for virtual memory expansion.

On VAX systems, the value returned is a longword integer. On Alpha 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

## System Service Descriptions

### \$GETJPI

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 SSS\_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\$_NO_TARGET_INSWAP	<p>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:</p> <ul style="list-style-type: none"><li>• Any data stored in the virtual address space of the process is not accessible.</li><li>• Any data stored in the process header (PHD) might not be accessible.</li><li>• Any data stored in resident data structures, such as the process control block (PCB) or the job information block (JIB), is accessible.</li></ul> <p>You must examine the return length of an item to verify that the item was retrieved.</p>

Flag	Description
JPI\$M_NO_TARGET_AST	<p>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:</p> <ul style="list-style-type: none"> <li>• Any data stored in the virtual address space of the process is not accessible.</li> <li>• 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.</li> </ul> <p>You must examine the return length of an item to verify that the item was retrieved.</p> <p>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.</p>
JPI\$M_IGNORE_TARGET_STATUS	<p>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.</p>
JPI\$M_THREAD	<p>Sets the wildcard mode to return information on all of the process's kernel threads beginning with the initial kernel thread.</p>

**JPI\$\_GPGCNT**

Returns, in pages (on VAX systems) or pagelets (on Alpha systems), 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 resource affinity domain (RAD).

Note: OpenVMS support for RADs is available only on the new AlphaServer GS series systems. For more information about using RADs, refer to the *OpenVMS Alpha Partitioning and Galaxy Guide*.

**JPI\$\_IMAGECOUNT**

Returns, as a longword integer value, the number of images that have been run down for the process.

**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 SYS1-13. 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

## System Service Descriptions

### \$GETJPI

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 SYS1–13 Attributes of an Identifier**

Symbolic Name	Description
KGB\$M_DYNAMIC	Identifier can be enabled or disabled.
†KGB\$M_NOACCESS	Rights of the identifier are null and void.
KGB\$M_RESOURCE	Resources can be charged to the identifier.
†KGB\$M_SUBSYSTEM	Identifier can be used to create protected subsystems.

†VAX specific

#### 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

Returns the PID of the initial thread for the target process. The PID is a longword hexadecimal value.

#### JPI\$\_JOBPRCNT

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

Returns the current count of kernel threads for the target process, which is a longword integer value.



**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.
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.

## System Service Descriptions

### \$GETJPI

#### 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

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 (on VAX systems) or pagelets (on Alpha systems).

**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**

Set by \$SET\_PROCESS\_PROPERTIES, and can be either PARSE\_STYLE\$C\_TRADITIONAL or PARSE\_STYLE\$C\_EXTENDED. The return length is one byte.

**JPI\$ PARSE\_STYLE\_PERM**

Set by \$SET\_PROCESS\_PROPERTIES, and can be either PARSE\_STYLE\$C\_TRADITIONAL or PARSE\_STYLE\$C\_EXTENDED. The return length is one byte.

**JPI\$ PERMANENT\_AFFINITY\_MASK**

On Alpha systems, returns the permanent explicit affinity mask for the associated kernel thread.

**JPI\$ PERMANENT\_USERCAP\_MASK**

On Alpha systems, returns the permanent user capability mask for the associated kernel thread.

**JPI\$ PGFLQUOTA**

Returns the paging file quota (maximum virtual page count) of the process, which is a longword integer value, measured in pages (on VAX systems) or pagelets (on Alpha systems).

**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 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 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 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).

## System Service Descriptions

### \$GETJPI

#### **JPI\$\_PPGCNT**

Returns the number of pages (on VAX systems) or pagelets (on Alpha systems) 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 SYS1-13. 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 SYS1-13. 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\_DFMBBC**

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\_POLICY**

On Alpha 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\$\_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.

## System Service Descriptions

### \$GETJPI

#### JPI\$\_SLOW\_VP\_SWITCH

Returns an unsigned longword containing the number of times this process has issued a vector instruction that resulted in an inactive vector processor being enabled with a full vector context switch. This vector context switch involves the saving of the vector context of the process that last used the vector processor and the restoration of the vector context of the current process.

#### 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. The SSTATEDEF macro defines the following symbols, which identify the various possible states:

State	Description
SCH\$C_CEF	Common event flag wait
SCH\$C_COM	Computable
SCH\$C_COMO	Computable, out of balance set
SCH\$C_CUR	Current process
SCH\$C_COLPG	Collided page wait
SCH\$C_FPG	Free page wait
SCH\$C_HIB	Hibernate wait
SCH\$C_HIBO	Hibernate wait, out of balance set
SCH\$C_LEF	Local event flag wait
SCH\$C_LEFO	Local event flag wait, out of balance set
SCH\$C_MWAIT	Mutex and miscellaneous resource wait
SCH\$C_PFW	Page fault wait
SCH\$C_SUSP	Suspended
SCH\$C_SUSPO	Suspended, out of balance set

#### JPI\$\_STS

Returns the first longword of the process status flags, which are contained in a longword bit vector. The \$PCBDEF macro defines 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

Symbol	Description
PCBSV_NETWRK	Process is a network connect object
PCBSV_NOACNT	No accounting for process
PCBSV_NODELET	No delete
PCBSV_PHDRES	Process header resident
PCBSV_PREEMPTED	Kernel mode suspend has overridden supervisor mode suspend
PCBSV_PSWAPM	Process swap mode (1=noswap)
PCBSV_PWRAST	Power fail AST
PCBSV_RECOVER	Process can recover locks
PCBSV_RES	Resident, in balance set
PCBSV_RESPEN	Resume pending, skip suspend
PCBSV_SECAUDIT	Mandatory security auditing
PCBSV_SOFTSUSP	Process is in supervisor mode suspend
PCBSV_SSFEXC	System service exception enable (kernel)
PCBSV_SSFEXCE	System service exception enable (exec)
PCBSV_SSFEXCS	System service exception enable (super)
PCBSV_SSFEXCU	System service exception enable (user)
PCBSV_SSRWAIT	System service resource wait disable
PCBSV_SUSPEN	Suspend pending
PCBSV_WAKEPEN	Wake pending, skip hibernate
PCBSV_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. The \$PCBDEF macro defines the following symbol for these flags:

Symbol	Description
PCBSV_NOUNSHELVE	Process does not automatically unshelve files.

**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, shown in Table SYS1-13. 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.

## System Service Descriptions

### \$GETJPI

#### **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 8 characters, the buffer length field in the item descriptor should specify at least 8 bytes. Trailing zeros are written to the output buffer if necessary.

#### **JPI\$\_THREAD\_INDEX**

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.) 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; otherwise, it 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 pages for VAX or pagelets for Alpha—of the process.

On VAX systems, the value returned is a longword integer. On Alpha 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 number of volumes that the process currently has mounted, which is a longword integer value.

**JPI\$\_VP\_CONSUMER**

Returns a byte, the low-order bit of which, when set, indicates that the process is a vector consumer.

**JPI\$\_VP\_CPUTIM**

Returns an unsigned longword that contains the total amount of time the process has accumulated as a vector consumer.

**JPI\$\_WSAUTH**

Returns the maximum authorized working set size, in pages (on VAX systems) or pagelets (on Alpha systems), of the process. This is a longword integer value.

**JPI\$\_WSAUTHEXT**

Returns, in pages (on VAX systems) or pagelets (on Alpha systems), the maximum authorized working set extent of the process as a longword integer value.

**JPI\$\_WSEXTENT**

Returns, in pages (on VAX systems) or pagelets (on Alpha systems), the current working set extent of the process as a longword integer value.

**JPI\$\_WSPEAK**

Returns, in pages (on VAX systems) or pagelets (on Alpha systems), the peak working set size of the process as a longword integer value.

**JPI\$\_WSQUOTA**

Returns, in pages (on VAX systems) or pagelets (on Alpha systems), the working set size quota of the process as a longword integer value.

**JPI\$\_WSSIZE**

Returns, in pages (on VAX systems) or pagelets (on Alpha systems), the current working set size 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

## System Service Descriptions

### \$GETJPI

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.

#### 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\$_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.

### Example

```
// COPYRIGHT (c) 1999 BY
// DIGITAL EQUIPMENT CORPORATION, NASHUA, NEW HAMPSHIRE.
// ALL RIGHTS RESERVED.
//
// THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
// ONLY IN ACCORDANCE OF THE TERMS OF SUCH LICENSE AND WITH THE
// INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
// COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
// OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
// TRANSFERRED.
//
// THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
// AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
// CORPORATION.
//
// DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
// SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
//
// FACILITY:
//
//     OpenVMS Documentation
//
// Compile and link instructions:
//
//     $ cc/decc getjpi_demo
//     $ link getjpi_demo
//     $ run getjpi_demo
//
//
#pragma module getjpi_demo "V1.0-000"
// The various external definitions and prototypes...
#include <descrip.h>           // static and dynamic string descriptor defs
#include <efndef.h>           // for the No Event Flag Event Flag
#include <jpidef.h>           // sys$getjpi item codes
#include <lib$routines.h>     // run-time library function prototypes
#include <pscandef.h>        // sys$process_scan item codes
#include <ssdef.h>           // System service message codes
#include <starlet.h>         // Function prototypes for system services
#include <stdio.h>           // C standard I/O functions
#include <stsdef.h>          // condition value definitions
// Constants used in this module
```

## System Service Descriptions

### \$GETJPI

```
#define      MAX_ITEMLIST_IL3_ENTRIES  10
#define      PROCESSNAME_LENGTH        16
#define      USERNAME_LENGTH           12
#define      IMAGE_LENGTH               255
#define      TRIMAGE_LENGTH            38

main()
{
    char ImageBuf[IMAGE_LENGTH+1];
    char ProcessBuf[PROCESSNAME_LENGTH+1];
    char UsernameBuf[USERNAME_LENGTH+1];
    int Count = 0;
    int ProcessId;
    int RetStat;
    int i;
    unsigned long int JpiCtx;
    unsigned short int TrimageLength = TRIMAGE_LENGTH;
    unsigned short int ImageLength;
    unsigned short int ProcessLength;
    unsigned short int UsernameLength;
    unsigned short int iosb[4];
    struct dsc$dscdescriptor TrimageDsc = { 0, DSC$K_DTYPE_T, DSC$K_CLASS_D, NULL };
    struct dsc$dscdescriptor ImageDsc = { 0, DSC$K_DTYPE_T, DSC$K_CLASS_S, NULL };
    struct itemlist3
    {
        unsigned short int il_siz;
        unsigned short int il_cod;
        void *il_buf;
        unsigned short int *il_rla;
    } il3[MAX_ITEMLIST_IL3_ENTRIES];
    char *CtlFmtT = "%-8.8s %-12s %-16s %-*.*s\n";
    char *CtlFmtD = "%08X %-12s %-16s %-*.*s\n";

    // Set up the static string descriptor...

    ImageDsc.dsc$a_pointer = ImageBuf;

    // Set up to scan only the interactive processes...

    i = 0;
    il3[i].il_siz = 0;
    il3[i].il_cod = PSCAN$MODE;
    il3[i].il_buf = (void *) JPI$K_INTERACTIVE;
    il3[i+1].il_rla = NULL;
    il3[i].il_siz = 0;
    il3[i].il_cod = 0;
    il3[i].il_buf = NULL;
    il3[i+1].il_rla = NULL;

    // the sys$process_scan service sets up the sys$getjpi context...

    RetStat = sys$process_scan( &JpiCtx, il3 );
    if (!$VMS_STATUS_SUCCESS( RetStat ))
        return RetStat;
}
```

## System Service Descriptions \$GETJPI

```
i = 0;
il3[i].il_siz   = USERNAME_LENGTH;
il3[i].il_cod   = JPI$_USERNAME;
il3[i].il_buf   = UsernameBuf;
il3[i++].il_rla = &UsernameLength;
il3[i].il_siz   = PROCESSNAME_LENGTH;
il3[i].il_cod   = JPI$_PRCNAM;
il3[i].il_buf   = ProcessBuf;
il3[i++].il_rla = &ProcessLength;
il3[i].il_siz   = IMAGE_LENGTH;
il3[i].il_cod   = JPI$_IMAGNAME;
il3[i].il_buf   = ImageBuf;
il3[i++].il_rla = &ImageLength;
il3[i].il_siz   = sizeof( int );
il3[i].il_cod   = JPI$_PID;
il3[i].il_buf   = &ProcessId;
il3[i++].il_rla = NULL;
il3[i].il_siz   = 0;
il3[i].il_cod   = 0;
il3[i].il_buf   = NULL;
il3[i++].il_rla = NULL;

// Display the header...
printf( CtlFmtT, "PID","Username","Process name", 6, 6, "Image");

// Loop through the available processes...
while ( TRUE )
{
    // Find the next process...

    RetStat = sys$getjpiw( EFN$C_ENF, &JpiCtx, NULL,
        il3, iosb, NULL, 0);
    if ((RetStat == SS$Nopriv) || (RetStat == SS$Suspended))
        continue;
    if ( RetStat == SS$Nomoreproc )
        break;
    if (!$VMS_STATUS_SUCCESS( RetStat ))
        return RetStat;

    Count++;

    // Zero-terminate the username, process and image name strings...
    UsernameBuf[UsernameLength] = '\0';
    ProcessBuf[ProcessLength] = '\0';
    ImageBuf[ImageLength] = '\0';

    // Trim the image filename to fit...
    ImageDsc.dsc$w_length = ImageLength;
    RetStat = lib$trim_filespec( &ImageDsc, &TrimageDsc, &TrimageLength );
    if (!$VMS_STATUS_SUCCESS( RetStat ))
        return RetStat;

    printf( CtlFmtD, ProcessId, UsernameBuf, ProcessBuf,
        TrimageDsc.dsc$w_length, TrimageDsc.dsc$w_length,
        TrimageDsc.dsc$a_pointer );
}

printf("Interactive process count: %d\n", Count );

// Free up the dynamic descriptor allocated by lib$trim_filespec
RetStat = lib$sfreel_dd( &TrimageDsc );
if (!$VMS_STATUS_SUCCESS( RetStat ))
    return RetStat;

return SS$NORMAL;
}
```

## **System Service Descriptions**

### **\$GETJPI**

This is a simple program that demonstrates calling various OpenVMS system services and run-time library routines. This example uses the available process privileges to display as many interactive processes as it can—when the WORLD privilege is available, all of the interactive processes will be displayed.

This example was built with OpenVMS (VAX and Alpha) V7.2, using the Compaq C V6.0 compiler.

## \$GETJPIW Get Job/Process Information and Wait

Returns information about one or more processes on the system.

The \$GETJPIW service completes synchronously; that is, it returns to the caller with the requested information. Compaq 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, refer to the description of \$GETJPI in this manual.

