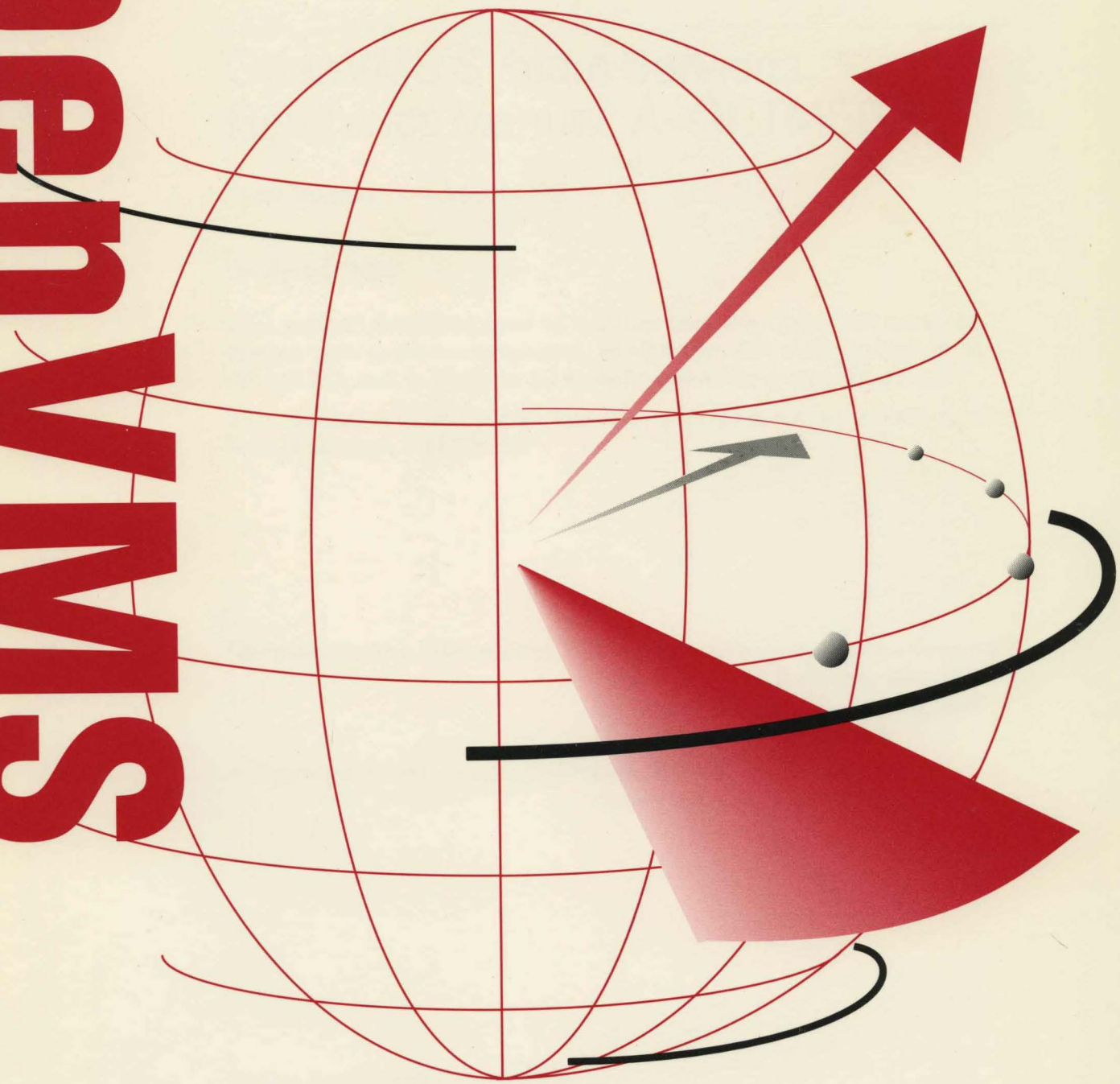
digital

OpenVMS System Services Reference Manual: A-GETMSG

OpenVMS

OpenVMS



OpenVMS System Services Reference Manual: A–GETMSG

Order Number: AA–QSBMA–TE

December 1995

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

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

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

Software Version: OpenVMS Alpha Version 7.0
OpenVMS VAX Version 7.0

Digital Equipment Corporation
Maynard, Massachusetts

December 1995

ZK6243

This document is available on CD–ROM.

Contents

Index

Tables

Preface

Intended Audience

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

System Services Support for OpenVMS Alpha 64-bit Addressing

Alpha

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

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

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

64-Bit System Services Terminology

32-bit system service

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

64-bit friendly interface

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

64-bit system service

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

Use of the _64 Suffix

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

Sign-Extension Checking

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

Document Structure

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

Related Documents

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

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

The following documents might also be useful:

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

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

How To Order Additional Documentation

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

Telephone and Direct Mail Orders

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

ZK–7654A–GE

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

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

Reader's Comments

Digital welcomes your comments on this manual.

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

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

Conventions

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

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

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

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

♦

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

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

The following conventions are also used in this manual:

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

System Service Descriptions

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

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

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

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

Returns

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

Longword condition value. All system services (except $EXIT) return by immediate value a condition value in R0.

$ABORT_TRANS
Abort Transaction

Ends a transaction by aborting it.

Format

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

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$M_SYNC	Set this flag to specify that successful synchronous completion is to be indicated by returning SS$_SYNCH. When SS$_SYNCH is returned, the AST routine is not called, the event flag is not set, and the I/O status block is not filled in.

iosb

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

I/O status block in which the following information is returned:

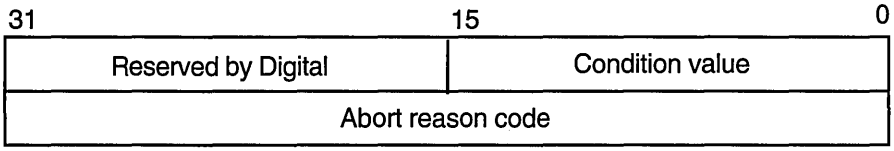
- The completion status of the service, returned as a condition value. See the Condition Values Returned section.
- An abort reason code that gives one reason why the transaction aborted, if the completion status of the service is SS$_NORMAL.

Note that if there are multiple reasons why the transaction aborted, the abort reason code returned in the I/O status block may 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 $ABORT_TRANS gives DDTM$_ABORTED as the reason and the transaction timeout expires at about the same time as the call to $ABORT_TRANS, then either the DDTM$_TIMEOUT or DDTM$_ ABORTED reason code may 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 SYS1–5.

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



ZK–3667A–GE

astadr

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

AST routine that is executed when the service completes, 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 $DDTMMSGDEF macro defines symbolic names for abort reason codes. Those currently defined are shown in Table SYS1–5. The default value for this argument is DDTM$_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).