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

### 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);
```

## \$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.

#### **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.

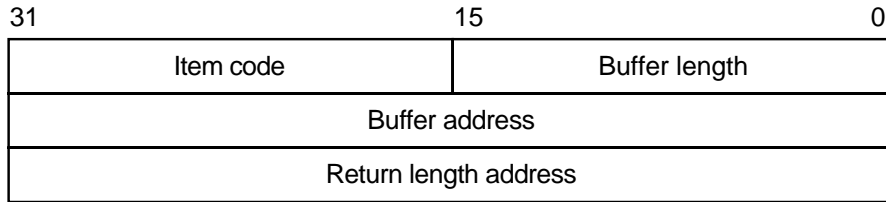
#### **itmlst**

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:



ZK-5186A-GE

The following table defines the item descriptor fields:

Descriptor Field	Definition
Buffer length	A word containing a user-supplied integer specifying the length (in bytes) of the buffer in which \$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 SSS_NORMAL.
Item code	A word containing a user-specified symbolic code specifying the item of information that \$GETLKI is to return. The \$LKIDF 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:

## System Service Descriptions

### \$GETLKI

Descriptor Field	Definition
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 LKIS_BLOCKEDBY, LKIS_BLOCKING, or LKIS_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 SSS_NORMAL condition value in R0. Therefore, to locate any faulty item descriptor, you need to check the state of bit 31 in the longword specified by the return length field of each item descriptor.

#### **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, Compaq 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

Placeholder argument reserved by Compaq.

**Item Codes**

**LKI\$\_BLOCKEDBY**

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\$_MSTLKID	Lock ID of the blocking lock on the system maintaining the resource (4 bytes)
LKI\$_PID	Process ID (PID) of the process that took out the blocked lock (4 bytes)
LKI\$_MSTCSID	OpenVMS Cluster system identifier (CSID) of the node maintaining the resource that is locked by the blocked lock (4 bytes)
LKI\$_RQMODE	Lock mode requested for the blocked lock; this lock mode was specified by the <b>lkmode</b> argument in the call to \$ENQ (1 byte)
LKI\$_GRMODE	Lock mode granted to the blocked lock; this lock mode is written to the lock value block (1 byte)

## System Service Descriptions

### \$GETLKI

Symbolic Name	Description
LKISB_QUEUE	Name of the queue on which the blocked lock currently resides (1 byte)
LKISL_LKID	Lock ID of the lock on the system where the lock was requested (4 bytes)
LKISL_CSID	OpenVMS Cluster system identifier (CSID) of the system where the lock was requested (4 bytes)

The values that \$GETLKI can write into the LKISB\_RQMODE, LKISB\_GRMODE, and LKISB\_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.

#### LKIS\_BLOCKING

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
LKISL_MSTLKID	Lock ID of the blocked lock on the system maintaining the resource (4 bytes)
LKISL_PID	Process ID (PID) of the process that took out the blocking lock (4 bytes)
LKISL_MSTCSID	OpenVMS Cluster system identifier (CSID) of the node maintaining the resource that is locked by the blocking lock (4 bytes)
LKISB_RQMODE	Lock mode requested for the blocking lock; this lock mode was specified by the <b>lkmode</b> argument in the call to SENQ (1 byte)
LKISB_GRMODE	Lock mode granted to the blocking lock; this lock mode is written to the lock value block (1 byte)
LKISB_QUEUE	Name of the queue on which the blocking lock currently resides (1 byte)
LKISL_LKID	Lock ID of the lock on the system where the lock was requested (4 bytes)
LKISL_CSID	OpenVMS Cluster system identifier (CSID) of the system where the lock was requested (4 bytes)

The values that \$GETLKI can write into the LKISB\_RQMODE, LKISB\_GRMODE, and LKISB\_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.

#### LKIS\_CSID

Returns the Cluster System ID (CSID) of the system where the process owning the lock resides. LKIS\_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, LKIS\_CSID returns 0.

The buffer length field in the item descriptor should specify 4 (bytes).

#### LKIS\_CVTCOUNT

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).

#### LKIS\_GRANTCOUNT

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 LKIS\_CVTCOUNT and LKIS\_GRANTCOUNT.

The buffer length field in the item descriptor should specify 4 (bytes).

#### LKIS\_LCKREFCNT

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).

#### LKIS\_LKID

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 LKIS\_CSID item code.

The buffer length field in the item descriptor should specify 4 (bytes).

#### LKIS\_LOCKID

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.

## System Service Descriptions

### \$GETLKI

The lock ID is a longword value, so the buffer length field in the item descriptor should specify 4 (bytes).

#### LKI\$\_LOCKS

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 <b>lkmode</b> argument in the call to SENQ (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 LKI\$B\_RQMODE, LKI\$B\_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

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. LKIS\_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).

**LKIS\_MSTLKID**

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 LKIS\_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 LKIS\_LKID.

The buffer length field in the item descriptor should specify 4 (bytes).

**LKIS\_NAMESPACE**

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
LKISW_GROUP	In this field (bits 0 to 15), \$GETLKI writes the UIC group number of the process that took out the first lock on the resource, thereby creating the resource name. This process issued a call to \$ENQ or \$ENQW specifying the name of the resource in the <b>resnam</b> 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.
LKISB_RMOD	In this field (bits 16 to 23), \$GETLKI writes the access mode associated with the first lock taken out on the resource.

## System Service Descriptions

### \$GETLKI

Symbolic Name	Description
LKISB_STATUS	This field (bits 24 to 30) is not used. \$GETLKI sets it to 0.
LKISV_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

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

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

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

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

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
LKISB_STATE_RQMODE	Lock mode requested
LKISB_STATE_GRMODE	Lock mode granted
LKISB_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**

Returns the lock value block of the locked resource. This lock value block is the master copy that the lock manager maintains for the resource, not the process-private copy.

Because the lock value block is 16 bytes, the buffer length field in the item descriptor should specify 16.

#### **LKI\$\_WAITCOUNT**

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.

## System Service Descriptions

### \$GETLKI

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 LKIS\_REMLKID item code is specified in the call.

Item codes LKIS\_BLOCKEDBY, LKIS\_BLOCKING, LKIS\_LOCKS, and LKIS\_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
LKISC_GRANTED	Granted queue, holding locks that have been granted
LKISC_CONVERT	Converting queue, holding locks that are currently being converted to another lock mode
LKISC_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 <b>lkidadr</b> 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 <b>lkidadr</b> 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 <b>lkidadr</b> argument, and \$GETLKI has exhausted the locks about which it can return information to the caller; or no <b>lkidadr</b> 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.

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, refer to 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. Refer to the descriptions of these other services for additional information about lock management.

#### **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);
```

## \$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 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 (Alpha)  
 mechanism: by 32-bit reference (VAX)

Length of the message string returned by \$GETMSG. The **msglen** argument is the 32- or 64-bit address (on Alpha systems) or the 32-bit address (on VAX systems) 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 (Alpha)  
 mechanism: by 32-bit descriptor-fixed-length string descriptor (VAX)

Buffer to receive the message string. The **bufadr** argument is the 32- or 64-bit address (on Alpha systems) or the 32-bit address (on VAX systems) 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.

## System Service Descriptions

### \$GETMSG

#### 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 (Alpha)  
mechanism: by 32-bit reference (VAX)

Optional information to be returned by \$GETMSG. The **outadr** argument is the 32- or 64-bit address (on Alpha systems) or the 32-bit address (on VAX systems) 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 *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=xxxxxxx[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`

## System Service Descriptions

### \$GETMSG

#### 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, refer to 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).

##### 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.

## System Service Descriptions

### \$GETQUI

You can use wildcard mode to make a sequence of calls to \$GETQUI to get information about all characteristics, form definitions, queues, or jobs contained in the system job queue file. For information on using wildcard mode, see the Description section.

#### context

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

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

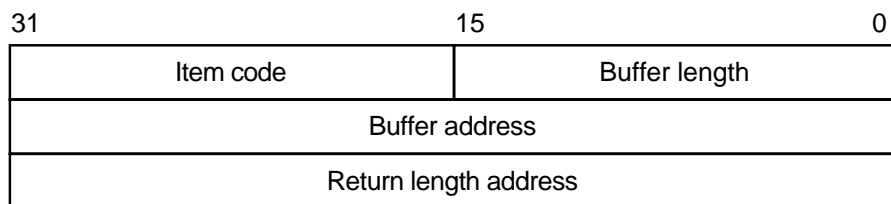
To generate a new context stream, the specified longword must contain -1. \$GETQUI then modifies the longword to hold the context number for that stream of operation. The context is marked with the caller's mode (user, supervisor, executive, or kernel). Any attempt to use that context in successive calls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

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

#### itmlst

OpenVMS usage: item\_list\_3  
type: longword (unsigned)  
access: read only  
mechanism: by reference

Item list supplying information to be used in performing the function specified by the **func** argument. The **itmlst** argument is the address of the item list. The item list consists of one or more item descriptors, each of which contains an item code. The item list is terminated by an item code of 0 or by a longword of 0. The following diagram depicts the structure of a single item descriptor:



ZK-5186A-GE

The following table defines the item descriptor fields:

Descriptor Field	Definition
Buffer length	A word specifying the length of the buffer; the buffer either supplies information to \$GETQUI or receives information from \$GETQUI. The required length of the buffer varies, depending on the item code specified, and is given in the description of each item code.
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.

## System Service Descriptions

### \$GETQUI

#### **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, Compaq 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

## System Service Descriptions

### \$GETQUI

does not find a job meeting the search criteria, it returns the condition value JBC\$\_NOSUCHENT in the first longword of the I/O status block.

You can specify the following input value item codes:

- QUI\$\_SEARCH\_FLAGS
- QUI\$\_SEARCH\_JOB\_NAME
- QUI\$\_SEARCH\_NUMBER
- QUI\$\_SEARCH\_USERNAME

You can specify the following output value item codes:

- QUI\$\_ACCOUNT\_NAME
- QUI\$\_AFTER\_TIME
- QUI\$\_ASSIGNED\_QUEUE\_NAME
- QUI\$\_CHARACTERISTICS
- QUI\$\_CHECKPOINT\_DATA
- QUI\$\_CLI
- QUI\$\_COMPLETED\_BLOCKS
- QUI\$\_CONDITION\_VECTOR
- QUI\$\_CPU\_LIMIT
- QUI\$\_ENTRY\_NUMBER
- QUI\$\_FILE\_COUNT
- QUI\$\_FORM\_NAME
- QUI\$\_FORM\_STOCK
- QUI\$\_JOB\_COMPLETION\_QUEUE
- QUI\$\_JOB\_COMPLETION\_TIME
- QUI\$\_JOB\_COPIES
- QUI\$\_JOB\_COPIES\_DONE
- QUI\$\_JOB\_FLAGS
- QUI\$\_JOB\_NAME
- QUI\$\_JOB\_PID
- QUI\$\_JOB\_RETENTION\_TIME
- QUI\$\_JOB\_SIZE
- QUI\$\_JOB\_STATUS
- QUI\$\_LOG\_QUEUE
- QUI\$\_LOG\_SPECIFICATION
- QUI\$\_NOTE
- QUI\$\_OPERATOR\_REQUEST
- QUI\$\_PARAMETER\_1 through 8
- QUI\$\_PENDING\_JOB\_REASON
- QUI\$\_PRIORITY
- QUI\$\_PROCESSOR
- QUI\$\_QUEUE\_FLAGS
- QUI\$\_QUEUE\_NAME
- QUI\$\_QUEUE\_STATUS
- QUI\$\_REQUEUE\_QUEUE\_NAME
- QUI\$\_RESTART\_QUEUE\_NAME
- QUI\$\_SUBMISSION\_TIME
- QUI\$\_UIC
- QUI\$\_USERNAME
- QUI\$\_WSDEFAULT
- QUI\$\_WSEXTENT
- QUI\$\_WSQUOTA

### **QUI\$\_DISPLAY\_FILE**

This request returns information about the next file defined for the current job context. You normally make this request as part of a nested wildcard sequence of queue-job-file operations or a nested wildcard sequence of entry-file operations; that is, before you make a call to \$GETQUI to request file information, you have already made a call to the \$GETQUI service to establish the job context of the job that contains the files in which you are interested.

The \$GETQUI service signals that it has returned information about all the files defined for the current job context by returning the condition value JBC\$\_NOMOREFILE in the I/O status block. If the current job context contains no files, \$GETQUI returns the condition value JBC\$\_NOSUCHFILE in the I/O status block.

A batch job can make a call to the \$GETQUI service to request information about the command file that is currently executing without first making calls to the service to establish a queue and job context. To do this, the batch job specifies the QUI\$\_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

## System Service Descriptions

### \$GETQUI

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\$\_SEARCH\_THIS\_JOB option of the QUI\$\_SEARCH\_FLAGS item code. The system does not save the queue or job context established in such a call.

For more information about how to request job information, see the Description section.

You can specify the following input value item code:

QUI\$\_SEARCH\_FLAGS

You can specify the following output value item codes:

QUI\$\_ACCOUNT\_NAME  
QUI\$\_AFTER\_TIME  
QUI\$\_CHARACTERISTICS  
QUI\$\_CHECKPOINT\_DATA  
QUI\$\_CLI  
QUI\$\_COMPLETED\_BLOCKS



QUI\$ \_CONDITION\_VECTOR  
QUI\$ \_CPU\_LIMIT  
QUI\$ \_ENTRY\_NUMBER  
QUI\$ \_FILE\_COUNT  
QUI\$ \_FORM\_NAME  
QUI\$ \_FORM\_STOCK  
QUI\$ \_INTERVENING\_BLOCKS  
QUI\$ \_INTERVENING\_JOBS  
QUI\$ \_JOB\_COMPLETION\_QUEUE  
QUI\$ \_JOB\_COMPLETION\_TIME  
QUI\$ \_JOB\_COPIES  
QUI\$ \_JOB\_COPIES\_DONE  
QUI\$ \_JOB\_FLAGS  
QUI\$ \_JOB\_NAME  
QUI\$ \_JOB\_PID  
QUI\$ \_JOB\_RETENTION\_TIME  
QUI\$ \_JOB\_SIZE  
QUI\$ \_JOB\_STATUS  
QUI\$ \_LOG\_QUEUE  
QUI\$ \_LOG\_SPECIFICATION  
QUI\$ \_NOTE  
QUI\$ \_OPERATOR\_REQUEST  
QUI\$ \_PARAMETER\_1 through 8  
QUI\$ \_PENDING\_JOB\_REASON  
QUI\$ \_PRIORITY  
QUI\$ \_QUEUE\_NAME  
QUI\$ \_REQUEUE\_QUEUE\_NAME  
QUI\$ \_RESTART\_QUEUE\_NAME  
QUI\$ \_SUBMISSION\_TIME  
QUI\$ \_UIC  
QUI\$ \_USERNAME  
QUI\$ \_WSDEFAULT  
QUI\$ \_WSEXTENT  
QUI\$ \_WSQUOTA

**QUI\$ \_DISPLAY\_MANAGER**

This request returns information about a specific queue manager, or the next queue manager in a wildcard operation.

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

You must specify the following input value item code:

QUI\$ \_SEARCH\_NAME

You can specify the following input value item code:

QUI\$ \_SEARCH\_FLAGS

You can specify the following output value item codes:

QUI\$ \_MANAGER\_NAME  
QUI\$ \_MANAGER\_NODES  
QUI\$ \_MANAGER\_STATUS

## System Service Descriptions

### \$GETQUI

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\$\_SEARCH\_THIS\_JOB option of the QUI\$\_SEARCH\_FLAGS item code. The system does not save the queue context established in such a call.

For more information about how to request queue information, see the Description section.

You must specify the following input value item code:

QUI\$\_SEARCH\_NAME

You can specify the following input value item code:

QUI\$\_SEARCH\_FLAGS

You can specify the following output value item codes:

QUI\$\_ASSIGNED\_QUEUE\_NAME  
QUI\$\_BASE\_PRIORITY  
QUI\$\_CHARACTERISTICS  
QUI\$\_CPU\_DEFAULT  
QUI\$\_CPU\_LIMIT  
QUI\$\_DEFAULT\_FORM\_NAME  
QUI\$\_DEFAULT\_FORM\_STOCK  
QUI\$\_DEVICE\_NAME  
QUI\$\_EXECUTING\_JOB\_COUNT  
QUI\$\_FORM\_NAME  
QUI\$\_FORM\_STOCK  
QUI\$\_GENERIC\_TARGET  
QUI\$\_HOLDING\_JOB\_COUNT  
QUI\$\_JOB\_LIMIT  
QUI\$\_JOB\_RESET\_MODULES  
QUI\$\_JOB\_SIZE\_MAXIMUM  
QUI\$\_JOB\_SIZE\_MINIMUM  
QUI\$\_LIBRARY\_SPECIFICATION  
QUI\$\_OWNER\_UIC  
QUI\$\_PENDING\_JOB\_BLOCK\_COUNT  
QUI\$\_PENDING\_JOB\_COUNT

QUI\$\_PROCESSOR  
QUI\$\_PROTECTION  
QUI\$\_QUEUE\_DESCRIPTION  
QUI\$\_QUEUE\_FLAGS  
QUI\$\_QUEUE\_NAME  
QUI\$\_QUEUE\_STATUS  
QUI\$\_RETAINED\_JOB\_COUNT  
QUI\$\_SCSNODE\_NAME  
QUI\$\_TIMED\_RELEASE\_JOB\_COUNT  
QUI\$\_WSDEFAULT  
QUI\$\_WSEXTENT  
QUI\$\_WSQUOTA

#### **QUI\$\_TRANSLATE\_QUEUE**

This request translates a logical name for a queue to the equivalence name for the queue. The logical name is specified by QUI\$\_SEARCH\_NAME. The translation is performed iteratively until the equivalence string is found or the number of translations allowed by the system has been reached.

You must specify the following input value item code:

QUI\$\_SEARCH\_NAME

You can specify the following output value item code:

QUI\$\_QUEUE\_NAME

### **Item Codes**

#### **QUI\$\_ACCOUNT\_NAME**

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**

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 \$GETQUI returns the system time at which the job was submitted.

(Valid for QUI\$\_DISPLAY\_ENTRY, QUI\$\_DISPLAY\_JOB function codes)

#### **QUI\$\_ASSIGNED\_QUEUE\_NAME**

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 (::).

## System Service Descriptions

### \$GETQUI

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, refer to the *OpenVMS System Manager's Manual*.

(Valid for `QUI$_DISPLAY_QUEUE` function code)

#### **QUI\$\_BASE\_PRIORITY**

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**

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**

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**

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**

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**

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 `SYSSSYSTEM` 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**

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**

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**

Returns, as a longword integer value, the default CPU time limit specified for the queue in 10-millisecond units. This item code is applicable only to batch execution queues.

For more information about default forms, see the *OpenVMS System Manager's Manual*.

(Valid for QUI\$\_DISPLAY\_QUEUE function code)

**QUI\$\_CPU\_LIMIT**

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**

Returns, as a character string, the name of the default form associated with the specified output queue. Because the form name can include up to 31 characters, the buffer length field of the item descriptor should specify 31 (bytes).

For more information about default forms, see the *OpenVMS System Manager's Manual*.

(Valid for QUI\$\_DISPLAY\_QUEUE function code)

**QUI\$\_DEFAULT\_FORM\_STOCK**

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, refer to the *OpenVMS System Manager's Manual*.

(Valid for QUI\$\_DISPLAY\_QUEUE function code)

**QUI\$\_DEVICE\_NAME**

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)

## System Service Descriptions

### \$GETQUI

#### QUI\$\_ENTRY\_NUMBER

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

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

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

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

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

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.

Symbolic Name	Description
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**

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 NAMST\_DVI field, the 6-byte NAMSW\_FID field, and the 6-byte NAMSW\_DID field.

(Valid for QUI\$\_DISPLAY\_FILE function code)

**QUI\$\_FILE\_SETUP\_MODULES**

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**

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**

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.

## System Service Descriptions

### \$GETQUI

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

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

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

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

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

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**

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**

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**

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**

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, refer to the *OpenVMS System Manager's Manual*.

(Valid for QUI\$\_DISPLAY\_ENTRY, QUI\$\_DISPLAY\_FORM, QUI\$\_DISPLAY\_JOB, QUI\$\_DISPLAY\_QUEUE function codes)

**QUI\$\_FORM\_NUMBER**

Returns, as a longword integer value, the number of the specified form.

(Valid for QUI\$\_DISPLAY\_FORM function code)

**QUI\$\_FORM\_SETUP\_MODULES**

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**

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, refer to the *OpenVMS System Manager's Manual*.

(Valid for QUI\$\_DISPLAY\_ENTRY, QUI\$\_DISPLAY\_FORM, QUI\$\_DISPLAY\_JOB, QUI\$\_DISPLAY\_QUEUE function codes)

**QUI\$\_FORM\_WIDTH**

Returns, as a longword integer value, the width of the specified form in characters.

## System Service Descriptions

### \$GETQUI

(Valid for QUI\$\_DISPLAY\_FORM function code)

#### **QUI\$\_GENERIC\_TARGET**

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**

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**

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**

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**

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**

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**

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**

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**

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 <i>OpenVMS DCL Dictionary</i> .
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.

## System Service Descriptions

### \$GETQUI

Symbolic Name	Description
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_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, refer to the /RETAIN qualifier for the PRINT or SUBMIT command in the <i>OpenVMS DCL Dictionary</i> .
QUI\$V_JOB_WSDEFAULT	Default working set size is specified for the job.
QUI\$V_JOB_WSEXTENT	Working set extent is specified for the job.
QUI\$V_JOB_WSQUOTA	Working set quota is specified for the job.

(Valid for QUI\$\_DISPLAY\_ENTRY, QUI\$\_DISPLAY\_JOB function codes)

#### QUI\$\_JOB\_LIMIT

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

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

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

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**

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 *OpenVMS DCL Dictionary*.

(Valid for QUI\$\_DISPLAY\_ENTRY, QUI\$\_DISPLAY\_JOB function codes)

**QUI\$\_JOB\_SIZE**

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**

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**

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**

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.

## System Service Descriptions

### \$GETQUI

Symbol Name	Description
QUI\$V_JOB_INACCESSIBLE	<p>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:</p> <p>QUI\$_AFTER_TIME  QUI\$_COMPLETED_BLOCKS  QUI\$_ENTRY_NUMBER  QUI\$_INTEVENING_BLOCKS  QUI\$_INTEVENING_JOBS  QUI\$_JOB_SIZE  QUI\$_JOB_STATUS</p>
QUI\$V_JOB_PENDING	Job is pending. See QUI\$_PENDING_JOB_REASON for the reason the job is in a pending state.
QUI\$V_JOB_REFUSED	Job was refused by symbiont and is waiting for symbiont to accept it for processing.
QUI\$V_JOB_RETAINED	Job has completed, but it is being retained in the queue.
QUI\$V_JOB_STALLED	Execution of the job is stalled because the physical device on which the job is printing is stalled.
QUI\$V_JOB_STARTING	The job has been scheduled for execution. Confirmation of execution has not been received.
QUI\$V_JOB_SUSPENDED	Execution of the job is suspended because the queue on which it is executing is paused.
QUI\$V_JOB_TIMED_RELEASE	Job is waiting for specified time to execute.

(Valid for QUI\$\_DISPLAY\_ENTRY, QUI\$\_DISPLAY\_JOB function codes)

#### QUI\$\_LAST\_PAGE

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

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 SYSS\$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**

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**

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 \$\$NDJBC 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**

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**

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**

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)

## System Service Descriptions

### \$GETQUI

#### **QUI\$\_NOTE**

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**

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**

Returns the owner UIC as a longword value in standard UIC format.

(Valid for QUI\$\_DISPLAY\_QUEUE function code)

#### **QUI\$\_PAGE\_SETUP\_MODULES**

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**

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**

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**

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**

Returns, as a longword bit vector, the reason that the job is in a pending state. The \$QUIDEF macro defines the following symbolic names for the flags:



Symbolic Name	Description
QUI\$V_PEND_CHAR_MISMATCH	Job requires characteristics that are not available on the execution queue.
QUI\$V_PEND_JOB_SIZE_MAX	Block size of job exceeds the upper block limit of the execution queue.
QUI\$V_PEND_JOB_SIZE_MIN	Block size of job is less than the lower limit of the execution queue.
QUI\$V_PEND_LOWERCASE_MISMATCH	Job requires lowercase printer.
QUI\$V_PEND_NO_ACCESS	Owner of job does not have access to the execution queue.
QUI\$V_PEND_QUEUE_BUSY	Job is pending because the number of jobs currently executing on the queue equals the job limit for the queue.
QUI\$V_PEND_QUEUE_STATE	Job is pending because the execution queue is not in a running, open state as indicated by QUI\$_QUEUE_STATUS.
QUI\$V_PEND_STOCK_MISMATCH	Stock type required by the job's form does not match the stock type of the form mounted on the execution queue.

(Valid for QUI\$\_DISPLAY\_ENTRY, QUI\$\_DISPLAY\_JOB function codes)

**QUI\$\_PRIORITY**

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**

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 SY\$\$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)

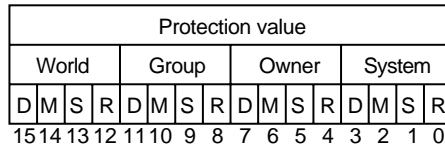
**QUI\$\_PROTECTION**

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

# System Service Descriptions

## \$GETQUI

The following diagram illustrates the protection mask:



ZK-3823A-GE

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

(Valid for `QUI$DISPLAY_QUEUE` function code)

### QUI\$QUEUE\_DESCRIPTION

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

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

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
<code>QUI\$V_QUEUE_ACL_SPECIFIED</code>	An access control list has been specified for the queue. You cannot retrieve a queue's ACL through the <code>\$GETQUI</code> service; instead, you must use the <code>\$GET_SECURITY</code> service.
<code>QUI\$V_QUEUE_AUTOSTART</code>	Queue is designated as an autostart queue.
<code>QUI\$V_QUEUE_BATCH</code>	Queue is a batch queue or a generic batch queue.
<code>QUI\$V_QUEUE_CPU_DEFAULT</code>	A default CPU time limit has been specified for all jobs in the queue.
<code>QUI\$V_QUEUE_CPU_LIMIT</code>	A maximum CPU time limit has been specified for all jobs in the queue.
<code>QUI\$V_QUEUE_FILE_BURST</code>	Burst and flag pages precede each file in each job initiated from the queue.

## System Service Descriptions

### \$GETQUI

Symbolic Name	Description
QUI\$V_QUEUE_FILE_BURST_ONE	Burst and flag pages precede only the first copy of the first file in each job initiated from the queue.
QUI\$V_QUEUE_FILE_FLAG	Flag page precedes each file in each job initiated from the queue.
QUI\$V_QUEUE_FILE_FLAG_ONE	Flag page precedes only the first copy of the first file in each job initiated from the queue.
QUI\$V_QUEUE_FILE_PAGINATE	Output symbiont paginates output for each job initiated from this queue. The output symbiont paginates output by inserting a form feed whenever output reaches the bottom margin of the form.
QUI\$V_QUEUE_FILE_TRAILER	Trailer page follows each file in each job initiated from the queue.
QUI\$V_QUEUE_FILE_TRAILER_ONE	Trailer page follows only the last copy of the last file in each job initiated from the queue.
QUI\$V_QUEUE_GENERIC	The queue is a generic queue.
QUI\$V_QUEUE_GENERIC_SELECTION	The queue is an execution queue that can accept work from a generic queue.
QUI\$V_QUEUE_JOB_BURST	Burst and flag pages precede each job initiated from the queue.
QUI\$V_QUEUE_JOB_FLAG	A flag page precedes each job initiated from the queue.
QUI\$V_QUEUE_JOB_SIZE_SCHED	Jobs initiated from the queue are scheduled according to size, with the smallest job of a given priority processed first (meaningful only for output queues).
QUI\$V_QUEUE_JOB_TRAILER	A trailer page follows each job initiated from the queue.
QUI\$V_QUEUE_PRINTER	The queue is a printer queue.
QUI\$V_QUEUE_RECORD_BLOCKING	The symbiont is permitted to concatenate, or block together, the output records it sends to the output device.
QUI\$V_QUEUE_RETAIN_ALL	All jobs initiated from the queue remain in the queue after they finish executing. Completed jobs are marked with a completion status.
QUI\$V_QUEUE_RETAIN_ERROR	Only jobs that do not complete successfully are retained in the queue.

## System Service Descriptions

### \$GETQUI

Symbolic Name	Description
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\$\_QUEUE\_NAME

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

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 <sup>1</sup>	Queue is processing work but is capable of processing additional work.
QUI\$V_QUEUE_BUSY <sup>1</sup>	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 <sup>1</sup>	Queue is not capable of being started or submitted to.
QUI\$V_QUEUE_IDLE <sup>1</sup>	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.

<sup>1</sup>Bit describes the current state of the queue. Only one of these bits can be set at any time.

Symbolic Name	Description
QUI\$V_QUEUE_PAUSED <sup>1</sup>	Execution of all current jobs in the queue is temporarily halted.
QUI\$V_QUEUE_PAUSING <sup>1</sup>	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 <sup>1</sup>	Queue is restarting after pausing.
QUI\$V_QUEUE_SERVER	Queue processing is directed to a server symbiont.
QUI\$V_QUEUE_STALLED <sup>1</sup>	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 <sup>1</sup>	Queue is starting.
QUI\$V_QUEUE_STOP_PENDING	Queue will be stopped when work currently in progress has completed.
QUI\$V_QUEUE_STOPPED <sup>1</sup>	Queue is stopped.
QUI\$V_QUEUE_STOPPING <sup>1</sup>	Queue is stopping.
QUI\$V_QUEUE_UNAVAILABLE	Physical device to which queue is assigned is not available.

---

<sup>1</sup>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**

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**

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**

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**

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).

## System Service Descriptions

### \$GETQUI

(Valid for QUI\$\_DISPLAY\_QUEUE, QUI\$\_DISPLAY\_MANAGER function codes)

#### QUI\$\_SEARCH\_FLAGS

When you specify QUI\$\_SEARCH\_FLAGS, an input value item code, it specifies a longword bit vector wherein each bit specifies the scope of \$GETQUI's search for objects specified in the call to \$GETQUI. The \$QUIDEF macro defines symbols for each option (bit) in the bit vector.

The following table contains the symbolic names for each option and the function code for which each flag is meaningful:

Symbolic Name	Function Code	Description
QUI\$V_SEARCH_ ALL_JOBS	QUI\$_DISPLAY_JOB	\$GETQUI searches all jobs included in the established queue context. If you do not specify this flag, \$GETQUI only returns information about jobs that have the same user name as the caller.
QUI\$V_SEARCH_ BATCH	QUI\$_DISPLAY_ENTRY QUI\$_DISPLAY_QUEUE	Selects batch queues.
QUI\$V_SEARCH_ EXECUTING_JOBS	QUI\$_DISPLAY_ENTRY QUI\$_DISPLAY_JOB QUI\$_DISPLAY_QUEUE	Selects executing jobs, or queues with executing jobs.
QUI\$V_SEARCH_ FREEZE_CONTEXT	QUI\$_DISPLAY_ CHARACTERISTIC QUI\$_DISPLAY_ENTRY QUI\$_DISPLAY_FILE QUI\$_DISPLAY_FORM QUI\$_DISPLAY_JOB QUI\$_DISPLAY_QUEUE QUI\$_DISPLAY_MANAGER	Does not advance wildcard context on completion of this service call.
QUI\$V_SEARCH_ GENERIC	QUI\$_DISPLAY_ENTRY QUI\$_DISPLAY_QUEUE	Selects generic queues.
QUI\$V_SEARCH_ HOLDING_JOBS	QUI\$_DISPLAY_ENTRY QUI\$_DISPLAY_JOB QUI\$_DISPLAY_QUEUE	Selects jobs on unconditional hold, or queues with jobs on unconditional hold.
QUI\$V_SEARCH_ PENDING_JOBS	QUI\$_DISPLAY_ENTRY QUI\$_DISPLAY_JOB QUI\$_DISPLAY_QUEUE	Selects pending jobs, or queues with pending jobs.
QUI\$V_SEARCH_ PRINTER	QUI\$_DISPLAY_ENTRY QUI\$_DISPLAY_QUEUE	Selects printer queues.
QUI\$V_SEARCH_ RETAINED_JOBS	QUI\$_DISPLAY_ENTRY QUI\$_DISPLAY_JOB QUI\$_DISPLAY_QUEUE	Selects jobs being retained, or queues with jobs being retained.
QUI\$V_SEARCH_ SERVER	QUI\$_DISPLAY_ENTRY QUI\$_DISPLAY_QUEUE	Selects server queues.
QUI\$V_SEARCH_ SYMBIONT	QUI\$_DISPLAY_ENTRY QUI\$_DISPLAY_QUEUE	Selects output queues.

## System Service Descriptions \$GETQUI

Symbolic Name	Function Code	Description
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)

## System Service Descriptions

### \$GETQUI

#### **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**

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**

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**

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**

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**

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**

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**

Returns, in pages (on VAX systems) or pagelets (on Alpha systems), the working set quota for the specified job or queue, which is a longword integer in the range 1 through 65,535. This value is meaningful only for batch jobs and execution queues.

(Valid for QUI\$\_DISPLAY\_ENTRY, QUI\$\_DISPLAY\_JOB, QUI\$\_DISPLAY\_QUEUE function codes)



## Description

The Get Queue Information service returns information about queues and the jobs initiated from those queues. The \$GETQUI and \$SNDJBC services together provide the user interface to the queue manager and job controller processes. See the Description section of the \$SNDJBC service for a discussion of the different types of jobs and queues.

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

You can specify the following function codes to return information for the object types listed:

Function Code	Object Type
QUI\$DISPLAY_CHARACTERISTIC	Characteristic
QUI\$DISPLAY_FORM	Form
QUI\$DISPLAY_QUEUE	Queue
QUI\$DISPLAY_MANAGER	Queue manager
QUI\$DISPLAY_JOB	Job within a queue context
QUI\$DISPLAY_FILE	File within a job context
QUI\$DISPLAY_ENTRY	Job independent of queue

When you call the \$GETQUI service, the queue manager establishes an internal GETQUI context block (GQC). The system uses the GQC to store information temporarily and to keep track of its place in a wildcard sequence of operations. The system provides any number of GQC blocks per process.

To allow you to obtain information either about a particular object in a single call or about several objects in a sequence of calls, \$GETQUI supports three different search modes. The following search modes affect the disposition of the GQC in different ways:

- Nonwildcard mode—\$GETQUI returns information about a particular object in a single call. After the call completes, the system dissolves the GQC.
- Wildcard mode—\$GETQUI returns information about several objects of the same type in a sequence of calls. The system saves the GQC between calls until the wildcard sequence completes.
- Nested wildcard mode—\$GETQUI returns information about objects defined within another object. Specifically, this mode allows you to query jobs contained in a selected queue or files contained in a selected job in a sequence of calls. After each call, the system saves the GQC so that the GQC can provide the queue or job context necessary for subsequent calls.

The sections that follow describe how each of the three search methods affects \$GETQUI's search for information; how you direct \$GETQUI to undertake each method; and how each method affects the contents of the GQC.

## System Service Descriptions

### \$GETQUI

#### Nonwildcard Mode

In nonwildcard mode, \$GETQUI can return information about the following objects:

- A specific characteristic or form definition that you identify by name or number.
- A specific queue that you identify by name.
- A specific queue manager that you identify by name.
- A specific batch or print job that you identify by job entry number or by name.
- The queue, job, or executing command procedure file associated with the calling batch job. You invoke this special case of nonwildcard mode by specifying the QUI\$\_SEARCH\_THIS\_JOB option of the QUI\$\_SEARCH\_FLAGS item code for a display queue, job, or file operation.

To obtain information about a specific characteristic or form definition, you call \$GETQUI using the QUI\$\_DISPLAY\_CHARACTERISTIC or QUI\$\_DISPLAY\_FORM function code. You need to specify either the name of the characteristic or form in the QUI\$\_SEARCH\_NAME item code or the number of the characteristic or form in the QUI\$\_SEARCH\_NUMBER item code. The name string you specify cannot include either of the wildcard characters (\* or %). You can specify both the QUI\$\_SEARCH\_NAME and QUI\$\_SEARCH\_NUMBER item codes, but the name and number you specify must be associated with the same characteristic or form definition.