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, $END_TRANS, $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$_ACCVIO	An argument was not accessible by the caller.
SS$_BADPARAM	The options flags were invalid.
SS$_BADREASON	The abort reason code was invalid.
SS$_CURTIDCHANGE	The **tid** argument was omitted and a call to change the default transaction of the calling process was in progress.
SS$_EXASTLM	The process AST limit (ASTLM) was exceeded.
SS$_ILLEFC	The event flag number was invalid.
SS$_INSFARGS	Not enough arguments were supplied.
SS$_INSFMEM	There was insufficient system dynamic memory for the operation.
SS$_NOCURTID	An attempt was made to abort the default transaction (the **tid** argument was omitted) but the calling process did not have a default transaction.
SS$_NOLOG	The local node did not have a transaction log.
SS$_NOSUCHTID	A transaction with the specified transaction identifier does not exist.
SS$_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 $END_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]]

$ADD_HOLDER
Add Holder Record to Rights Database

Adds a specified holder record to a target identifier.

Format

SYS$ADD_HOLDER id ,holder ,[attrib]

Arguments

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

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

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

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

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

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

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

Bit Position	Meaning When Set
KGB$V_DYNAMIC	Allows holders of the identifier to remove it from or add it to the process rights database by using the DCL command SET RIGHTS_LIST.

Bit Position	Meaning When Set
KGB$V_HOLDER_HIDDEN	Prevents someone from getting a list of users who hold an identifier, unless they own the identifier themselves.
KGB$V_NAME_HIDDEN	Allows holders of an identifier to have it translated—either from binary to ASCII or vice versa—but prevents unauthorized users from translating the identifier.
KGB$V_NOACCESS	Makes any access rights of the identifier null and void. This attribute is intended as a modifier for a resource identifier or the Subsystem attribute.
KGB$V_RESOURCE	Allows holders of an identifier to charge disk space to the identifier. It is used only for file objects.
KGB$V_SUBSYSTEM	Allows holders of the identifier to create and maintain protected subsystems by assigning the Subsystem ACE to the application images in the subsystem.

Description

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

Required Access or Privileges

Write access to the rights database is required.

Required Quota

None

Related Services

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

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The **holder** argument cannot be read by the caller.
SS$_BADPARAM	The specified attributes contain invalid attribute flags.
SS$_DUPIDENT	The specified holder already exists in the rights database for this identifier.
SS$_INSFMEM	The process dynamic memory is insufficient for opening the rights database.
SS$_IVIDENT	The specified identifier or holder is of an invalid format, the specified holder is 0, or the specified identifier and holder are equal.

SS$_NORIGHTSDB	The rights database does not exist.
SS$_NOSUCHID	The specified identifier does not exist in the rights database, or the specified holder identifier does not exist in the rights database.
RMS$_PRV	The user does not have write access to the rights database.

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

Arguments

name

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

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

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

id

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

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

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

attrib

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

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

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

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

resid

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

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

Description

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

Required Access or Privileges

Write access to the rights database is required.

Required Quota

None

Related Services

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

Condition Values Returned

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

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

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. 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.

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

rem_user

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

Remote user name of the proxy to be added to or modified in the proxy database. The **rem_user** argument is the address of a character-string descriptor pointing to the user name string.

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

The **rem_user** argument can be specified in user identification code (UIC) format ([*group*, *member*]). Brackets are allowed only if the remote user name string specifies a UIC. Group and member are character-string representations of octal numbers with no leading zeros.

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

local_user

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

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

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

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

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

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

flags

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

Functional specification for the service and type of user the **local_user** argument represents. The **flags** argument is a longword bit mask wherein each bit corresponds to an option.

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

Symbolic Name	Description
PRX$M_BYPASS_EXPAND	The service should not convert the node name specified in the **rem_node** argument to its corresponding DECnet for OpenVMS full name. If this flag is set, it is the caller's responsibility to ensure that the fully expanded node name is passed into the service.
PRX$M_DEFAULT	The user name specified by the **local_user** argument is the default user for the proxy. If this flag is not specified, the user name specified by the **local_user** argument is added to the proxy record's local user list.

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 **rem_node**, **rem_user**, **local_user**, or **flags** argument cannot be read by the service.
SS$_BADPARAM	An invalid flag was specified in the **flags** argument.
SS$_BADBUFLEN	The length of the **rem_node**, **rem_user**, or **local_user** argument was out of range.
SS$_NOSYSPRV	The caller does not have access to the proxy database.

This service can also return any of the following messages passed from the security server, or any OpenVMS RMS error message encountered during operations on the proxy database:

SECSRV$_ ALREADYDEFAULT	The default user specified is already the default.
SECSRV$_ BADLOCALUSERLEN	The local user name length is out of range.
SECSRV$_ BADNODENAMELEN	The node name length is out of range.
SECSRV$_ BADREMUSERLEN	The remote user name length is out of range.
SECSRV$_ DUPLICATEUSER	The user name specified by the **local_user** argument already exists in the proxy record's local user list.
SECSRV$_NOSUCHUSER	The user name specified in the **local_user** argument is not known to the system.

SECSRV$_PROXYEXISTS	The specified proxy already exists.
SECSRV$_ PROXYNOTACTIVE	Proxy processing is currently stopped. Try the request again later.
SECSRV$_ SERVERNOTACTIVE	The security server is not currently active. Try the request again later.
SECSRV$_ TOOMANYUSERS	The specified proxy already has 16 local users and cannot accommodate any more.
SECSRV$_USEREXISTS	The specified local user already exists in the proxy's local user list, or is the proxy's default user.

$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

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 adjust Argument Specifies:	And the Value Specified by newadr Is:	The Stack Pointer Is:
0	0	Not changed
0	An address	Loaded with the address specified
A value	0	Adjusted by the specified value
A value	An address	Loaded with the specified address, adjusted by the specified value

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

Required Access or Privileges

None

Required Quota

None

Related Services

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

Condition Values Returned

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

$ADJWSL
Adjust Working Set Limit

Adjusts a process's current working set limit by the specified number of 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]

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.

Alpha

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-bit or 64-bit reference (Alpha) 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-bit 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]

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 a VMScluster environment.

Required Access or Privileges

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

Required Quota

None

Related Services

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

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_BUFFEROVF	The service completed successfully. The physical name returned overflowed the buffer provided, and has been truncated.

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

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

$ASCEFC
Associate Common Event Flag Cluster

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]

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 may 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.

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 may 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 when accessing an existing common event flag cluster.

perm

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

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

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

Description

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

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

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

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

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

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

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

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

Required Access or Privileges

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

Required Quota

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

Related Services

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

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The cluster name string or string descriptor cannot be read by the caller.
SS$_EXPORTQUOTA	The process has exceeded the number of event flag clusters with which processes on this port of the multiport (shared) memory can associate.
SS$_EXQUOTA	The process has exceeded its timer queue entry quota; this quota controls the creation of temporary common event flag clusters.
SS$_INSFMEM	The system dynamic memory is insufficient for completing the service.
SS$_ILLEFC	You specified an illegal event flag number. The cluster number must be in the range of event flags 64 through 127.
SS$_INTERLOCK	The bit map lock for allocating common event flag clusters from the specified shared memory is locked by another process.
SS$_IVLOGNAM	The cluster name string has a length of 0 or has more than 15 characters.