To obtain information about a specific queue definition, you specify the QUI\$\_DISPLAY\_QUEUE function code and provide the name of the queue in the QUI\$\_SEARCH\_NAME item code. The name string you specify cannot include the wildcard characters (\* or %).

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

To obtain information about a specific batch or print job, specify the QUI\$\_DISPLAY\_ENTRY function code and provide the entry number of the job in the QUI\$\_SEARCH\_NUMBER item code.

Finally, the \$GETQUI service provides an option that allows a batch job to obtain information about the queue, job, or command file that the associated batch job is executing without first entering wildcard mode to establish a queue or job context. You can make a call from the batch job that specifies the QUI\$\_DISPLAY\_QUEUE function code to obtain information about the queue from which the batch job was initiated; the QUI\$\_DISPLAY\_JOB function code to obtain information about the batch job itself; or the QUI\$\_DISPLAY\_FILE function code to obtain information about the command file for the batch job. For each of these calls, you must select the QUI\$V\_SEARCH\_THIS\_JOB option of the QUI\$\_SEARCH\_FLAGS item code. When you select this option, \$GETQUI ignores all other options in the QUI\$\_SEARCH\_FLAGS item code.

**Wildcard Mode**

In wildcard mode, the system saves the GQC between calls to \$GETQUI so that you can make a sequence of calls to \$GETQUI to get information about all characteristics, forms, queues, jobs, or queue managers contained in the queue database.

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

To set up a wildcard search for characteristic or form definitions, specify the QUI\$\_DISPLAY\_CHARACTERISTIC or QUI\$\_DISPLAY\_FORM function code and specify a name in the QUI\$\_SEARCH\_NAME item code that includes one or more wildcard characters (\* or %).

To set up a wildcard search for queues, use the QUI\$\_DISPLAY\_QUEUE function code and specify a name in the QUI\$\_SEARCH\_NAME item code that includes one or more wildcard characters (\* or %). You can indicate the type of the queue you want to search for by specifying any combination of the following options for the QUI\$\_SEARCH\_FLAGS item code:

- QUI\$V\_SEARCH\_BATCH
- QUI\$V\_SEARCH\_PRINTER
- QUI\$V\_SEARCH\_SERVER
- QUI\$V\_SEARCH\_TERMINAL
- QUI\$V\_SEARCH\_SYMBIONT
- QUI\$V\_SEARCH\_GENERIC

For example, if you select the QUI\$V\_SEARCH\_BATCH option, \$GETQUI returns information only about batch queues; if you select the QUI\$V\_SEARCH\_SYMBIONT option, \$GETQUI returns information only about output queues (printer, terminal, and server queues). If you specify none of the queue type options, \$GETQUI searches all queues.

To set up a wildcard search for queue managers, specify the QUI\$\_DISPLAY\_MANAGER function code and specify a name in the QUI\$\_SEARCH\_NAME item code that includes one or more wildcard characters (\* or %).

To set up a wildcard search for jobs, specify the QUI\$\_DISPLAY\_ENTRY function code and the QUI\$\_SEARCH\_WILDCARD option of the QUI\$\_SEARCH\_FLAGS item code. When you specify this option, omit the QUI\$\_SEARCH\_NUMBER item code. You can restrict the search to jobs having particular status or to jobs residing in specific types of queues, or both, by including any combination of the following options for the QUI\$\_SEARCH\_FLAGS item code:

- QUI\$V\_SEARCH\_BATCH
- QUI\$V\_SEARCH\_EXECUTING\_JOBS
- QUI\$V\_SEARCH\_HOLDING\_JOBS
- QUI\$V\_SEARCH\_PENDING\_JOBS
- QUI\$V\_SEARCH\_PRINTER
- QUI\$V\_SEARCH\_RETAINED\_JOBS
- QUI\$V\_SEARCH\_SERVER
- QUI\$V\_SEARCH\_SYMBIONT
- QUI\$V\_SEARCH\_TERMINAL
- QUI\$V\_SEARCH\_TIMED\_RELEASE\_JOBS

## System Service Descriptions

### \$GETQUI

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

## System Service Descriptions

### \$GETQUI

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, \$STRNLNM

### 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 <b>astadr</b> argument was specified, and the process has exceeded its ASTLM quota.
SS\$_ILLEFC	The <b>efn</b> 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 <b>efn</b> 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.

## System Service Descriptions

### \$GETQUI

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.

# System Service Descriptions

## \$GETQUI

### 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$_SEARCH_THIS_JOB
```



## System Service Descriptions \$GETQUI

```
! 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\$\_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 BUFLN, ITMCD
  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
```

## System Service Descriptions

### \$GETQUI

```
! 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).ITMCO = QUI$_SEARCH_NAME
QUEUE_LIST(1).BUFADR = %LOC(SEARCH_NAME)
QUEUE_LIST(1).RETADR = 0
QUEUE_LIST(2).BUFLEN = 4
QUEUE_LIST(2).ITMCO = QUI$_SEARCH_FLAGS
QUEUE_LIST(2).BUFADR = %LOC(SEARCH_FLAGS)
QUEUE_LIST(2).RETADR = 0
QUEUE_LIST(3).BUFLEN = 31
QUEUE_LIST(3).ITMCO = 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).ITMCO = QUI$_SEARCH_FLAGS
JOB_LIST(1).BUFADR = %LOC(SEARCH_FLAGS)
JOB_LIST(1).RETADR = 0
JOB_LIST(2).BUFLEN = 4
JOB_LIST(2).ITMCO = QUI$_JOB_SIZE
JOB_LIST(2).BUFADR = %LOC(JOB_SIZE)
JOB_LIST(2).RETADR = 0
JOB_LIST(3).BUFLEN = 39
JOB_LIST(3).ITMCO = QUI$_JOB_NAME
JOB_LIST(3).BUFADR = %LOC(JOB_NAME)
JOB_LIST(3).RETADR = %LOC(JOB_NAME_LEN)
JOB_LIST(4).BUFLEN = 12
JOB_LIST(4).ITMCO = QUI$_USERNAME
JOB_LIST(4).BUFADR = %LOC(USERNAME)
JOB_LIST(4).RETADR = %LOC(USERNAME_LEN)
JOB_LIST(5).BUFLEN = 4
JOB_LIST(5).ITMCO = 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$_SEARCH_WILDCARD .OR.
2              QUI$_SEARCH_SYMBIONT .OR.
2              QUI$_SEARCH_ALL_JOBS)

! Dissolve any internal search context block for the process
STATUS_Q = SYS$GETQUIW (,%VAL(QUI$_CANCEL_OPERATION),,,,,)
```

## System Service Descriptions \$GETQUI

```

! 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$_JOB_INACCESSIBLE)
    IF (NOACCESS .NE. 0) THEN
      TYPE 9030, JOB_SIZE
    ELSE
      TYPE 9040, JOB_SIZE,
2          USERNAME(1:USERNAME_LEN),
2          JOB_NAME(1:JOB_NAME_LEN)
    ENDIF
  ENDIF
ENDDO
ENDDO

9000  FORMAT (' Enter queue name to search: ', $)
9010  FORMAT (Q, A31)
9020  FORMAT ('0Queue name = ', A)
9030  FORMAT ('   Job size = ', I5, '   <no read access privilege>')
9040  FORMAT ('   Job size = ', I5,
2          '   Username = ', A, T46,
2          '   Job name = ', A)
END

```

This Fortran program demonstrates how any job can obtain information about other jobs from the system job queue file by using the \$GETQUIW system service. This program lists all print jobs in output queues with a job size of 500 blocks or more. It also displays queue name, job size, user name, and job name information for each job listed.

## \$GETQUIW

### Get Queue Information and Wait

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

The \$GETQUIW service completes synchronously; that is, it returns to the caller with the requested information. For asynchronous completion, use the Get Queue Information (\$GETQUI) service; \$GETQUI returns to the caller after queuing the information request, without waiting for the information to be returned.

In all other respects, \$GETQUIW is identical to \$GETQUI. For more information about \$GETQUIW, refer to the description of \$GETQUI in this manual.

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

### Format

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

### 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);
```

---

## \$GETRMI

### Get Resource Monitor Information

Returns system performance information about the local system.

#### Format

SYS\$GETRMI [efn] [,nullarg] [,nullarg] ,itmlst [,iosb] [,astadr] [,astprm]

#### C Prototype

```
int 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

Placelholding argument reserved to Compaq.

##### nullarg

OpenVMS usage: null\_arg  
type: longword (unsigned)  
access: read only  
mechanism: by value

Placelholding argument reserved to Compaq.

##### itmlst

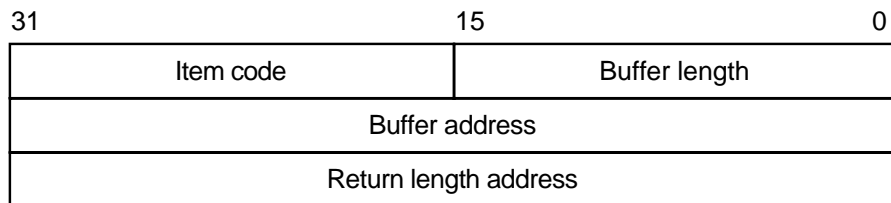
OpenVMS usage: item\_list\_3  
type: longword (unsigned)  
access: read only  
mechanism: by reference

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.

## System Service Descriptions

### \$GETRMI

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



ZK-5186A-GE

The following table defines the item descriptor fields:

Descriptor Field	Definition
Buffer length	A word 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\$\_BLKAST**

Returns the number of blocking ast's.

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/O's

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\$\_DIRIO**

Returns the number of direct I/O's

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\$\_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\$\_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\$\_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).

## System Service Descriptions

### \$GETRMI

#### **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\$\_FAULTS**

Returns the number of page faults since last system initialization.

This number is a longword, therefore 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\$\_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\$\_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\$\_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 window hits tics used by the FCP.

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\$\_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\$\_FREFLT**

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 freelist.

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

**RMI\$\_GVALFLT**

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\$\_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\$\_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\$\_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\$\_MODLIST**

Returns the number of pages on the modified page list.

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

**RMI\$\_MFYFLT**

Returns the number of page faults from the modified list.

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

## System Service Descriptions

### \$GETRMI

#### **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\$\_PREADIO**

Returns physical page read I/O's.

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

#### **RMI\$\_PREADS**

Returns the number of pages read.

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

#### **RMI\$\_PWRITES**

Returns the number of pages written.

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

#### **RMI\$\_PWRITIO**

Returns physical page write I/O's.

This number is a longword, therefore 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\$\_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).

## Description

The Get Resource Monitor Information service returns performance information about the local system.

#### **Required Access or Privileges**

CMKRNL is required to return system performance data.

#### **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 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.

#### csidadr

OpenVMS usage: process\_id  
type: longword (unsigned)  
access: modify  
mechanism: by 32- or 64-bit reference (Alpha)  
mechanism: by 32-bit reference (VAX)

OpenVMS Cluster system identification of the node about which \$GETSYI is to return information. The **csidadr** argument is the 32-bit address (on VAX systems) or the 32- or 64-bit address (on Alpha systems) 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 SSS\_NOMORENODE after each call to \$GETSYI and should stop calling \$GETSYI when SSS\_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 (Alpha)  
 mechanism: by 32-bit descriptor-fixed-length string descriptor (VAX)

Name of the node about which \$GETSYI is to return information. The **nodename** argument is the 32-bit address (on VAX systems) or the 32- or 64-bit address (on Alpha systems) 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.

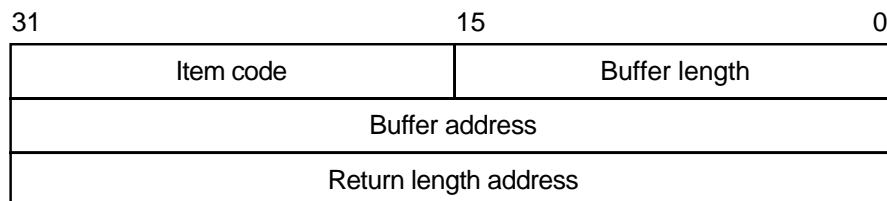
You can also specify a node to \$GETSYI by using the **csidadr** argument. See the description of **csidadr**.

**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 (Alpha)  
 mechanism: by 32-bit reference (VAX)

Item list specifying which information is to be returned about the node or nodes. The **itmlst** argument is the 32-bit address (on VAX systems) or the 32- or 64-bit address (on Alpha systems) 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

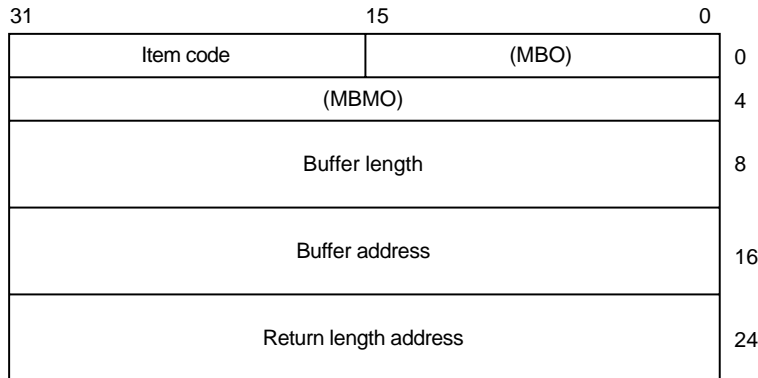
## System Service Descriptions

### \$GETSYI

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 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, \$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 \$SYIDF 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-GE

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 \$SYIDF macro defines these codes. A description of each item code is given in the Item Codes section.

Descriptor Field	Definition
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 \$GETSYI 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 is too small, \$GETSYI truncates the data.
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 (Alpha)  
 mechanism: by 32-bit reference (VAX)

I/O status block to receive the final completion status. The **iosb** argument is the 32-bit address (on VAX systems) or the 32- or 64-bit address (on Alpha systems) 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, Compaq 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.

## System Service Descriptions

### \$GETSYI

#### **astadr**

OpenVMS usage: ast\_procedure  
type: procedure value  
access: call without stack unwinding  
mechanism: by 32- or 64-bit reference (Alpha)  
mechanism: by 32-bit reference (VAX)

AST service routine to be executed when \$GETSYI completes. The **astadr** argument is the 32-bit address (on VAX systems) or the 32- or 64-bit address (on Alpha systems) of this routine.

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

#### **astprm**

OpenVMS usage: user\_arg  
type: longword (unsigned)  
access: read only  
mechanism: by value

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

## Item Codes

#### **SYI\$\_ACTIVECPU\_CNT**

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\$\_ACTIVE\_CPU\_MASK**

On Alpha systems, returns a mask of the CPUs actively participating in the current boot of the symmetric multiprocessing (SMP) system.

#### **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. Currently, either of two strings is returned: "Alpha" for Alpha or "VAX" for VAX.

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

#### **SYI\$\_ARCH\_TYPE**

Returns the type of CPU architecture on which the process is executing. SYI\$\_ARCH\_TYPE returns 1 on VAX or 2 on Alpha.

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

#### **SYI\$\_CPU\_AUTOSTART**

On Alpha systems, returns the array of CPU autostart states.



**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\$\_AVAIL\_CPU\_MASK**

On Alpha systems, returns a mask of the present and available CPUs participating in the current boot of the symmetric multiprocessing (SMP) system.

**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.

## System Service Descriptions

### \$GETSYI

#### **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 systems, returns the integer Galaxy community ID.

#### **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**

On VAX systems, returns the CPU processor type, as represented in the processor's system identification (SID) register.

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

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

The \$PRDEF macro defines the following symbols for the processor types:

<b>Processor</b>	<b>Symbol</b>
VAX-11/730	PR\$_SID_TYP730
VAX-11/750	PR\$_SID_TYP750
VAX-11/780, 785	PR\$_SID_TYP780
VAXstation II, II/GPX, and MicroVAX II	PR\$_SID_TYPUV2
VAXstation 2000/MicroVAX 2000	PR\$_SID_TYP410
VAX 8200, 8250, 8300, 8350	PR\$_SID_TYP8SS
VAX 8530, 8550, 8810 (8700), and 8820-N (8800)	PR\$_SID_TYP8NN

<b>Processor</b>	<b>Symbol</b>
VAX 8600, 8650	PR\$_SID_TYP790
VAX 8820, 8830, 8840	PR\$_SID_TYP8PS
VAXft 3000 Model 310	PR\$_SID_TYP520
VAXstation, MicroVAX 3100 series	PR\$_SID_TYP420
MicroVAX 3300, 3400, 3500, 3600, 3800, 3900	PR\$_SID_TYP650
VAXstation 3520, 3540	PR\$_SID_TYP60
VAX 4000-300	PR\$_SID_TYP670
VAX 6000-200, 6000-300 series	PR\$_SID_TYP9CC
VAX 6000-400 series	PR\$_SID_TYP9RR
VAX 9000-200, 9000-400 series	PR\$_SID_TYP9AQ

On Alpha systems, \$GETSYI returns PR\$\_SID\_TYP\_NOTAVAX.

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

**SYI\$\_CPUCAP\_MASK**

On Alpha 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**

On Alpha systems, returns the CPU Configuration bit mask: 0 through 31.

**SYI\$\_CPU\_FAILOVER**

On Alpha systems, returns the array of CPU failover partitions.

**SYI\$\_CPUTYPE**

On Alpha 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:

<b>Processor Code</b>	<b>Processor</b>
2	21064
4	21066
5	21164
6	21064A

# System Service Descriptions

## \$GETSYI

Processor Code	Processor
7	21164A
8	21264

### 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 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 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 pages (on VAX systems) or pagelets (on Alpha systems) in an error log buffer.

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

**SYI\$\_ERRORLOGBUFFERS**

Returns the number of system pages (on VAX systems) or pagelets (on Alpha systems) in use as buffers for the error logger.

Because this number is a word in length, the buffer length field in the item descriptor should specify 2 (bytes).

**SYI\$\_F\_FLOAT\_EMULATED**

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.

## System Service Descriptions

### \$GETSYI

#### **SYI\$\_GALAXY\_MEMBER**

On Alpha systems, returns 1 if you are member of a Galaxy sharing community, 0 if you are not a member.

#### **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.

#### **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.

#### **SYI\$\_GH\_RSRVPGCNT**

On Alpha 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\$\_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\$\_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 node.

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

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

Because SYI\$\_HW\_MODEL is a word, the buffer length field in the item descriptor should specify 2 (bytes).

#### **SYI\$\_HW\_NAME**

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

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

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

<b>Alpha Model Processor Name</b>	<b>Alpha Model Type</b>
DEC 3000 400	ALPHA\$K_A3000_400W
DEC 3000 400S	ALPHA\$K_A3000_400S
DEC 3000 500	ALPHA\$K_A3000_500W

<b>Alpha Model Processor Name</b>	<b>Alpha Model Type</b>
DEC 3000 500S	ALPHASK_A3000_500S
DEC 4000 610	ALPHASK_A4000_610
DEC 4000 620	ALPHASK_A4000_620
DEC 4000 630	ALPHASK_A4000_630
DEC 4000 640	ALPHASK_A4000_640
DEC 7000 Model 610	ALPHASK_A7000_610
DEC 7000 Model 620	ALPHASK_A7000_620
DEC 7000 Model 630	ALPHASK_A7000_630
DEC 7000 Model 640	ALPHASK_A7000_640
DEC 10000 Model 610	ALPHASK_A10000_610
DEC 10000 Model 620	ALPHASK_A10000_620
DEC 10000 Model 630	ALPHASK_A10000_630
DEC 10000 Model 640	ALPHASK_A10000_640

The following table lists the VAX model processor names and the corresponding model types:

<b>VAX Model Processor Name</b>	<b>VAX Model Type</b>
VAX-11/730	VAXSK_V730
VAX-11/750	VAXSK_V750
VAX-11/780	VAXSK_V780
VAX-11/785	VAXSK_V785
MicroVAX II	VAXSK_VUV2
VAXstation II	VAXSK_VWS2
VAXstation II/GPX	VAXSK_VWSD
VAXstation 2000	VAXSK_VWS2000
MicroVAX 2000	VAXSK_VUV2000
VAXstation 2000/GPX	VAXSK_VWSD2000
VAX 8200	VAXSK_V8200
VAX 8250	VAXSK_V8250
VAX 8300	VAXSK_V8300
VAX 8350	VAXSK_V8350
VAX 8530	VAXSK_V8500
VAX 8550	VAXSK_V8550
VAX 8600	VAXSK_V8600
VAX 8650	VAXSK_V8650
VAX 8810 (8700)	VAXSK_V8700
VAX 8820-N (8800)	VAXSK_V8800
VAX 8820, 8830, or 8840 with one CPU enabled	VAXSK_V8810

## System Service Descriptions

### \$GETSYI

VAX Model Processor Name	VAX Model Type
VAX 8820	VAXSK_V8820
VAX 8830	VAXSK_V8830
VAX 8840	VAXSK_V8840
VAXft 3000 Model 310	VAXSK_V520FT
VAXstation 3520	VAXSK_V3520L
VAXstation 3540	VAXSK_V3540L
VAX 4000-300 timeshare	VAXSK_V670
VAX 4000-300 server	VAXSK_V670_S
VAX 6000-210 timeshare	VAXSK_V6210_T
VAX 6000-220 timeshare	VAXSK_V6220_T
VAX 6000-230 timeshare	VAXSK_V6230_T
VAX 6000-240 timeshare	VAXSK_V6240_T
VAX 6000-250 timeshare	VAXSK_V6250_T
VAX 6000-260 timeshare	VAXSK_V6260_T
VAX 6000-210 server	VAXSK_V6210_S
VAX 6000-220 server	VAXSK_V6220_S
VAX 6000-310 timeshare	VAXSK_V6310_T
VAX 6000-320 timeshare	VAXSK_V6320_T
VAX 6000-330 timeshare	VAXSK_V6330_T
VAX 6000-340 timeshare	VAXSK_V6340_T
VAX 6000-350 timeshare	VAXSK_V6350_T
VAX 6000-360 timeshare	VAXSK_V6360_T
VAX 6000-310 server	VAXSK_V6310_S
VAX 6000-320 server	VAXSK_V6320_S
VAX 6000-410 timeshare	VAXSK_V9RR10_T
VAX 6000-420 timeshare	VAXSK_V9RR20_T
VAX 6000-430 timeshare	VAXSK_V9RR30_T
VAX 6000-440 timeshare	VAXSK_V9RR40_T
VAX 6000-450 timeshare	VAXSK_V9RR50_T
VAX 6000-460 timeshare	VAXSK_V9RR60_T
VAX 6000-410 server	VAXSK_V9RR10_S
VAX 6000-420 server	VAXSK_V9RR20_S
VAX 9000-210	VAXSK_V9AR10
VAX 9000-410	VAXSK_V9AQ10
VAX 9000-420	VAXSK_V9AQ20
VAX 9000-430	VAXSK_V9AQ30
VAX 9000-440	VAXSK_V9AQ40

#### **SYI\$\_IO\_PREFER\_CPU**

On Alpha systems, returns the bit mask of CPUs available to be Fast Path preferred CPUs.



**SYI\$\_ITB\_ENTRIES**

On Alpha 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 systems, returns the maximum number of CPUs that can exist in the current configuration of the symmetric multiprocessing (SMP) system.

**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**

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.

## System Service Descriptions

### \$GETSYI

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 software type of the node. The software type indicates whether the node is a VAX system, an Alpha system, or an HSC storage controller.

Because the software type is a 4-byte ASCII string, the buffer length field in the item descriptor should specify 4 (bytes).

#### **SYI\$\_NODE\_SWVERS**

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).

**SYIS\_PAGE\_SIZE**

Returns the number of CPU-specific bytes per page in the system.

On VAX systems, \$GETSYI always returns 512.

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

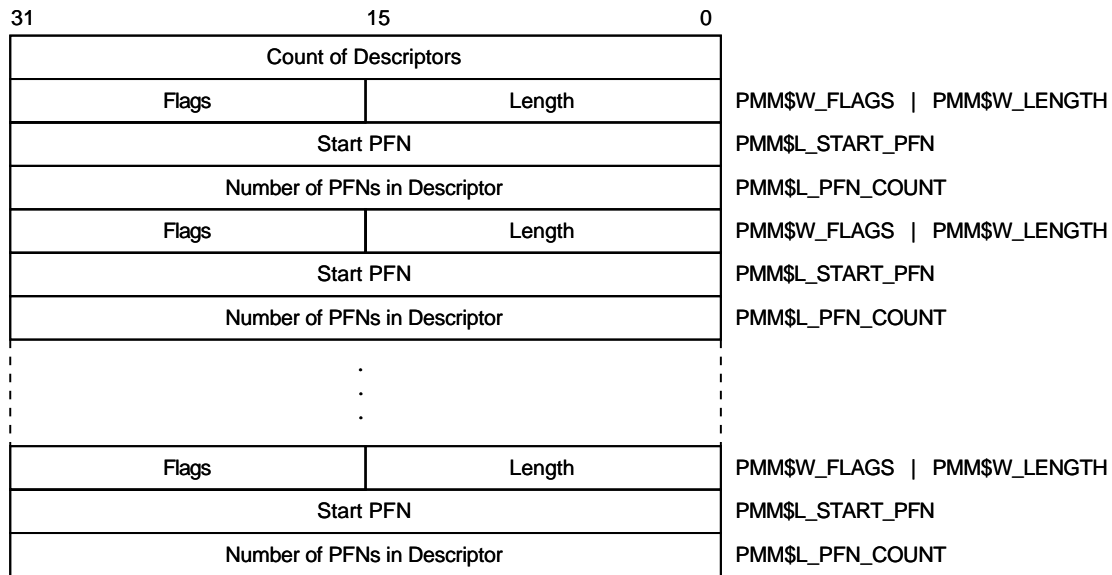
On Alpha and VAX systems, because this number is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

**SYIS\_PARTITION\_ID**

On Alpha systems, returns the integer Galaxy partition ID.

**SYIS\_PFN\_MEMORY\_MAP**

Returns a map describing the system's use of physical memory. The following figure shows an example of a physical memory map:



ZK-8783A-GE

The first longword of the physical memory contains a count of descriptors. This number is equal to the value returned when the SYIS\_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 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).

## System Service Descriptions

### \$GETSYI

Bit	Meaning When Set
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 for Compaq.

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, Compaq 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\$\_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\$\_POTENTIALCPU\_CNT**

On Alpha systems, returns the number of CPUs that could potentially run on this instance.

#### **SYI\$\_POTENTIAL\_CPU\_MASK**

On Alpha systems, returns the mask of CPUs that are compatible on this instance.

#### **SYI\$\_PRIMARY\_CPUID**

On Alpha systems, returns the ID of the primary CPU in the current boot of the symmetric multiprocessing (SMP) system.

**SYI\$\_PROCESS\_SPACE\_LIMIT**

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

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

**SYI\$\_PSXFIFO\_PRIO\_MAX**

On Alpha 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 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 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 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 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 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 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.

Note: OpenVMS support for RADs is available only on the new AlphaServer GS series systems. For more information about using RADs, refer to the *OpenVMS Alpha Partitioning and Galaxy Guide*.

## System Service Descriptions

### \$GETSYI

#### **SYI\$\_RAD\_MEMSIZE**

On Alpha 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.

Note: OpenVMS support for RADs is available only on the new AlphaServer GS series systems. For more information about using RADs, refer to the *OpenVMS Alpha Partitioning and Galaxy Guide*.

#### **SYI\$\_RAD\_MAX\_RAD**

On Alpha systems, returns the maximum number of RADs possible on this platform. If there is no RAD support, 1 is returned.

Note: OpenVMS support for RADs is available only on the new AlphaServer GS series systems. For more information about using RADs, refer to the *OpenVMS Alpha Partitioning and Galaxy Guide*.

#### **SYI\$\_RAD\_SHMEMSIZE**

On Alpha 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.

Note: OpenVMS support for RADs is available only on the new AlphaServer GS series systems. For more information about using RADs, refer to the *OpenVMS Alpha Partitioning and Galaxy Guide*.

#### **SYI\$\_REAL\_CPUYPE**

Returns the actual CPU type of the primary CPU of the system.

See the SYI\$\_CPUYPE item code for a list of symbols and processors.

#### **SYI\$\_SCSNODE**

On Alpha systems, returns the ASCII string Galaxy instance name.

#### **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 out of the Hardware Restart Parameter Block (HWRPB).

#### **SYI\$\_SHARED\_VA\_PTES**

On Alpha 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 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.
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\$\_SYSTYPE**

On Alpha 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).

## System Service Descriptions

### \$GETSYI

#### 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\$\_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).

On VAX systems, the \$PRDEF macro defines the following symbols for the extended processor types:

VAX Processor Type Symbol	Extended Processor Type	Extended Processor Symbol
PR\$_SID_TYPUV	MicroVAX II VAXstation II	PR\$_XSID_UV_UV2
	MicroVAX 2000 VAXstation 2000	PR\$_XSID_UV_410
PR\$_SID_TYPCV	MicroVAX 3300, 3400, 3500, 3600, 3800, 3900 series	PR\$_XSID_CV_650
	VAX 6000-200, 6000- 300 series	PR\$_XSID_CV_9CC
	VAXstation 3520, 3540	PR\$_XSID_CV_60
	VAXstation 3100 series VAXft 3000 Model 310	PR\$_XSID_CV_420 PR\$_XSID_CV_520
PR\$_SID_TYP8NN	VAX 8530	PR\$\$_XSID_N8500
	VAX 8550	PR\$\$_XSID_N8550
	VAX 8810 (8700)	PR\$\$_XSID_N8700
	VAX 8820-N (8800)	PR\$\$_XSID_N8800
PR\$_SID_TYPRV	VAX 4000-300	PR\$_XSID_RV_670
	VAX 6000-400 series	PR\$_XSID_RV_9RR

#### 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 SYIS\_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).

**SYIS\_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 *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, \$SENDERR, \$SENDJBC, \$SENDJBCW, \$SENDOPR

**Condition Values Returned**

SSS_NORMAL	The service completed successfully.
SSS_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.
SSS_BADPARAM	The item list contains an invalid item code.
SSS_EXASTLM	The process has exceeded its AST limit quota.
SSS_NOMORENODE	You requested a wildcard operation, and \$GETSYI has returned information about all available nodes.
SSS_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.

## System Service Descriptions

### \$GETSYI

#### 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$getsysiw (
    EFN$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.

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

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

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

### 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.

---

## \$GETTIM Get Time

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

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

### Format

SYS\$GETTIM timadr

### 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 (Alpha)

mechanism: by 32-bit reference (VAX)

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

### Description

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

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

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

#### Required Access or Privileges

None

#### Required Quota

None

#### Related Services

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

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

### Condition Values Returned

SS\$NORMAL

The service completed successfully.

SS\$ACCVIO

The quadword to receive the time cannot be written by the caller.

## \$GETUAI

### Get User Authorization Information

Returns authorization information about a specified user.

#### Format

SYS\$GETUAI [nullarg] ,[contxt] ,usnam ,itmlst ,[nullarg] ,[nullarg] ,[nullarg]

#### C Prototype

```
int sys$getuai (unsigned int efn, unsigned int *contxt, void *usnam, 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

Placelholding argument reserved to Compaq.

##### usnam

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

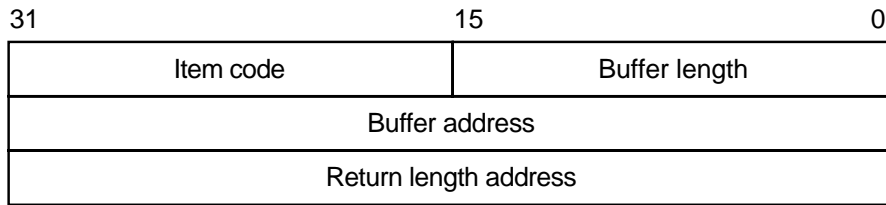
Name of the user about whom \$GETUAI returns authorization information. The **usnam** argument is the address of a descriptor pointing to a character text string containing the user name. The user name string can contain a maximum of 12 alphanumeric characters.

##### itmlst

OpenVMS usage: item\_list\_3  
type: longword (unsigned)  
access: read only  
mechanism: by reference

Item list specifying which information from the specified user's user authorization file (UAF) record is to be returned. The **itmlst** argument is the address of a list of one or more item descriptors, each of which specifies an item code. The item list is terminated by an item code value of 0 or by a longword value of 0.

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



ZK-5186A-GE

The following table defines the item descriptor fields:

Descriptor Field	Definition
Buffer length	A word specifying the length (in bytes) of the buffer in which \$GETUAI is to write the information. The length of the buffer varies, depending on the item code specified in the item code field of the item descriptor, and is given in the description of each item code. If the value of the buffer length field is too small, \$GETUAI truncates the data.
Item code	A word containing a user-supplied symbolic code specifying the item of information that \$GETUAI is to return. The \$UAIDEF macro defines these codes.
Buffer address	A longword containing the user-supplied address of the buffer in which \$GETUAI is to write the information.
Return length address	A longword containing the user-supplied address of a word in which \$GETUAI writes the length in bytes of the information it actually returned.

The symbolic codes have the following format:

\$UAI\_code

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

**contxt**

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

On VAX systems, 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.

## System Service Descriptions

### \$GETUAI

#### Item Codes

##### **UAI\$\_ACCOUNT**

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**

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**

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**

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**

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**

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**

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**

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**

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 SYSS\$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**

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**

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**

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 pages (on VAX systems) or pagelets (on Alpha systems).

Because the default working set size is a longword decimal number, the buffer length field in the item descriptor should specify 4 (bytes).