SS$_NOPRIV	The process does not have the privilege to create a permanent cluster; the process does not have the privilege to create a common event flag cluster in memory shared by multiple processors; or the protection applied to an existing cluster by its creator prohibits association.
SS$_NOSHMBLOCK	The common event flag cluster has no shared memory control block available.
†SS$_SHMNOTCNT	The shared memory named in the **name** argument is not known to the system. This error can be caused by a spelling error in the string, an improperly assigned logical name, or the failure to identify the multiport memory as shared at system generation time.

†VAX specific

$ASCTIM
Convert Binary Time to ASCII String

Converts an absolute or delta time from 64-bit system time format to an ASCII string.

Format

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

Arguments

timlen

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

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

timbuf

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

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

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

timadr

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

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

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

cvtflg

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

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

minute, second, and hundredths-of-second fields. The default value 0 specifies that $ASCTIM should return the full date and time.

Description

The Convert Binary Time to ASCII String service converts an absolute or delta time from 64-bit system time format to an ASCII string. The service executes at the access mode of the caller and does not check whether address arguments are accessible before it executes. Therefore, an access violation causes an exception condition if the input time value cannot be read or the output buffer or buffer length cannot be written.

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

The ASCII strings returned have the following formats:

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

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

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

Month abbreviations must be uppercase.

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

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

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

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

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 **timbuf** argument is too small.
SS$_INSFARG	One or both required arguments are missing.
SS$_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.

Format

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

Arguments

name

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

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

id

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

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

attrib

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

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

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

Bit Position	Meaning When Set
KGB$V_DYNAMIC	Allows holders of the identifier to remove it from or add it to the process rights database by using the DCL command SET RIGHTS_LIST.

Bit Position	Meaning When Set
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 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. Note that when you use wildcards with this service, the records are returned alphabetically by identifier name.

Required Access or Privileges

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

Required Quota

None

Related Services

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

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The **name** argument cannot be read by the caller, or the **id** or **attrib** arguments cannot be written by the caller.
SS$_INSFMEM	The process dynamic memory is insufficient for opening the rights database.
SS$_IVIDENT	The format of the specified identifier is invalid.

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

Because the rights database is an indexed file accessed with OpenVMS RMS, this service may 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.

Format

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

Arguments

timlen

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

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

timbuf

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

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

utcadr

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

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

cvtflg

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

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

Description

The Convert UTC to ASCII service converts an absolute time from 128-bit UTC format to an ASCII string. The service executes at the access mode of the caller and does not check whether address arguments are accessible before it executes. Therefore, an access violation causes an exception condition if the input time value cannot be read or the output buffer or buffer length cannot be written.

The $ASCUTC service uses the time zone differential factor encoded in the 128-bit UTC to convert the UTC to an ASCII string.

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

The ASCII strings returned have the following format:

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

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

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

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

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

Required Access or Privileges

None

Required Quota

None

Related Services

$BINUTC, $GETUTC, $NUMUTC, $TIMCON

Condition Values Returned

SS_$NORMAL	The service completed successfully.
SS_$BUFFEROVF	The buffer length specified in the **timbuf** argument is too small.
SS_$INVTIME	The UTC time supplied is too small to be represented as a Smithsonian Time, or the UTC time is not valid.

$ASSIGN
Assign I/O Channel

Provides a process with an I/O channel so that input/output operations can be performed on a device, or establishes a logical link with a remote node on a network.

Format

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

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 which $ASSIGN is to assign a channel. The **devnam** argument is the address of a character string descriptor pointing to the device name string.

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

chan

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

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

acmode

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

Access mode to be associated with the channel. The **acmode** argument specifies the access mode. The $PSLDEF macro defines the following symbols for the four access modes.

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

The specified access mode and the access mode of the caller are compared. The less privileged (but the higher numeric valued) of the two access modes becomes

the access mode associated with the assigned channel. I/O operations on the channel can be performed only from equal and more privileged access modes. For more information, see the section on access modes in the *OpenVMS Programming Concepts Manual*.

mbxnam

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

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

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

You must specify the **mbxnam** argument when performing a nontransparent, task-to-task, 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

An optional device-specific argument. The **flags** argument is a longword bit mask. For more information on the applicability of the **flags** argument for a particular device, see the *OpenVMS I/O User's Reference Manual*.

Description

The Assign I/O Channel service provides a process with an I/O channel so that 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.

Required Quota

If the target of the assignment is on a remote node, the process needs sufficient buffer quota to allocate a network control block.

Related Services

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

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_REMOTE	The service completed successfully. A logical link is established with the target on a remote node.
SS$_ABORT	A physical line went down during a network connect operation.
SS$_ACCVIO	The device or mailbox name string or string descriptor cannot be read by the caller, or the channel number cannot be written by the caller.
SS$_CONNECFAIL	For network operations, the connection to a network object timed out or failed.
SS$_DEVACTIVE	You specified a mailbox name, but a mailbox is already associated with the device.
SS$_DEVALLOC	The device is allocated to another process.
SS$_DEVNOTMBX	You specified a logical name for the associated mailbox, but the logical name refers to a device that is not a mailbox.
SS$_DEVOFFLINE	For network operations, the physical link is shutting down.
SS$_EXQUOTA	The target of the assignment is on a remote node and the process has insufficient buffer quota to allocate a network control block.

SS$_FILALRACC	For network operations, a logical link already exists on the channel.
SS$_INSFMEM	The target of the assignment is on a remote node and there is insufficient system dynamic memory to complete the request.
SS$_INVLOGIN	For network operations, the access control information was found to be invalid at the remote node.
SS$_IVDEVNAM	No device name was specified, the logical name translation failed, or the device or mailbox name string contains invalid characters. If the device name is a target on a remote node, this status code indicates that the Network Connect Block has an invalid format.
SS$_IVLOGNAM	The device or mailbox name string has a length of 0 or has more than 63 characters.
SS$_LINKEXIT	For network operations, the network partner task was started, but exited before confirming the logical link (that is, $ASSIGN to SYS$NET).
SS$_NOIOCHAN	No I/O channel is available for assignment.
SS$_NOLINKS	For network operations, no logical links are available. The maximum number of logical links as set for the NCP executor MAXIMUM LINKS parameter was exceeded.
SS$_NOPRIV	For network operations, the issuing task does not have the required privilege to perform network operations or to confirm the specified logical link.
SS$_NOSUCHDEV	The specified device or mailbox does not exist, or, for DECnet for OpenVMS operations, the network device driver is not loaded (for example, the DECnet for OpenVMS software is not currently running on the local node).
SS$_NOSUCHNODE	The specified network node is nonexistent or unavailable.
SS$_NOSUCHOBJ	For network operations, the network object number is unknown at the remote node; for a TASK= connect, the named DCL command procedure file cannot be found at the remote node.
SS$_NOSUCHUSER	For network operations, the remote node could not recognize the login information supplied with the connection request.
SS$_PROTOCOL	For network operations, a network protocol error occurred, most likely because of a network software error.
SS$_REJECT	The network connect was rejected by the network software or by the partner at the remote node, or the target image exited before the connect confirm could be issued.

SS$_REMRSRC	For network operations, the link could not be established because system resources at the remote node were insufficient.
SS$_SHUT	For network operations, the local or remote node is no longer accepting connections.
SS$_THIRDPARTY	For network operations, the logical link connection was terminated by a third party (for example, the system manager).
SS$_TOOMUCHDATA	For network operations, the task specified too much optional or interrupt data.
SS$_UNREACHABLE	For network operations, the remote node is currently unreachable.