**UAI\$\_DIOLM**

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**

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**

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**

Returns one of the values shown in the following table, identifying the encryption algorithm for the primary password.

## System Service Descriptions

### \$GETUAI

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

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

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

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

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

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. Same as /FLAG = DISUSER qualifier in AUTHORIZE.
UAI\$V_DISCTLY	User cannot use Ctrl/Y.
UAI\$V_DISFORCE_PWD_CHANGE	User will not be forced to change expired passwords at login.
UAI\$V_DISIMAGE	User cannot issue the RUN or MCR commands or use the foreign command mechanism in DCL.
UAI\$V_DISMAIL	Announcement of new mail is suppressed.
UAI\$V_DISPWDDIC	Automatic checking of user-selected passwords against the system dictionary is disabled.
UAI\$V_DISPWDHIS	Automatic checking of user-selected passwords against previously used passwords is disabled.
UAI\$V_DISRECONNECT	User cannot reconnect to existing processes.
UAI\$V_DISREPORT	User will not receive last login messages.
UAI\$V_DISWELCOME	User will not receive the login welcome message.
UAI\$V_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 SY\$\$SETUAI and is likely to be inconsistent with the user's external user password. If password migration is enabled, the system will attempt to update the external authentication service 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_RESTRICTED	User is limited to operating under a restricted account. (Refer to the <i>OpenVMS Guide to System Security</i> for a description of restricted and captive accounts.)

**UAI\$\_JTQUOTA**

Returns the initial byte quota with which the jobwide logical name table is to be created.

## System Service Descriptions

### \$GETUAI

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

#### **UAI\$\_LASTLOGIN\_I**

Returns, as a quadword absolute time value, the date of the last interactive login.

#### **UAI\$\_LASTLOGIN\_N**

Returns, as a quadword absolute time value, the date of the last noninteractive login.

#### **UAI\$\_LGICMD**

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**

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**

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**

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**

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**

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**

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**

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**

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**

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**

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

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

**UAI\$\_PRCCNT**

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**

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**

Returns, as a longword bit vector, the primary and secondary days of the week.

Each bit represents a day of the week, with the bit clear representing a primary day and the bit set representing a secondary day. The \$UAIDEF macro defines the following symbolic names for these bits:

UAI\$V\_MONDAY  
UAI\$V\_TUESDAY  
UAI\$V\_WEDNESDAY

## System Service Descriptions

### \$GETUAI

UAI\$V\_THURSDAY  
UAI\$V\_FRIDAY  
UAI\$V\_SATURDAY  
UAI\$V\_SUNDAY

#### **UAI\$\_PRIV**

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**

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**

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**

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**

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**

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**

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**

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**

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**

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**

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**

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**

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**

Returns, as a longword, the user identification code (UIC). For the format of the UIC, see the *OpenVMS Guide to System Security*.

**UAI\$\_USER\_DATA**

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 *OpenVMS Guide to System Security*.

**UAI\$\_WSEXTENT**

Returns the working set extent, in pages (on VAX systems) or pagelets (on Alpha systems), for the user of the specified queue or job.

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

**UAI\$\_WSQUOTA**

Returns the working set quota, in pages (on VAX systems) or pagelets (on Alpha systems), for the specified user.

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

# System Service Descriptions

## \$GETUAI

### 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 UAI\$ \_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 UAI\$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.



## System Service Descriptions

### \$GETUAI

SS\$_BADPARAM	The function code is invalid; the item list contains an invalid item code; a buffer descriptor has an invalid length; or the reserved parameter has a nonzero value.
SS\$_NOGRPPRV	The user does not have the privileges required to examine the authorization information for other members of the UIC group.
SS\$_NOSYSPRV	The user does not have the privileges required to examine the authorization information associated with the user or for users outside of the user's UIC group.
RMS\$_RSZ	The UAF record is smaller than required; the caller's SYSUAF is probably corrupt.

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



---

# Index

## A

---

Aborting a transaction, SYS1-3  
\$ABORT\_TRANS system service, SYS1-3  
\$ABORT\_TRANSW system service, SYS1-7  
Absolute time  
    as input to \$BINTIM, SYS1-67  
    as input to \$BINUTC, SYS1-70  
    converting to numeric, SYS2-174  
Access  
    checking, SYS1-88  
Access modes  
    changing to executive, SYS1-117, SYS1-119  
    changing to kernel, SYS1-121, SYS1-124  
Access protection  
    checking, SYS1-103  
Accounting messages  
    format of, SYS1-215  
ACLs (access control lists)  
    formatting, SYS1-466  
\$ACQUIRE\_GALAXY\_LOCK system service,  
    SYS1-8  
Adding holder records to rights database,  
    SYS1-10  
Adding identifiers to rights database, SYS1-13  
Address space  
    creating virtual, SYS1-227  
\$ADD HOLDER system service, SYS1-10  
\$ADD\_IDENT system service, SYS1-13  
\$ADD\_PROXY system service, SYS1-16  
\$ADJSTK system service, SYS1-20  
\$ADJWSL system service, SYS1-22  
Alignment fault data  
    getting for system process, SYS2-25  
    getting for user image, SYS2-4  
Alignment fault reporting  
    disabling for user image, SYS2-497  
    disabling for user process, SYS2-181  
    enabling for user process, SYS2-182  
    initializing for system process, SYS2-89  
    starting for user image, SYS2-487  
Allocating  
    devices, SYS1-24  
Allocation classes, SYS1-489  
\$ALLOC system service, SYS1-24  
Arithmetic exceptions  
    getting information about, SYS2-6

\$ASCEFC system service, SYS1-27  
ASCII character set  
    converting strings to binary, SYS1-66  
    converting strings to UTC, SYS1-69  
ASCII output  
    formatting character string, SYS1-426  
\$ASCTIM system service, SYS1-31  
\$ASCTOID system service, SYS1-34  
\$ASCUTC system service, SYS1-37  
Assigning  
    an I/O channel, SYS1-40  
\$ASSIGN system service, SYS1-40  
ASTLM (AST limit) quota  
    effect of canceling wakeup on, SYS1-86  
ASTs (asynchronous system traps)  
    declaring, SYS1-302  
    disabling, SYS2-326  
    enabling, SYS2-326  
    setting for power recovery, SYS2-342  
    setting timer for, SYS2-339  
Asynchronous system traps  
    See ASTs  
Audit event messages  
    converting, SYS1-479  
Auditing events, SYS1-45, SYS1-63  
\$AUDIT\_EVENT system service, SYS1-45  
\$AUDIT\_EVENTW system service, SYS1-63  
Automatic unshelving  
    controlling, SYS2-361  
    determining, SYS1-525  
\$AVOID\_PREEMPT system service, SYS1-64

## B

---

Binary time  
    converting to ASCII string, SYS1-31  
    converting to numeric time, SYS2-174,  
        SYS2-176  
Binary values  
    converting to ASCII string, SYS1-426  
\$BINTIM system service, SYS1-66  
\$BINUTC system service, SYS1-69  
64-bit virtual addressing  
    system services support, vii  
\$BRKTHRU system service, SYS1-72  
\$BRKTHRUW system service, SYS1-80

Buffer objects  
  creating, SYS1-142  
  deleting, SYS1-310  
BYTLM quota  
  using with \$GETJPI buffers, SYS2-235

## C

---

Caching attribute  
  write-back, SYS1-491  
  write-through, SYS1-491  
Call frames  
  removing from stack, SYS2-527  
Call stacks  
  unwinding, SYS2-30  
Canceling  
  exit handlers, SYS1-83  
  I/O requests, SYS1-81  
  timer requests, SYS1-84  
  wakeup requests, SYS1-86  
\$CANCEL system service, SYS1-81  
\$SCANEXH system service, SYS1-83  
\$CANTIM system service, SYS1-84  
\$SCANWAK system service, SYS1-86  
Change-mode handlers  
  declaring, SYS1-304  
Channels  
  canceling I/O, SYS1-81  
\$CHECK\_ACCESS system service, SYS1-88  
\$CHECK\_FEN system service (Alpha only),  
  SYS1-96  
\$CHECK\_PRIVILEGE system service, SYS1-97  
\$CHECK\_PRIVILEGEW system service,  
  SYS1-102  
\$CHKPRO system service, SYS1-103  
Class scheduler  
  processes, SYS2-324  
Clearing an event flag, SYS1-116  
\$CLEAR\_SYSTEM\_EVENT system service,  
  SYS1-111  
\$CLRAST system service, SYS1-113  
\$CLRCLUEVT system service, SYS1-114  
\$CLREF system service, SYS1-116  
Cluster events  
  clearing request for notification of, SYS1-114  
  requesting notification of, SYS2-327  
\$CMEXEC system service, SYS1-117  
\$CMEXEC\_64 system service, SYS1-119  
\$CMKRNL system service, SYS1-121  
\$CMKRNL\_64 system service, SYS1-124  
Common event flag clusters  
  associating, SYS1-27  
  disassociating, SYS1-296  
Compatibility mode handlers  
  declaring, SYS1-304  
Control region  
  adding page to, SYS1-419  
  deleting page from, SYS1-332

Converting  
  ASCII string to binary time, SYS1-66  
  ASCII string to UTC format, SYS1-69  
  audit event message, SYS1-479  
  binary time to ASCII string, SYS1-31  
  binary time to numeric time, SYS2-174  
  64-bit system time to UTC time, SYS2-506  
  UTC format to ASCII, SYS1-37  
  UTC time to numeric time, SYS2-176  
CPU affinity set  
  modifying, SYS2-221  
CPU user capability set  
  modifying, SYS1-126  
\$CPU\_CAPABILITIES system service (Alpha  
  only), SYS1-126  
\$CPU\_TRANSITION system service, SYS1-130  
\$CREATE\_BUFOBJ\_64 system service, SYS1-142  
  description, SYS1-144  
\$CREATE\_GALAXY\_LOCK system service,  
  SYS1-136  
\$CREATE\_GALAXY\_LOCK\_TABLE system  
  service, SYS1-139  
\$CREATE\_GDZRO system service (Alpha only),  
  SYS1-146  
  description, SYS1-150  
\$CREATE\_GFILE system service (Alpha only),  
  SYS1-154  
  description, SYS1-157  
\$CREATE\_GPFILE system service (Alpha only),  
  SYS1-160  
  description, SYS1-162  
\$CREATE\_GPFN system service (Alpha only),  
  SYS1-164  
  description, SYS1-167  
\$CREATE\_RDB system service, SYS1-169  
\$CREATE\_REGION\_64 system service, SYS1-171  
  description, SYS1-174  
\$CREATE\_USER\_PROFILE system service,  
  SYS1-179  
Creating  
  disk file sections, SYS1-232  
  logical names, SYS1-185  
  logical name tables, SYS1-193  
  mailboxes, SYS1-199  
  processes, SYS1-206  
  rights databases, SYS1-169  
  user profiles, SYS1-179  
  virtual address space, SYS1-224  
\$CRELNM system service, SYS1-185  
\$CRELNT system service, SYS1-193  
\$CREMBX system service, SYS1-199  
\$CREPRC system service, SYS1-206  
\$CRETVA system service, SYS1-224  
  See also \$EXPREG system service  
\$CRETVA\_64 system service, SYS1-227  
  description, SYS1-229

SCRMPSC system service, SYS1-232  
SCRMPSC\_FILE\_64 system service, SYS1-245  
    description, SYS1-248  
SCRMPSC\_GDZRO\_64 system service, SYS1-251  
    description, SYS1-257  
SCRMPSC\_GFILE\_64 system service, SYS1-264  
    description, SYS1-269  
SCRMPSC\_GPFILE\_64 system service, SYS1-272  
    description, SYS1-276  
SCRMPSC\_GPFN\_64 system service, SYS1-279  
    description, SYS1-283  
SCRMPSC\_PFN\_64 system service, SYS1-286  
    description, SYS1-288  
SCVT\_FILENAME system service, SYS1-291

## D

---

SDACEFC system service, SYS1-296  
SDALLOC system service, SYS1-298  
SDASSGN system service, SYS1-300  
SDCLAST system service, SYS1-302  
SDCLCMH system service, SYS1-304  
SDCLEXH system service, SYS1-307  
Deallocating devices, SYS1-298  
Deassigning an I/O channel, SYS1-300  
DECdns (Digital Distributed Name Service)  
    converting names, SYS1-376, SYS1-377,  
        SYS1-378, SYS1-380  
    creating objects, SYS1-371  
    deleting objects, SYS1-372  
    enumerating objects, SYS1-374  
Declaring an AST (asynchronous system trap),  
    SYS1-302  
Default directories  
    setting, SYS2-330  
Default file protection  
    setting, SYS2-332  
Default form, SYS2-438  
SDELETE\_BUFOBJ system service (Alpha only),  
    SYS1-310  
    description, SYS1-310  
SDELETE\_GALAXY\_LOCK system service,  
    SYS1-312  
SDELETE\_GALAXY\_LOCK\_TABLE system  
    service, SYS1-314  
SDELETE\_INTRUSION system service,  
    SYS1-315  
SDELETE\_PROXY system service, SYS1-318  
SDELETE\_REGION\_64 system service,  
    SYS1-321  
    description, SYS1-322  
Deleting  
    DECdns objects, SYS1-372  
    event flag clusters, SYS1-365  
    global sections, SYS1-351  
    intrusion records, SYS1-315  
    logical names, SYS1-324  
    mailboxes, SYS1-327

## Deleting (cont'd)

    processes, SYS1-329  
    proxies, SYS1-318  
    virtual address space, SYS1-332  
SDELLNM system service, SYS1-324  
SDELMBX system service, SYS1-327  
SDELPRC system service, SYS1-329  
Delta time  
    as input to SBINTIM, SYS1-67  
    converting to numeric, SYS2-174  
SDELTVA system service, SYS1-332  
SDELTVA\_64 system service, SYS1-334  
    description, SYS1-336  
SDEQ system service, SYS1-338  
Dequeuing lock requests, SYS1-338  
Detached processes  
    creating, SYS1-219  
Device characteristics  
    set and wait, SYS2-385  
    setting, SYS2-380  
Device paths  
    scan for, SYS1-343  
Devices  
    allocating, SYS1-24  
    deallocating, SYS1-298  
    dual-pathed, SYS1-489  
    getting information asynchronously, SYS1-485  
    getting information synchronously, SYS1-505  
    lock name, SYS1-493  
    scanning of across the cluster, SYS1-347  
    served, SYS1-497  
SDEVICE\_PATH\_SCAN system service,  
    SYS1-343  
SDEVICE\_SCAN system service, SYS1-347  
SDGBLSC system service, SYS1-351  
Disk file sections  
    creating, SYS1-232  
    mapping, SYS1-232  
Disks  
    initializing within a program, SYS2-92  
Dismounting a volume, SYS1-355  
SDISMOU system service, SYS1-355  
SDISPLAY\_PROXY system service, SYS1-359  
SDLCEFC system service, SYS1-365  
SDNS system service (VAX only), SYS1-367  
SDNSW system service (VAX only), SYS1-394

## E

---

SEND\_TRANS system service, SYS1-395  
SEND\_TRANSW system service, SYS1-400  
SENQ system service, SYS1-401  
SENQW system service, SYS1-414  
Equivalence names  
    specifying, SYS1-185  
SERAPAT system service, SYS1-415

- Error logger
  - sending message to, SYS2-414
- Event flag clusters
  - associating with a process, SYS1-27
  - deleting, SYS1-365
  - disassociating, SYS1-296
  - getting current status, SYS2-264
- Event flags, SYS1-367
  - clearing, SYS1-116
  - getting current status, SYS2-264
  - setting, SYS2-334
  - waiting for entire set of, SYS2-550
  - waiting for one of set, SYS2-552
  - waiting for setting of, SYS2-546
- Events
  - auditing, SYS1-45, SYS1-63
- Exception vectors
  - setting, SYS2-335
- Executive mode
  - changing to, SYS1-117, SYS1-119
- Exit handlers
  - canceling, SYS1-83
  - control block, SYS1-307
    - deleting, SYS1-83
  - declaring, SYS1-307
- Exits
  - forcing, SYS1-463
- \$EXIT system service, SYS1-418
  - issuing for specified process, SYS1-463
- Expanding program/control region, SYS1-419
- \$EXPREG system service, SYS1-419
- \$EXPREG\_64 system service, SYS1-422
  - description, SYS1-424

## F

---

- \$FAOL system service, SYS1-426
- \$FAOL\_64 system service, SYS1-447
- \$FAO system service, SYS1-426
- \$FAO system service directives
  - format of, SYS1-428
  - table of, SYS1-430
- Files
  - getting information asynchronously, SYS1-551
  - getting information synchronously, SYS1-594
- \$FILESCAN system service, SYS1-449
- File specifications
  - parsing components of, SYS1-449
  - searching string for, SYS1-449
- \$FIND\_HELD system service, SYS1-455
- \$FIND HOLDER system service, SYS1-458
- \$FINISH\_RDB system service, SYS1-461
- Floating point
  - checking, SYS1-96
- \$FORCEX system service, SYS1-463
  - See also \$DELPRC and \$EXIT