$AUDIT_EVENT
Audit Event

Appends an event message to the system security audit log file or sends an alarm to a security operator terminal.

Format

SYS$AUDIT_EVENT [efn] ,[flags] ,itmlst ,[audsts] ,[astadr] ,[astprm]

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$M_ACL	Specifies an event generated by an Alarm ACE or Audit ACE. This flag is reserved to Digital.
NSA$M_FLUSH	Specifies that all messages in the audit server buffer be written to the audit log file.
NSA$M_INTERNAL	Specifies that the $AUDIT_EVENT call originates in the context of a trusted computing base (TCB) component. The auditing components use this flag to indicate that internal auditing failures should result in a SECAUDTCB bugcheck. This flag is reserved to Digital.
NSA$M_MANDATORY	Specifies that an audit is to be performed, regardless of system alarm and audit settings.

Symbolic Name	Description
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 Digital.

itmlst

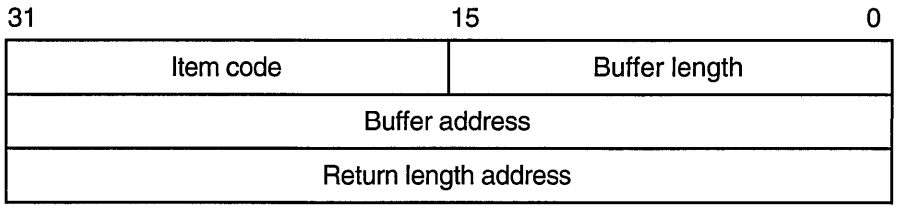
OpenVMS usage: item_list_3
type: longword (unsigned)
access: read only
mechanism: by reference

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$_EVENT_TYPE (see Table SYS1–1)
- NSA$_EVENT_SUBTYPE (see Table SYS1–1)
- At least one of the NSA$_ALARM_NAME item code or the NSA$_AUDIT_NAME item code.
- If the event being reported is an object access (NSA$C_MSG_OBJ_ACCESS) or an object delete (NSA$C_MSG_OBJ_DELETE), the NSA$_FINAL_STATUS, NSA$_ACCESS_DESIRED, and NSA$_OBJECT_CLASS item codes must be specified.
- If the event being reported is an object create (NSA$C_MSG_OBJ_CREATE), the NSA$_FINAL_STATUS and NSA$_OBJECT_CLASS item codes must be specified.
- If the event being reported is a privilege audit (NSA$C_MSG_PRVAUD), the NSA$_PRIVS_USED or the NSA$_PRIVS_MISSING item code must be specified.
- If the audit event being reported is a deaccess event (NSA$C_MSG_OBJ_DEACCESS), the NSA$_OBJECT_CLASS item code must be specified.

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



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 Digital. 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 SS$_EVTNOTENAB. In addition, the caller must either make use of the **astadr** argument or use the $AUDIT_EVENTW service before attempting to access **audsts**.

astadr

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

Asynchronous system trap (AST) routine to be executed after the **audsts** is updated. The **astadr** argument, which is the address of a longword value, is the procedure value of the AST routine.

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

astprm

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

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

Item Codes

This section provides a list of item codes that may 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 SYS1–1 for a list of valid event subtypes.

NSA$_EVENT_TYPE
NSA$_EVENT_TYPE is a longword value specifying an event message type. See Table SYS1–1 for a list of valid event types.

Table SYS1–1 Description of $AUDIT_EVENT Types and Subtypes

Symbol of Event Type	Meaning
NSA$C_MSG_AUDIT	Systemwide change to auditing
Subtype and Meaning	
NSA$C_AUDIT_DISABLED	Audit events disabled
NSA$C_AUDIT_ENABLED	Audit events enabled
NSA$C_AUDIT_INITIATE	Audit server startup
NSA$C_AUDIT_TERMINATE	Audit server shutdown
NSA$C_AUDIT_LOG_FINAL	Final entry in audit log (forward link)
NSA$C_AUDIT_LOG_FIRST	First entry in audit log (backward link)
NSA$C_MSG_BREAKIN	Break-in attempt detected
Subtype and Meaning	
NSA$C_DETACHED	Detached process
NSA$C_DIALUP	Dialup interactive process
NSA$C_LOCAL	Local interactive process
NSA$C_NETWORK	Network server process
NSA$C_REMOTE	Interactive process from another network node
NSA$C_MSG_CONNECTION	Logical link connection or termination
Subtype and Meaning	
NSA$C_CNX_ABORT	Connection aborted
NSA$C_CNX_ACCEPT	Connection accepted
NSA$C_CNX_DECNET_CREATE	DECnet for OpenVMS logical link created
NSA$C_CNX_DECNET_DELETE	DECnet for OpenVMS logical link disconnected
NSA$C_CNX_DISCONNECT	Connection disconnected
NSA$C_CNX_IPC_CLOSE	Interprocess communication association closed
NSA$C_CNX_IPC_OPEN	Interprocess communication association opened
NSA$C_CNX_REJECT	Connection rejected
NSA$C_CNX_REQUEST	Connection requested
NSA$C_CNX_INC_REQUEST	Incoming connection requested
NSA$C_CNX_INC_ACCEPT	Incoming connection accepted
NSA$C_CNX_INC_REJECT	Incoming connection rejected
NSA$C_CNX_INC_DISCONNECT	Incoming connection disconnected
NSA$C_CNX_INC_ABORT	Incoming connection aborted
NSA$C_MSG_INSTALL	Use of the Install utility (INSTALL)
Subtype and Meaning	
NSA$C_INSTALL_ADD	Known image installed
NSA$C_INSTALL_REMOVE	Known image deleted
NSA$C_MSG_LOGFAIL	Login failure

(continued on next page)

Table SYS1–1 (Cont.) Description of $AUDIT_EVENT Types and Subtypes

Symbol of Event Type	Meaning
Subtype and Meaning	
NSA$C_BATCH	Batch process
NSA$C_DETACHED	Detached process
NSA$C_DIALUP	Dialup interactive process
NSA$C_LOCAL	Local interactive process
NSA$C_NETWORK	Network server process
NSA$C_REMOTE	Interactive process from another network node
NSA$C_SUBPROCESS	Subprocess
NSA$C_MSG_LOGIN	Successful login
Subtype and Meaning	
See subtypes for NSA$C_MSG_LOGFAIL	
NSA$C_MSG_LOGOUT	Successful logout
Subtype and Meaning	
See subtypes for NSA$C_MSG_LOGFAIL	
NSA$C_MSG_MOUNT	Volume mount or dismount
Subtype and Meaning	
NSA$C_VOL_DISMOUNT	Volume dismount
NSA$C_VOL_MOUNT	Volume mount
NSA$C_MSG_NCP	Modification to network configuration database
Subtype and Meaning	
NSA$C_NCP_COMMAND	Network Control Program (NCP) command issued
NSA$C_MSG_NETPROXY	Modification to network proxy database
Subtype and Meaning	
NSA$C_NETPROXY_ADD	Record added to network proxy database
NSA$C_NETPROXY_DELETE	Record removed from network proxy database
NSA$C_NETPROXY_MODIFY	Record modified in network proxy database
NSA$C_MSG_OBJ_ACCESS	Object access attempted
Subtype and Meaning	
NSA$C_OBJ_ACCESS	Object access attempted
NSA$C_MSG_OBJ_CREATE	Object created
Subtype and Meaning	
NSA$C_OBJ_CREATE	Object created