- Forcing an exit, SYS1-463
- Formatting
  - ACL entry, SYS1-466
  - security audit messages, SYS1-479
- \$FORMAT\_ACL system service, SYS1-466
- \$FORMAT\_AUDIT system service, SYS1-479
- Forms
  - getting information asynchronously, SYS1-551
  - getting information synchronously, SYS1-594
- \$FREE\_USER\_CAPABILITY system service
  - (Alpha only), SYS1-483
- Full names
  - converting to string, SYS1-376

## G

---

- \$GETDVI system service, SYS1-485
- \$GETDVIW system service, SYS1-505
- \$GETENV system service, SYS1-506
- \$GETJPI system service, SYS1-508
- \$GETJPIW system service, SYS1-533
- \$GETLKI system service, SYS1-534
- \$GETLKIW system service, SYS1-546
- \$GETMSG system service, SYS1-547
- \$GETQUI system service, SYS1-551
- \$GETQUIW system service, SYS1-594
- \$GETRMI system service, SYS1-595
- \$GETSYI system service, SYS1-602
- \$GETSYIW system service, SYS1-626
- \$GETTIM system service, SYS1-627
- \$GETUAI system service, SYS1-628
- \$GETUTC system service, SYS2-3
- \$GET\_ALIGN\_FAULT\_DATA system service
  - (Alpha only), SYS2-4
- \$GET\_ARITH\_EXCEPTION system service (Alpha only), SYS2-6
- \$GET\_GALAXY\_LOCK\_INFO system service, SYS2-8
- \$GET\_GALAXY\_LOCK\_SIZE system service, SYS2-10
- \$GET\_REGION\_INFO system service (Alpha only), SYS2-12
  - description, SYS2-15
- \$GET\_SECURITY system service, SYS2-17
- \$GET\_SYS\_ALIGN\_FAULT\_DATA system service
  - (Alpha only), SYS2-25
- \$GET\_USER\_CAPABILITY system service (Alpha only), SYS2-27
  - description, SYS2-28
- Global demand-zero section
  - creating, SYS1-146
  - creating and mapping to, SYS1-251
- Global disk file section
  - creating, SYS1-154
  - creating and mapping, SYS1-264
- Global page file
  - creating, SYS1-160

- Global page file section
  - creating and mapping, SYS1-272
- Global page frame section
  - creating and mapping, SYS1-279
  - mapping, SYS2-144
- Global section file
  - updating on disk (asynchronously), SYS2-535
  - updating on disk (synchronously), SYS2-540
- Global sections
  - creating, SYS1-232
  - deleting, SYS1-351
  - mapping, SYS1-232, SYS2-128, SYS2-134
- \$GOTO\_UNWIND system service (Alpha only), SYS2-30
- \$GRANTID system service, SYS2-32

## H

---

- \$HASH\_PASSWORD system service, SYS2-36
- \$HIBER system service, SYS2-39
  - See also \$WAKE
- Holder records
  - adding to rights database, SYS1-10
  - modifying in rights database, SYS2-150
  - removing from rights database, SYS2-294
- Holders of an identifier
  - finding, SYS1-458
- Hosts
  - checking availability of, SYS1-489

## I

---

- I/O channels
  - assigning, SYS1-40
  - deassigning, SYS1-300
- I/O devices
  - getting information asynchronously, SYS1-485
  - getting information synchronously, SYS1-505
- I/O requests
  - canceling, SYS1-81
  - queuing asynchronously, SYS2-257
  - queuing synchronously, SYS2-263
- \$ICC\_ACCEPT system service, SYS2-41
- \$ICC\_CLOSE\_ASSOC system service, SYS2-44
- \$ICC\_CONNECT system service, SYS2-46
- \$ICC\_CONNECTW system service, SYS2-51
- \$ICC\_DISCONNECT system service, SYS2-52
- \$ICC\_DISCONNECTW system service, SYS2-55
- \$ICC\_OPEN\_ASSOC system service, SYS2-56
- \$ICC\_RECEIVE system service, SYS2-63
- \$ICC\_RECEIVEW system service, SYS2-66
- \$ICC\_REJECT system service, SYS2-67
- \$ICC\_REPLY system service, SYS2-69
- \$ICC\_REPLYW system service, SYS2-72
- \$ICC\_TRANSCEIVE system service, SYS2-73
- \$ICC\_TRANSCEIVEW system service, SYS2-77

- \$ICC\_TRANSMIT system service, SYS2-78
- \$ICC\_TRANSMITW system service, SYS2-81
- Identifier names
  - translating to identifier, SYS1-34
- Identifiers
  - adding record to rights list, SYS2-32
  - finding, SYS1-455
  - modifying in rights database, SYS2-153
  - removing from rights database, SYS2-296
  - revoking from process, SYS2-305
  - translating value to identifier name, SYS2-82
- \$IDTOASC system service, SYS2-82
- IEEE floating-point control register
  - setting, SYS2-86
- \$IEEE\_SET\_FP\_CONTROL system service (Alpha only), SYS2-86
- Image exit, SYS1-418
- Image rundown
  - forcing, SYS1-463
- Initializing
  - a volume within a program, SYS2-92
- \$INIT\_SYS\_ALIGN\_FAULT\_REPORT system service (Alpha only), SYS2-89
- \$INIT\_VOL system service, SYS2-92
- Intrusion records
  - deleting, SYS1-315
- Intrusions
  - returning information about, SYS2-407
  - scanning for, SYS2-313
- \$IO\_CLEANUP system service (Alpha only), SYS2-105
  - description, SYS2-105
- \$IO\_FASTPATH system service (Alpha only), SYS2-106
- \$IO\_FASTPATHW system service (Alpha only), SYS2-108
- \$IO\_PERFORM system service (Alpha only), SYS2-109
  - description, SYS2-110
- \$IO\_PERFORMW system service (Alpha only), SYS2-112
- \$IO\_SETUP system service (Alpha only), SYS2-113
  - description, SYS2-114

## J

---

- Job controllers
  - asynchronous, SYS2-415
  - synchronous, SYS2-472
- Jobs
  - getting information asynchronously, SYS1-508, SYS1-551
  - getting information synchronously, SYS1-533, SYS1-594

## K

---

Kernel mode  
changing to, SYS1-121, SYS1-124

## L

---

\$LCKPAG system service, SYS2-116  
\$LCKPAG\_64 system service (Alpha only),  
SYS2-119  
description, SYS2-120  
\$LKWSET system service, SYS2-122  
\$LKWSET\_64 system service (Alpha only),  
SYS2-125  
description, SYS2-126

Lock database  
in an OpenVMS Cluster, SYS1-543

Lock requests  
dequeuing, SYS1-338  
queuing asynchronously, SYS1-401  
queuing synchronously, SYS1-414

Locks  
getting information asynchronously, SYS1-534  
getting information synchronously, SYS1-546

Logical names  
creating, SYS1-185  
deleting, SYS1-324  
getting information about, SYS2-508  
translating, SYS2-508

Logical name tables  
creating, SYS1-193  
deleting, SYS1-324

## M

---

Magnetic tapes  
initializing within a program, SYS2-92

Mailboxes  
assigning channel to, SYS1-199  
creating, SYS1-199  
deleting permanent, SYS1-203, SYS1-327  
deleting temporary, SYS1-203

Mapping disk file sections, SYS1-232  
MAXBUF system parameter, SYS1-73, SYS1-78,  
SYS1-79

Memory  
locking page into, SYS2-116  
unlocking page from, SYS2-517  
unlocking pages from, SYS2-519

Messages  
converting security message from binary to  
ASCII, SYS1-479  
filtering sensitive information, SYS1-481  
formatting and outputting, SYS2-250  
obtaining text of, SYS1-547  
sending to error logger, SYS2-414

## Messages (cont'd)

sending to one or more terminals, SYS1-72,  
SYS1-80  
sending to operator, SYS2-473  
writing to terminal, SYS1-72, SYS1-80  
Message symbols, SYS2-255  
\$MGBLSC system service, SYS2-128  
\$MGBLSC\_64 system service (Alpha only),  
SYS2-134  
description, SYS2-139  
\$MGBLSC\_GPFN\_64 system service, SYS2-144  
description, SYS2-148  
\$MOD\_HOLDER system service, SYS2-150  
\$MOD\_IDENT system service, SYS2-153  
\$MOUNT system service, SYS2-157  
\$MTACCESS system service, SYS2-171

## N

---

Notification ASTs  
testing functionality of, SYS2-515  
\$NUMTIM system service, SYS2-174  
\$NUMUTC system service, SYS2-176

## O

---

Obsolete system services, A-1  
Opaque names  
converting to string, SYS1-376  
Operators  
sending messages to, SYS2-473

## P

---

Page frame sections  
creating, SYS1-164  
Page protection  
setting, SYS2-352  
Pages  
locking into memory, SYS2-116  
locking into working set, SYS2-122  
removing from working set, SYS2-246  
setting protection, SYS2-349  
unlocking from memory, SYS2-517  
unlocking from working set, SYS2-522  
\$PARSE\_ACL system service, SYS2-178  
Passwords  
returning hash value, SYS2-36  
\$PERM\_DIS\_ALIGN\_FAULT\_REPORT system  
service (Alpha only), SYS2-181  
\$PERM\_REPORT\_ALIGN\_FAULT system service  
(Alpha only), SYS2-182  
Persona  
assuming, SYS2-184, SYS2-186  
cloning, SYS2-188  
creating, SYS2-190, SYS2-192  
creating an extension, SYS2-197  
delegating to a server process, SYS2-200



## Persona (cont'd)

- deleting, SYS2-202
- deleting an extension, SYS2-203
- finding with characteristics, SYS2-207
- querying for data, SYS2-214
- reserving a slot in the persona table, SYS2-219
- setting attribute values, SYS2-211
- translating an extension name to an ID, SYS2-205
- \$PERSONA\_ASSUME system service, SYS2-184, SYS2-186
- \$PERSONA\_CLONE system service, SYS2-188
- \$PERSONA\_CREATE system service, SYS2-190, SYS2-192
- \$PERSONA\_CREATE\_EXTENSION system service (Alpha only), SYS2-197
- \$PERSONA\_DELEGATE system service (Alpha only), SYS2-200
- \$PERSONA\_DELETE system service, SYS2-202
- \$PERSONA\_DELETE\_EXTENSION system service (Alpha only), SYS2-203
- \$PERSONA\_EXTENSION\_LOOKUP system service (Alpha only), SYS2-205
- \$PERSONA\_FIND system service (Alpha only), SYS2-207
- \$PERSONA\_MODIFY system service (Alpha only), SYS2-211
- \$PERSONA\_QUERY system service (Alpha only), SYS2-214
- \$PERSONA\_RESERVE system service (Alpha only), SYS2-219
- PID numbers
  - using with \$GETJPI to return information about a process, SYS1-508
- Power recovery
  - setting AST for, SYS2-342
- Priority setting, SYS2-344
- Private disk file section
  - create and map, SYS1-245
- Private page frame
  - create and map, SYS1-286
- Privileges
  - checking, SYS1-97
  - setting for process, SYS2-355
- Processes
  - affecting scheduling of, SYS2-321
  - avoiding preemption, SYS1-64, SYS2-379
  - creating, SYS1-206
  - deleting, SYS1-329
  - getting information asynchronously, SYS1-508
  - getting information synchronously, SYS1-533
  - hibernating, SYS2-39
  - locating a subset of, SYS2-231
  - rescheduling, SYS2-298
  - resuming after suspension, SYS2-303
  - scanning, SYS2-231
  - scheduling wakeup for, SYS2-318
  - setting default protection for, SYS2-332

## Processes (cont'd)

- setting name of, SYS2-348
- setting priority of, SYS2-344
- setting privileges, SYS2-355
- setting stack limits, SYS2-363
- setting swap mode for, SYS2-365
- suspending, SYS2-501
- waiting for entire set of event flags, SYS2-550
- waiting for event flag to be set, SYS2-546
- waiting for one of set of event flags, SYS2-552
- waking, SYS2-548
- writing messages to, SYS2-250
- Process identification numbers
  - See PID numbers
- Process names
  - setting, SYS2-348
  - specifying processes by, SYS2-238
  - specifying processes with node name, SYS2-237
- Process scan, SYS2-231
- Process scheduling
  - affecting, SYS2-321
- Process user capability set
  - modifying, SYS2-226
- \$PROCESS\_AFFINITY system service (Alpha only), SYS2-221
- \$PROCESS\_CAPABILITIES system service (Alpha only), SYS2-226
- \$PROCESS\_SCAN system service, SYS2-231
- Program regions
  - adding page to, SYS1-419
  - deleting page from, SYS1-332
- Protection
  - of queues, SYS2-464
  - setting for page, SYS2-349
- Proxies
  - adding, SYS1-16
  - deleting, SYS1-318
  - displaying, SYS1-359
  - modifying, SYS1-16, SYS1-318
  - verifying, SYS2-541
- \$PURGE\_WS system service (Alpha only), SYS2-248
  - description, SYS2-248
- \$PURGWS system service, SYS2-246
  - See also \$ADJWSL
- \$PUTMSG system service, SYS2-250

## Q

- \$QIO system service, SYS2-257
- \$QIOW system service, SYS2-263
- Queues
  - creating and managing asynchronously, SYS2-415
  - creating and managing synchronously, SYS2-472
  - getting information asynchronously, SYS1-551

## Queues (cont'd)

- getting information synchronously, SYS1-594
- protection, SYS2-464
- types of, SYS2-462

## Quotas

- resource wait mode, SYS2-359
- SS\$\_EXQUOTA condition value, SYS2-360

## R

---

### RADs support

- See Resource Affinity Domains support

\$READDEF system service, SYS2-264

### Regions

- creating virtual, SYS1-171
- deleting, SYS1-321
- \$REGISTRY system service, SYS2-266
- \$REGISTRYW system service, SYS2-290
- \$RELEASE\_GALAXY\_LOCK system service, SYS2-291
- \$RELEASE\_VP system service (VAX only), SYS2-293
- \$REM HOLDER system service, SYS2-294
- \$REM\_IDENT system service, SYS2-296
- \$RESCHED system service, SYS2-298
- Resource Affinity Domains (RADs) support, SYS1-149, SYS1-217, SYS1-257, SYS1-517, SYS1-620, SYS2-390

### Resource wait mode

- setting, SYS2-359

\$RESTORE\_VP\_EXCEPTION system service (VAX only), SYS2-299

\$RESTORE\_VP\_STATE system service (VAX only), SYS2-301

\$RESUME system service, SYS2-303

\$REVOKID system service, SYS2-305

### Rights database context

- terminating, SYS1-461

### Rights databases

- creating, SYS1-169

\$RMSRUNDWN system service, SYS2-309

## S

---

\$SAVE\_VP\_EXCEPTION system service (VAX only), SYS2-311

### Scanning

- for devices, SYS1-347
- intrusion database, SYS2-313
- processes, SYS2-231
- \$SCAN\_INTRUSION system service, SYS2-313
- \$SCHDWK system service, SYS2-318
- \$SCHED system service, SYS2-321

### Section files

- updating asynchronously, SYS2-529
- updating synchronously, SYS2-534

## Sections

- creating, SYS1-232
- deleting global, SYS1-351
- mapping, SYS1-232
- writing modifications to disk, SYS2-529, SYS2-534

## Security

- auditing events, SYS1-45, SYS1-63
- checking privileges, SYS1-97, SYS1-102
- converting message from binary to ASCII, SYS1-479
- filtering sensitive message information, SYS1-479
- getting erase patterns, SYS1-415
- hashing passwords, SYS2-36
- modifying characteristics of an object, SYS2-397
- retrieving information about objects, SYS2-17

## Security characteristics

- modifying for an object, SYS2-397
  - retrieving for an object, SYS2-17
- Sending a message to one or more terminals, SYS1-72, SYS1-80

\$SETAST system service, SYS2-326

\$SETCLUEVT system service, SYS2-327

\$SETDDIR system service, SYS2-330

\$SETDFPROT system service, SYS2-332

\$SETEF system service, SYS2-334

\$SETEXV system service, SYS2-335

\$SETIME system service, SYS2-337

\$SETIMR system service, SYS2-339

\$SETPRA system service, SYS2-342

\$SETPRI system service, SYS2-344

\$SETPRN system service, SYS2-348

\$SETPRT system service, SYS2-349

\$SETPRT\_64 system service, SYS2-352

- description, SYS2-354

\$SETPRV system service, SYS2-355

\$SETRWM system service, SYS2-359

\$SETSHLV system service, SYS2-361

\$SETSTK system service, SYS2-363

\$SETSWM system service, SYS2-365

Setting the resource wait mode, SYS2-359

\$SETUAI system service, SYS2-367

\$SETUP\_AVOID\_PREEMPT system service, SYS2-379

\$SET\_DEVICE system service, SYS2-380

\$SET\_DEVICEW system service, SYS2-385

\$SET\_IMPLICIT\_AFFINITY (Alpha only), SYS2-386

\$SET\_PROCESS\_PROPERTIESW system service, SYS2-390

\$SET\_RESOURCE\_DOMAIN system service, SYS2-392

\$SET\_SECURITY system service, SYS2-397

SSET\_SYSTEM\_EVENT system service, SYS2-404

Shelving  
See Automatic unshelving

SSHOW\_INTRUSION system service, SYS2-407

SSIGNAL\_ARRAY\_64 system service, SYS2-412

Simple names  
converting to opaque, SYS1-378

SSNDERR system service, SYS2-414

SSNDJBC system service, SYS2-415

SSNDJBCW system service, SYS2-472

SSNDOPR system service, SYS2-473

Stack limit  
changing size of, SYS2-363

Stack pointer  
adjusting, SYS1-20

SSTART\_ALIGN\_FAULT\_REPORT system service (Alpha only), SYS2-487

SSTART\_TRANS system service, SYS2-491

SSTART\_TRANSW system service, SYS2-496

SSTOP\_ALIGN\_FAULT\_REPORT system service (Alpha only), SYS2-497

SSTOP\_SYS\_ALIGN\_FAULT\_REPORT system service (Alpha only), SYS2-498

Strings  
converting, SYS1-291  
formatting output, SYS1-426  
searching for file specification in, SYS1-449