(continued on next page)

Table SYS1–1 (Cont.) Description of $AUDIT_EVENT Types and Subtypes

Symbol of Event Type	Meaning
NSA$C_MSG_OBJ_DEACCESS	Object deaccessed
Subtype and Meaning	
NSA$C_OBJ_DEACCESS	Object deaccessed
NSA$C_MSG_OBJ_DELETE	Object deleted
Subtype and Meaning	
NSA$C_OBJ_DELETE	Object deleted
NSA$C_MSG_PROCESS	Process control system service issued
Subtype and Meaning	
NSA$C_PRC_CANWAK	Process wakeup canceled
NSA$C_PRC_CREPRC	Process created
NSA$C_PRC_DELPRC	Process deleted
NSA$C_PRC_FORCEX	Process exit forced
NSA$C_PRC_GETJPI	Process information gathered
NSA$C_PRC_GRANTID	Process identifier granted
NSA$C_PRC_RESUME	Process resumed
NSA$C_PRC_REVOKID	Process identifier revoked
NSA$C_PRC_SCHDWK	Process wakeup scheduled
NSA$C_PRC_SETPRI	Process priority altered
NSA$C_PRC_SIGPRC	Process exception issued
NSA$C_PRC_SUSPND	Process suspended
NSA$C_PRC_WAKE	Process wakeup issued
NSA$C_PRC_PRCTERM	Process termination notification requested
NSA$C_MSG_PRVAUD	Attempt to use privilege
Subtype and Meaning	
NSA$C_PRVAUD_FAILURE	Unsuccessful use of privilege
NSA$C_PRVAUD_SUCCESS	Successful use of privilege
NSA$C_MSG_RIGHTSDB	Modification to rights database
Subtype and Meaning	
NSA$C_RDB_ADD_ID	Identifier added to rights database
NSA$C_RDB_CREATE	Rights database created
NSA$C_RDB_GRANT_ID	Identifier given to user
NSA$C_RDB_MOD_HOLDER	List of identifier holders modified
NSA$C_RDB_MOD_ID	Identifier name or attributes modified
NSA$C_RDB_REM_ID	Identifier removed from rights database
NSA$C_RDB_REVOKE_ID	Identifier revoked from user
NSA$C_MSG_SYSGEN	Modification of a System Generation utility (SYSGEN) parameter

(continued on next page)

Table SYS1–1 (Cont.) Description of $AUDIT_EVENT Types and Subtypes

Symbol of Event Type	Meaning
Subtype and Meaning	
NSA$C_SYSGEN_SET	System Generation utility parameter modified
NSA$C_MSG_SYSTIME	Modification to system time
Subtype and Meaning	
NSA$C_SYSTIM_SET	System time set
NSA$C_SYSTIM_CAL	System time calibrated
NSA$C_MSG_SYSUAF	Modification to system user authorization file (SYSUAF)
Subtype and Meaning	
NSA$C_SYSUAF_ADD	Record added to system user authorization file
NSA$C_SYSUAF_COPY	Record copied in system user authorization file
NSA$C_SYSUAF_DELETE	Record deleted from system user authorization file
NSA$C_SYSUAF_MODIFY	Record modified in system user authorization file
NSA$C_SYSUAF_RENAME	Record renamed in system user authorization file

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$_MESSAGE. The default file specification is SYS$MESSAGE:.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 bitmask directing $AUDIT_EVENT to ignore the defaults for the following values and either omit the information from the event record or use the value provided in another parameter. The bits in the mask inhibit the use of default values for the following item codes:

NSA$V_ACCOUNT_NAME	NAS$V_PROCESS_NAME
NSA$V_FINAL_STATUS	NSA$V_SUBJECT_CLASS
NSA$V_IMAGE_NAME	NSA$V_SUBJECT_OWNER
NSA$V_PARENT_ID	NSA$V_SYSTEM_ID
NSA$V_PARENT_NAME	NSA$V_SYSTEM_OWNER
NSA$V_PARENT_OWNER	NSA$V_TERMINAL
NSA$V_PARENT_USERNAME	NSA$V_TIME_STAMP
NSA$V_PROCESS_ID	NSA$V_USERNAME

Use NSA$_SUPPRESS, for example, when auditing events from server processes when the default values for many of these items need to explicitly reference the client context rather than be defaulted from the environment of the server.

The following section provides a list of additional item codes that are valid as an item descriptor in the **itmlst** argument.

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 quadword value specifying the dismount flags, which 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.

NSA$_INSTALL_FLAGS
NSA$_INSTALL_FLAGS is a longword value specifying the INSTALL flags. They correspond to qualifiers for the Install utility; for example, NSA$M_INS_EXECUTE_ONLY.

NSA$_LNM_PARENT_NAME
NSA$_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 longword 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.

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.

NSA$_SYSTEM_ID
NSA$_SYSTEM_ID is a longword value specifying the SCS identification of the cluster node where the event took place (SYSGEN 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 (SYSGEN 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 in order to append an event message to the audit log file or send an alarm to an operator terminal. For example, AUTHORIZE calls $AUDIT_EVENT whenever a UAF record is altered and LOGINOUT calls the service whenever a user logs in.

$AUDIT_EVENT takes the event message, checks the auditing database to determine whether a class of event is being audited, and, if the event class is enabled, creates an alarm or audit record.

$AUDIT_EVENT completes asynchronously; that is, it does not wait for final status. For synchronous completion, use the $AUDIT_EVENTW service.

Required Access or Privileges

AUDIT

Required Quota

None

Related Services

$CHECK_ACCESS, $CHECK_PRIVILEGE, $CHKPRO

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	A parameter is not accessible.
SS$_BADBUFADR	The buffer address is invalid or not readable.
SS$_BADBUFLEN	The specified buffer length is invalid or out of range.
SS$_BADCHAIN	The address of the next item list to be processed, as identified in the buffer address field, is either not readable or points to itself.
SS$_BADITMCOD	The specified item code is invalid or out of range.
SS$_EVTNOTENAB	The event is not enabled.
SS$_INSFARG	A required item code or parameter is missing.
SS$_INVAJLNAM	The alarm or audit journal name is invalid.
SS$_IVSTSFLG	The specified system service flags are invalid.
SS$_NOAUDIT	The caller does not have the required privilege to perform the audit.
SS$_OVRMAXAUD	There is insufficient memory to perform the audit.
SS$_SYNCH	An audit was not required.

$AUDIT_EVENTW
Audit Event and Wait

Determines whether a security-related event should be reported. If the event should be reported, the service sends the event report to the audit server.

The $AUDIT_EVENTW service completes synchronously; that is, it returns only after receiving an explicit confirmation from the audit server that the associated audit, if enabled, has been performed.

Format

SYS$AUDIT_EVENTW efn ,[flags] ,itmlst ,audsts ,[astadr] ,[astprm]

$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.

Format

SYS$BINTIM timbuf ,timadr

Arguments

timbuf

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

Buffer that holds the ASCII time to be converted. The **timbuf** argument specifies the address of a character string descriptor pointing to the 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 reference

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

Description

The Convert ASCII String to Binary Time service converts an ASCII string to an absolute or delta time value in the system 64-bit time format suitable for input to the Set Timer ($SETIMR) or Schedule Wakeup ($SCHDWK) service. The service executes at the access mode of the caller and does not check whether address arguments are accessible before it executes. Therefore, an access violation causes an exception condition if the input buffer or buffer descriptor cannot be read or the output buffer cannot be written.

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

The required ASCII input strings have the following format:

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

The following table lists the length (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

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

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

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

  ```
  0 ::10
  ```

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

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

Required Access or Privileges

None

Required Quota

None

Related Services

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

Condition Values Returned

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

Example

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

Format

SYS$BINUTC timbuf ,utcadr

Arguments

timbuf

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

Buffer that holds the ASCII time to be converted. The **timbuf** argument specifies the address of a character string descriptor pointing to 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 reference

Time value that $BINUTC has converted. The **utcadr** argument is the address of a 16-byte location to receive the converted time.

Description

The Convert ASCII String to UTC Binary Time service converts an ASCII string to an absolute time in the 128-bit UTC format. The service executes at the access mode of the caller and does not check whether address arguments are accessible before it executes. Therefore, an access violation causes an exception condition if the input buffer or buffer descriptor cannot be read or the output buffer cannot be written.

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

$BINUTC uses the time zone differential factor of the local system to encode the 128-bit UTC.

The required ASCII input strings have the following format:

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

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

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

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

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

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

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

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

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

Required Access or Privileges

None

Required Quota

None

Related Services

$ASCUTC, $GETUTC, $NUMUTC, $TIMCON

Condition Values Returned

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

$BRKTHRU
Breakthrough

Sends a message to one or more terminals. The $BRKTHRU service completes asynchronously; that is, it returns to the caller after queuing the message request, without waiting for the message to be written to the specified terminals.

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

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

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

Format

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

Arguments

efn

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

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

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

msgbuf

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

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

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

sendto

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

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

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

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

sndtyp

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

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

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

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

iosb

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

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

When the **iosb** argument is specified, $BRKTHRU sets the quadword to 0 when it queues the message request. Then, after the message is sent to the specified terminals, $BRKTHRU returns four informational items, one item per word, in the quadword I/O status block.

These informational items indicate the status of the messages sent only to terminals and mailboxes on the local node; these items do not include the status of messages sent to terminals and mailboxes on other nodes in a VMScluster 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 BRK$M_SCREEN flag in the **flags** argument. See the description of the **flags** argument.

flags

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

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

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

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

reqid

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

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

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

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

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

For example, you can prevent the Mail utility (which is an image) from writing to the terminal by issuing 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 (1) the image invoked by the DCL command REPLY and (2) the callers of the $BRKTHRU service. This is the default if the **reqid** argument is not specified.
BRK$C_PHONE	This class name is used by the OpenVMS Phone utility.
BRK$C_MAIL	This class name is used by the OpenVMS Mail utility.
BRK$C_DCL	This class name is used by the DIGITAL Command Language (DCL) interpreter for the Ctrl/T command, which displays the process status.
BRK$C_QUEUE	This class name is used by the queue manager, which manages print and batch jobs.
BRK$C_SHUTDOWN	This class name is used by the system shutdown image, which is invoked by the DCL command REPLY /ID=SHUTDOWN.

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.

timout

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

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

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

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

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

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

astadr

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

AST service routine to be executed after $BRKTHRU has sent the message to the specified terminals. The **astadr** argument is the address of this routine.

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

astprm

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

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

Description

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

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

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

- If the terminal is performing a read operation when $BRKTHRU sends the message, the read operation is suspended, the message is displayed, and then the line that was being read when the read operation was suspended is redisplayed (equivalent to the action produced by CTRL/R).
- If the terminal is performing a write operation when $BRKTHRU sends the message, the message is displayed after the current write operation has completed.
- If the terminal has the NOBROADCAST characteristic set for all images, or if you have disabled the receiving of messages from the image that is issuing the $BRKTHRU call (see the description of the **reqid** argument), the message is not displayed.

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

Required Access or Privileges

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

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

Required Quota

The $BRKTHRU service allows the message text to be as long as 16,350 bytes; however, both the system parameter MAXBUF and the caller's available process buffered I/O byte count limit (BYTLM) quota must be sufficient to handle the message.

Related Services

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

Condition Values Returned

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

Condition Values Returned in the I/O Status Block

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

$BRKTHRUW
Breakthrough and Wait

Sends a message to one or more terminals. The $BRKTHRUW service operates synchronously; that is, it returns to the caller after the message has been sent to the specified terminals.

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

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

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

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

Format

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

$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

Argument

chan

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

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

Description

The Cancel I/O on Channel service cancels all pending I/O requests on a specified channel. In general, this includes all I/O requests that are queued, as well as the request currently in progress.

When you cancel a request currently in progress, the driver is notified immediately. The actual cancellation might occur immediately, depending on the logical state of the driver. When cancellation does occur, the following action for I/O in progress, similar to that for queued requests, takes place:

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

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

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

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

Required Access or Privileges

To cancel I/O on a channel, the access mode of the calling process must be equal to or more privileged than the access mode that the process had when it originally made the channel assignment.

Required Quota

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

Related Services

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

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_EXQUOTA	The process has exceeded its buffered I/O limit (BIOLM) quota.
SS$_INSFMEM	The system dynamic memory is insufficient for canceling the I/O.
SS$_IVCHAN	You specified an invalid channel, that is, a channel number of 0 or a number larger than the number of channels available.
SS$_NOPRIV	The specified channel is not assigned or was assigned from a more privileged access mode.

$CANEXH
Cancel Exit Handler

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

Format

SYS$CANEXH [desblk]

Argument

desblk

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

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

Condition Values Returned

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

$CANTIM
Cancel Timer

Cancels all or a selected subset of the Set Timer requests previously issued by the current image executing in a process. Cancellation is based on the request identification specified in the Set Timer ($SETIMR) service. If you give the same request identification to more than one timer request, all requests with that request identification are canceled.

Format

SYS$CANTIM [reqidt] ,[acmode]

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, $CANWAK, $GETTIM, $NUMTIM, $SCHDWK, $SETIME, $SETIMR

Condition Values Returned

SS$_NORMAL	The service completed successfully.

$CANWAK
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 ($SCHDWK) service makes scheduled wakeup requests.

Format

SYS$CANWAK [pidadr] ,[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 VMScluster 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 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 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 ($SCHDWK) 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 $CANWAK:

- 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.)

$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]

Arguments

objtyp

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

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

Symbol	Meaning
ACL$C_CAPABILITY	Object is a restricted resource; use the reserved name VECTOR.
ACL$C_DEVICE	Object is a device.
ACL$C_FILE	Object is a Files–11 On-Disk Structure Level 2 file.
ACL$C_GROUP_GLOBAL_SECTION	Object is a group global section.
ACL$C_JOBCTL_QUEUE	Object is a batch, print, or server queue.
ACL$C_LOGICAL_NAME_TABLE	Object is a logical name table.
ACL$C_SYSTEM_GLOBAL_SECTION	Object is a system global section.