Subprocesses  
creating, SYS1-219

SSUBSYSTEM system service, SYS2-499

SSUSPND system service, SYS2-501

SSYNCH system service, SYS2-504

SYSSSYSTEM:LOGINOUT.EXE file  
using as image to create new processes, SYS1-206, SYS1-219

System alignment fault reporting  
disabling for user image, SYS2-498

Systems  
getting information asynchronously, SYS1-602  
getting information synchronously, SYS1-626

System services  
Abort Transaction, SYS1-3, SYS1-7  
Accept (ICC), SYS2-41  
Acquire GALAXY Lock, SYS1-8  
Add Holder Record to Rights Database, SYS1-10  
Add Identifier to Rights Database, SYS1-13  
Add Proxy, SYS1-16  
Adjust Outer Mode Stack Pointer, SYS1-20  
Adjust Working Set Limit, SYS1-22  
Affect Process Scheduling, SYS2-321  
Allocate Device, SYS1-24  
Assign I/O Channel, SYS1-40  
Associate Common Event Flag Cluster, SYS1-27  
Assume Persona, SYS2-184, SYS2-186  
Audit Event, SYS1-45, SYS1-63

## System services (cont'd)

Avoid Process Preemption, SYS1-64

Breakthrough, SYS1-72, SYS1-80

Cancel Exit Handler, SYS1-83

Cancel I/O on Channel, SYS1-81

Cancel Timer, SYS1-84

Cancel Wakeup, SYS1-86

Change to Executive Mode, SYS1-117  
with quadword argument list, SYS1-119

Change to Kernel Mode, SYS1-121  
with quadword argument list, SYS1-124

Check Access, SYS1-88

Check Access Protection, SYS1-103

Check Floating Point (Alpha only), SYS1-96

checking completion status of, SYS2-504

Check Privilege, SYS1-97, SYS1-102

Clean Up Fast I/O, SYS2-105

Clear Cluster Event, SYS1-114

Clear Event Flag, SYS1-116

Clear System Event, SYS1-111

Clone Persona, SYS2-188

Close association (ICC), SYS2-44

\$CLRAST, SYS1-113

Connect (ICC), SYS2-46, SYS2-51

Control Fast Path Devices, SYS2-106, SYS2-108

Convert ASCII String to Binary Time, SYS1-66

Convert ASCII String to UTC Binary Time, SYS1-69

Convert Binary Time to ASCII String, SYS1-31

Convert Binary Time to Numeric Time, SYS2-174

Convert UTC Time to Numeric Components, SYS2-176

Convert UTC to ASCII, SYS1-37

CPU Transition, SYS1-130

Create and Map a Global Disk File Section, SYS1-264

Create and Map Global Page File Section, SYS1-272

Create and Map Global Page Frame Section, SYS1-279

Create and Map Private Disk File Section, SYS1-245

Create and Map Private Page Frame Section, SYS1-286

Create and Map Section, SYS1-232

Create and Map to Global Demand-Zero Section, SYS1-251

Create Buffer Object, SYS1-142

Create Galaxy Lock, SYS1-136

Create Galaxy Lock Table, SYS1-139

Create Logical Name, SYS1-185

Create Logical Name Table, SYS1-193

Create Mailbox and Assign Channel, SYS1-199

## System services (cont'd)

Create Permanent Global Demand-Zero Section, SYS1-146  
Create Permanent Global Disk File Section, SYS1-154  
Create Permanent Global Page File Section, SYS1-160  
Create Permanent Global Page Frame Section, SYS1-164  
Create Persona, SYS2-190, SYS2-192  
create Persona Extension, SYS2-197  
Create Process, SYS1-206  
Create Rights Database, SYS1-169  
Create User Profile, SYS1-179  
Create Virtual Address Space, SYS1-224, SYS1-227  
Create Virtual Region, SYS1-171  
SCVT\_FILENAME, SYS1-291  
Deallocate Device, SYS1-298  
Deassign I/O Channel, SYS1-300  
Declare AST, SYS1-302  
Declare Change Mode or Compatibility Mode Handler, SYS1-304  
Declare Exit Handler, SYS1-307  
Delegate Persona to a Server Process, SYS2-200  
Delete a Virtual Region, SYS1-321  
Delete Buffer Object, SYS1-310  
Delete Common Event Flag Cluster, SYS1-365  
Delete Galaxy Lock, SYS1-312  
Delete Galaxy Lock Table, SYS1-314  
Delete Global Section, SYS1-351  
Delete Intrusion Records, SYS1-315  
Delete Logical Name, SYS1-324  
Delete Mailbox, SYS1-327  
Delete or Modify Proxy, SYS1-318  
Delete Persona, SYS2-202  
delete Persona Extension, SYS2-203  
Delete Process, SYS1-329  
Delete Virtual Address Space, SYS1-332, SYS1-334  
Dequeue Lock Request, SYS1-338  
Disable Alignment Fault Reporting (Alpha only), SYS2-181  
Disassociate Common Event Flag Cluster, SYS1-296  
Disconnect (ICC), SYS2-52, SYS2-55  
Dismount Volume, SYS1-355  
Display Proxy Information, SYS1-359  
Distributed Name Service (DNS) Clerk (VAX only), SYS1-367, SYS1-394  
End Transaction, SYS1-395, SYS1-400  
Enqueue Lock Request, SYS1-401, SYS1-414  
Exit, SYS1-418  
Expand Program/Control Region, SYS1-419  
Expand Virtual Address Space, SYS1-422  
Find Holder of Identifier, SYS1-458  
Find Identifiers Held by User, SYS1-455

## System services (cont'd)

Find Persona, SYS2-207  
Force Exit, SYS1-463  
Format Access Control List Entry, SYS1-466  
Format Security Audit Event Message, SYS1-479  
Formatted ASCII Output Services, SYS1-426  
Formatted ASCII Output with List Parameter for 64-Bit Memory, SYS1-447  
Get Alignment Fault Data (Alpha only), SYS2-4  
Get Arithmetic Exception Information (Alpha only), SYS2-6  
Get Device/Volume Information, SYS1-485, SYS1-505  
Get Environment, SYS1-506  
Get Galaxy Lock Information, SYS2-8  
Get Galaxy Lock Size, SYS2-10  
Get Information About a Specified Virtual Region, SYS2-12  
Get Job/Process Information, SYS1-508, SYS1-533  
Get Lock Information, SYS1-534, SYS1-546  
Get Message, SYS1-547  
Get Queue Information, SYS1-551, SYS1-594  
Get Resource Monitor Information, SYS1-595  
Get Security Characteristics, SYS2-17  
Get Security Erase Pattern, SYS1-415  
Get System Alignment Fault Data (Alpha only), SYS2-25  
Get Systemwide Information, SYS1-602, SYS1-626  
Get Time, SYS1-627  
Get User Authorization Information, SYS1-628  
Get UTC Time, SYS2-3  
Grant Identifier to Process, SYS2-32  
Hash Password, SYS2-36  
Hibernate, SYS2-39  
Initialize System Alignment Fault Reporting (Alpha only), SYS2-89  
Initialize Volume, SYS2-92  
Lock Pages in Memory, SYS2-116, SYS2-119  
Lock Pages in Working Set, SYS2-122, SYS2-125  
Magnetic Tape Accessibility, SYS2-171  
Map Global Page Frame Section, SYS2-144  
Map Global Section, SYS2-128  
Map to Global Section, SYS2-134  
Modify CPU User Capabilities, SYS1-126  
Modify Holder Record in Rights Database, SYS2-150  
Modify Identifier in Rights Database, SYS2-153  
Modify Persona Data, SYS2-211  
Modify Process Affinity, SYS2-221  
Modify Process Implicit Affinity, SYS2-386  
Modify Process User Capabilities, SYS2-226  
Mount Volume, SYS2-157

## System services (cont'd)

obsolete, A-1  
Open association (ICC), SYS2-56  
Parse Access Control List Entry, SYS2-178  
Perform Fast I/O, SYS2-109, SYS2-112  
Process Scan, SYS2-231  
Purge Working Set, SYS2-246, SYS2-248  
Put Message, SYS2-250  
Query for Persona Data, SYS2-214  
Queue I/O Request, SYS2-257  
Queue I/O Request and Wait, SYS2-263  
Read Event Flags, SYS2-264  
Receive data (ICC), SYS2-63, SYS2-66  
Registry Database Interface, SYS2-266  
Registry Database Interface and Wait,  
SYS2-290  
Reject (ICC), SYS2-67  
Release a Reserved User Capability, SYS1-483  
Release Galaxy Lock, SYS2-291  
Release Vector Processor (VAX only), SYS2-293  
Remove Holder Record from Rights Database,  
SYS2-294  
Remove Identifier from Rights Database,  
SYS2-296  
Reply (ICC), SYS2-69, SYS2-72  
Report Alignment Fault (Alpha only),  
SYS2-182  
Reschedule Process, SYS2-298  
Reserve a User Capability, SYS2-27  
Reserve Persona Slot, SYS2-219  
Restore Vector Processor Exception State (VAX  
only), SYS2-299  
Restore Vector State (VAX only), SYS2-301  
Resume Process, SYS2-303  
Revoke Identifier from Process, SYS2-305  
RMS Rundown, SYS2-309  
Save Vector Processor Exception State (VAX  
only), SYS2-311  
Scan for Device Paths, SYS1-343  
Scan for Devices, SYS1-347  
Scan Intrusion Database, SYS2-313  
Scan String for File Specification, SYS1-449  
Schedule Wakeup, SYS2-318  
Send Message to Error Logger, SYS2-414  
Send Message to Operator, SYS2-473  
Send to Job Controller, SYS2-415, SYS2-472  
Set AST Enable, SYS2-326  
Set Automatic Unshelving, SYS2-361  
Set Cluster Event, SYS2-327  
Set Default Directory, SYS2-330  
Set Default File Protection, SYS2-332  
Set Device Characteristics, SYS2-380  
Set Device Characteristics and Wait, SYS2-385  
Set Event Flag, SYS2-334  
Set Exception Vector, SYS2-335  
Set IEEE Floating-Point Control Register  
(Alpha only), SYS2-86  
Set Power Recovery AST, SYS2-342

## System services (cont'd)

Set Priority, SYS2-344  
Set Privileges, SYS2-355  
Set Process Name, SYS2-348  
Set Process Swap Mode, SYS2-365  
Set Protection on Pages, SYS2-349, SYS2-352  
Set Resource Domain, SYS2-392  
Set Resource Wait Mode, SYS2-359  
Set Security, SYS2-397  
Sets Simple Value, SYS2-390  
Set Stack Limits, SYS2-363  
Set System Event, SYS2-404  
Set System Time, SYS2-337  
Set Timer, SYS2-339  
Set Up Fast I/O, SYS2-113  
Setup for Process Preemption Avoidance,  
SYS2-379  
Set User Authorization Information, SYS2-367  
Show Intrusion Information, SYS2-407  
Signal Array, SYS2-412  
Start Alignment Fault Reporting (Alpha only),  
SYS2-487  
Start Transaction, SYS2-491, SYS2-496  
Stop Alignment Fault Reporting (Alpha only),  
SYS2-497  
Stop System Alignment Fault Reporting (Alpha  
only), SYS2-498  
Subsystem, SYS2-499  
Suspend Process, SYS2-501  
Synchronize, SYS2-504  
Terminate Rights Database Context, SYS1-461  
Test Cluster Event, SYS2-515  
Time Converter, SYS2-506  
Transceive data (ICC), SYS2-73, SYS2-77  
translate an extension name to an ID,  
SYS2-205  
Translate Identifier Name to Identifier,  
SYS1-34  
Translate Identifier to Identifier Name,  
SYS2-82  
Translate Logical Name, SYS2-508  
Transmit data (ICC), SYS2-78, SYS2-81  
Unlock Pages from Memory, SYS2-517,  
SYS2-519  
Unlock Pages from Working Set, SYS2-522,  
SYS2-524  
Unwind Call Stack, SYS2-527  
Unwind Call Stack (Alpha only), SYS2-30  
Update Global Section File on Disk, SYS2-535,  
SYS2-540  
Update Section File on Disk, SYS2-529,  
SYS2-534  
Verify Proxy, SYS2-541  
Wait for Logical AND of Event Flags,  
SYS2-550  
Wait for Logical OR of Event Flags, SYS2-552  
Wait for Single Event Flag, SYS2-546  
Wake Process from Hibernation, SYS2-548

## System time

See also Time

converting 64-bit time to UTC time, SYS2-506  
setting, SYS2-337

## T

---

### Tapes

initializing within a program, SYS2-92

### Termination messages

format of, SYS1-215

\$TIMCON system service, SYS2-506

### Time

converting 64-bit system format to UTC,  
SYS2-506

converting binary to ASCII string, SYS1-31

converting binary to numeric, SYS2-174

converting UTC to 64-bit system format,  
SYS2-506

converting UTC to ASCII, SYS1-37

converting UTC to numeric components,  
SYS2-176

getting current system, SYS1-627

setting system, SYS2-337

### Timer requests

canceling, SYS1-84

### Timers

setting, SYS2-339

TQELM (timer queue entry limit)

See TQELM process limit

TQELM process limit

effect of canceling timer request, SYS1-85

### Transactions

aborting asynchronously, SYS1-3

aborting synchronously, SYS1-7

default, SYS2-494

ending asynchronously, SYS1-395

ending synchronously, SYS1-400

starting asynchronously, SYS2-491

starting synchronously, SYS2-496

Translating identifier name to identifier, SYS1-34

\$STRNLNM system service, SYS2-508

\$TSTCLUEVT system service, SYS2-515

## U

---

### UAFs (user authorization files)

getting information about, SYS1-628

modifying, SYS2-367

\$ULKPAG system service, SYS2-517

\$ULKPAG\_64 system service, SYS2-519

description, SYS2-520

\$ULWSET system service, SYS2-522

\$ULWSET\_64 system service, SYS2-524

description, SYS2-525

Unlocking pages from memory, SYS2-517,  
SYS2-519

Unlocking pages from the working set, SYS2-522,  
SYS2-524

\$UNWIND system service, SYS2-527

\$UPDSEC system service, SYS2-529

\$UPDSECW system service, SYS2-534

\$UPDSEC\_64 system service, SYS2-535

description, SYS2-538

\$UPDSEC\_64W system service, SYS2-540

### User capabilities

reserving, SYS2-27

unreserving, SYS1-483

### User profiles

creating, SYS1-179

### UTC (Coordinated Universal Time)

converting format to ASCII, SYS1-37

### UTC format

converting to numeric components, SYS2-176

getting, SYS2-3

## V

---

### Vector processors

releasing, SYS2-293

restoring the exception state of, SYS2-299

saving the exception state of, SYS2-311

### Vector state

restoring, SYS2-301

\$VERIFY\_PROXY system service, SYS2-541

### Virtual address space

adding page to, SYS1-224, SYS1-419

creating, SYS1-224

deleting, SYS1-334

deleting page from, SYS1-332

expanding, SYS1-422

### Virtual I/O

canceling requests for, SYS1-81

### Virtual region

getting information about, SYS2-12

### Volumes

dismounting, SYS1-355

getting information asynchronously, SYS1-485

getting information synchronously, SYS1-505

initializing within a program, SYS2-92

mounting, SYS2-157

VTF-7 characters, SYS1-291, SYS1-292,

SYS1-294

## W

---

\$WAITFR system service, SYS2-546

\$WAKE system service, SYS2-548

See also \$HIBER

### Wakeup requests

canceling, SYS1-86

- SWFLAND system service, SYS2-550
- SWFLOR system service, SYS2-552
- Wildcard operations, SYS1-508
- Wildcard searches
  - obtaining information about processes,  
SYS2-231
- Working set
  - purging, SYS2-248
- Working sets
  - adjusting limit, SYS1-22
  - locking page into, SYS2-122
  - purging, SYS2-246
  - unlocking page from, SYS2-522, SYS2-524