For further information about these symbols, see the description of the **clsnam** argument.

objnam

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

Name of the object being accessed. The **objnam** argument is the address of a character-string descriptor pointing to the object name.

usrnam

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

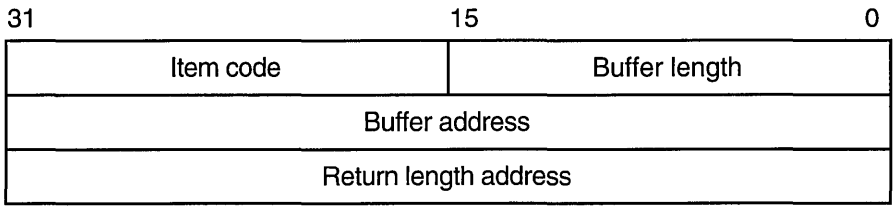
Name of the user attempting access. The **usrnam** argument is the address of a descriptor that points to a character string that contains the name of the user attempting to gain access to the specified object. The user name string can contain a maximum of 12 alphanumeric characters.

itmlst

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

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

For each item code, you must include a set of four elements and end the list with a longword containing the value 0 (CHP$_END), as 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 upon the item code specified in the item code field of the item descriptor. If the value of buffer length is too small, the service truncates the data.
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.

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 **bufadr**. If the buffer pointed to by **bufadr** is used to pass information to $CHECK_ACCESS, **retlenadr** is ignored but must be included.

contxt

OpenVMS usage: longword
type: longword (unsigned)
access: read-write
mechanism: by reference

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

Using the **contxt** argument keeps the UAF open across all calls, thereby improving the performance of the system on subsequent calls. To close the UAF, you must run down the image.

The resulting **contxt** value from a $CHECK_ACCESS call may 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
COMMON_EVENT_CLUSTER
DEVICE
FILE
GROUP_GLOBAL_SECTION
LOGICAL_NAME_TABLE
QUEUE
RESOURCE_DOMAIN
SECURITY_CLASS
SYSTEM_GLOBAL_SECTION
VOLUME

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 may 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 may also be used to specify the first four access types for any object class.

The following table shows the access types available and lists their common interpretations. These symbols are defined in the $ARMDEF system macro library. For more information, see the *OpenVMS Guide to System Security*.

Access Type	Access Permitted
ARM$M_READ	Allows holders to read an object, perform wildcard directory lookups, display jobs in a queue, or use an associated vector processor.
ARM$M_WRITE	Allows holders to alter the contents of an object, remove a directory entry, write or extend existing files on a volume, or submit a job to a queue.
ARM$M_EXECUTE	Allows holders to run an image or command procedure, perform exact directory lookups, issue physical I/O requests to a device, create new files on a volume, or act as operator for a queue.
ARM$M_DELETE	Allows holders to delete an object, perform logical I/O to a device, or delete a job in a queue.
ARM$M_CONTROL	Allows holders to display or alter the security characteristics of an object.

CHP$_ACMODE
A byte that defines the accessor's processor access mode ($PSLDEF). The following access modes and their symbols are defined in the system macro library ($PSLDEF). Objects supported by the operating system do not consider access mode in determining object access.

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

If CHP$_ACMODE is not specified, access mode is not used to determine access.

CHP$_ALARMNAME
Address of a buffer to receive the alarm name from any Alarm ACE contained in the object's ACL. Currently, if a matching Alarm ACE exists, the string SECURITY will be returned. The string returned by CHP$_ALARMNAME may be used as input to the $AUDIT_EVENT system service, using the NSA$_ ALARM_NAME item code.

CHP$_AUDIT_LIST
A list containing information to be added to any resulting security audit. The **bufadr** argument points to the beginning of an $AUDIT_EVENT item list. See the **itmlst** argument of the $AUDIT_EVENT system service for a list of valid security auditing item codes. Note that the NSA$_EVENT_TYPE and NSA$_ EVENT_SUBTYPE items are ignored when auditing with $CHECK_ACCESS. The CHP$V_AUDIT flag must be specified.

CHP$_AUDITNAME
Address of a buffer to receive the audit name from any Audit ACE contained in the object's ACL. Currently, if a matching Audit ACE exists, the string SECURITY will be returned. The string returned by CHP$_AUDITNAME may be used as input to the $AUDIT_EVENT system service, using the NSA$_ AUDIT_NAME item code.

CHP$_FLAG

A longword that controls various aspects of the protection check. The symbols in the following table are offsets to the bits within the longword. You can also obtain the values as masks with the appropriate bit set by using the prefix CHP$M rather than CHP$V. These symbols are defined in the system macro library ($CHPDEF).

Symbol	Access
CHP$V_ALTER	Accessor desires write access to object.
CHP$V_AUDIT	Access audit requested.
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 Digital.
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.

Symbol	Meaning
CHP$V_SYSNAM	SYSNAM was used to gain the requested access.
CHP$V_GROUP	GROUP was used to gain the requested access.
CHP$V_WORLD	WORLD was used to gain the requested access.
CHP$V_PRMCEB	PRMCEB was used to gain the requested access.
CHP$V_UPGRADE	UPGRADE was used to gain the requested access.
CHP$V_DOWNGRADE	DOWNGRADE was used to gain the requested access.

Description

The Check Access service invokes the operating system control protection check mechanism, $CHKPRO, to determine whether a named user is allowed the described access to the named object. A file server, for example, might check the access attributes of a user who attempts to access a file (the object).

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

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

The caller may also request that an object access audit be performed if security auditing has been enabled for the object class or if Audit ACEs are contained in the object's ACL. Auditing ACEs include both Alarm ACEs and Audit ACEs. The CHP$V_AUDIT flag requests an access audit. This requires that the caller be in executive or kernel mode or possess the AUDIT privilege.

Normally, $CHECK_ACCESS generates an object access audit when an audit is required. The caller may specify the CHP$V_CREATE flag to force an object creation audit instead of an object access audit. Similarly, the CHP$V_DELETE flag forces an object deletion audit. The CHP$_AUDIT_LIST item code may be used to specify additional information to be included in any resulting audit records.

With certain types of devices, $CHECK_ACCESS may 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 may 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 may be necessary to test for these additional privileges using the $CHECK_PRIVILEGE system service. See 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 may also be returned.

$CHECK_FEN (Alpha Only) Check Floating Point

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

Format

SYS$CHECK_FEN

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] ,prvadr ,[altprv] ,[flags] ,[itmlst] ,[audsts] ,[astadr] ,[astprm]

Arguments

efn

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

Number of the event flag to be set when the audit completes. The **efn** argument is a longword containing the number of the event flag; however, $CHECK_PRIVILEGE uses only the low-order byte. If **efn** is not specified, event flag 0 is used.

Upon request initiation, $CHECK_PRIVILEGE clears the specified event flag.

prvadr

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

The privilege or identifier to be checked. The **prvadr** argument is either the address of a quadword bit array, where each bit corresponds to a privilege, or the address of a quadword identifier.

When the array lists privileges, each bit has a symbolic name. The $PRVDEF macro defines these names. You form the bit array by specifying the symbolic name of each desired privilege in a logical OR operation. See the $SETPRV system service for the symbolic name and description of each privilege.

If the caller passes an identifier, the caller must set the NSA$M_IDENTIFIER bit in the **flags** longword. The identifier structure is defined by the $KGBDEF macro. The identifier attributes (KGB$) are reserved for future use and should be set to 0.

altprv

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

Alternate privilege mask to check against. The **altprv** argument is the address of a quadword privilege mask, where each bit corresponds to a privilege. This argument and the flags NSA$M_AUTHPRIV, NSA$M_IDENTIFIER, and NSA$M_PROCPRIV are mutually exclusive.

With this argument, $CHECK_PRIVILEGE uses the supplied set of privileges instead of the current, active privileges. Each bit in the mask has a symbolic name, defined by the $PRVDEF macro. You form the bit array by specifying the symbolic name of each desired privilege in a logical OR operation. See the $SETPRV system service for the symbolic name and description of each privilege.

flags

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

Flags that specify options for the $CHECK_PRIVILEGE operation. The **flags** argument is a longword bit mask, where each bit corresponds to an option.

Each flag option has a symbolic name. The $NSADEF macro defines the following symbolic names. Be aware that the flags NSA$M_AUTHPRIV, NSA$M_IDENTIFIER, and NSA$M_PROCPRIV are mutually exclusive; therefore, you can specify only one of these flag options.

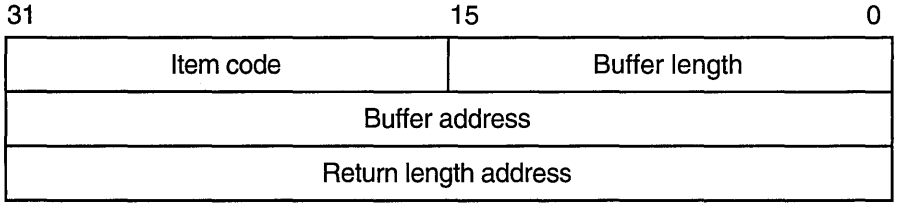
Symbolic Name	Description
NSA$M_AUTHPRIV	Checks the authorized privileges of the process instead of the current (active) privileges.
NSA$M_FLUSH	Specifies that all messages in the audit server buffer be written to the audit log file.
NSA$M_IDENTIFIER	Interprets the **prvadr** argument as the address of an identifier instead of a privilege mask.
NSA$M_INTERNAL	Specifies that the $CHECK_PRIVILEGE call originates in the context of a trusted computing base (TCB) component. The auditing components use this flag to indicate that internal auditing failures should result in a SECAUDTCB bugcheck. This flag is reserved to Digital.
NSA$M_MANDATORY	Specifies that an audit is to be performed, regardless of system alarm and audit settings.
NSA$M_PROCPRIV	Checks the permanent privileges of the process, instead of the privileges in the current (active) mask.
NSA$M_SERVER	Indicates that the call originates in a TCB server process and that the event should be audited regardless of the state of a process-specific no-audit bit. Trusted servers use this flag to override the no-audit bit when they want to perform explicit auditing on behalf of a client process. This flag is reserved to Digital.

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 Digital. You should specify 0.

All item codes listed in the Item Codes section of the $AUDIT_EVENT service are valid within the item list used by the $CHECK_PRIVILEGE service except for the NSA$_EVENT_TYPE and NSA$_EVENT_SUBTYPE item codes, which are supplied internally by the $CHECK_PRIVILEGE service.

$CHECK_PRIVILEGE should be called with an item list identifying the alarm and audit journals, and does not need to use the NSA$_PRIVS_USED item code. NSA$_PRIVS_USED is supplied automatically by the $CHECK_PRIVILEGE service. Note that $CHECK_PRIVILEGE returns SS$_BADPARAM if you supply either NSA$_EVENT_TYPE or NSA$_EVENT_SUBTYPE. These items are supplied internally by $CHECK_PRIVILEGE.

audsts

OpenVMS usage: cond_value_type
type: longword (unsigned)
access: write only

mechanism: by reference

Longword condition value that receives a final completion status from the operation. If a security audit is required, the final completion status represents either the successful completion of the resulting security audit or any failing status that occurred while the security audit was performed within the AUDIT_SERVER process.

The **audsts** argument is valid only when the service returns success and the status is not SS$_EVTNOTENAB. In addition, the caller must either make use of the **astadr** argument or use the $CHECK_PRIVILEGEW service before attempting to access **audsts**.

astadr

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

Asynchronous system trap (AST) routine to be executed after the **audsts** argument is written. The **astadr** argument, which is the address of a longword value, is the procedure value of the AST routine.

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

astprm

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

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

Description

The Check Privilege service determines whether a user has the privileges or identifier that an operation requires. In addition, $CHECK_PRIVILEGE audits the use of privilege if privilege auditing has been enabled by the site security administrator. The caller does not need to determine whether privilege auditing has been enabled.

Required Access or Privileges

AUDIT privilege is required.

Required Quota

None

Related Services

$AUDIT_EVENT, $SETPRV

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The specified parameter of the item list buffer is not accessible.
SS$_BADBUFADR	The buffer address is invalid or not readable.
SS$_BADBUFLEN	The specified buffer length is invalid or out of range.
SS$_BADCHAIN	The address of the next item list to be processed, as identified in the buffer address field, is either not readable or points to itself.
SS$_BADITMCOD	The specified item code is invalid or out of range.
SS$_BADPARAM	The specified list entry is invalid or out of range.
SS$_EVTNOTENAB	No audit required; privilege granted.
SS$_ILLEFC	You specified an illegal event flag number.
SS$_INSFARG	The required item code is not present.
SS$_INVAJLNAM	The alarm or audit journal name is invalid.
SS$_IVSTSFLG	The specified system service flags are invalid.
SS$_NOAUDIT	The caller does not have the required privilege to perform the audit.
SS$_NOPRIV	The subject does not have the required privileges or identifier.
SS$_NO[privilege-name]	The subject does not have a specific privilege.
SS$_OVRMAXAUD	There is insufficient memory to perform the audit.
SS$_TOOMANYAJL	Too many alarm or audit journals were specified.
SS$_UNASEFC	An unassociated event flag cluster was specified.

$CHECK_PRIVILEGEW
Check Privilege and Wait

Determines whether the caller has the specified privileges or identifier. In addition to checking for a privilege or an identifier, the Check Privilege and Wait service determines if the caller's use of privilege needs to be audited.

$CHECK_PRIVILEGEW completes synchronously; that is, it returns the final status to the caller only after receiving an explicit confirmation from the audit server that the associated audit, if enabled, has been performed.

Format

SYS$CHECK_PRIVILEGEW efn ,prvadr ,[altprv] ,[flags] ,[itmlst] ,audsts ,[astadr] ,[astprm]

$CHKPRO
Check Access Protection

Determines whether an accessor with the specified rights and privileges can access an object with the specified attributes.

Format

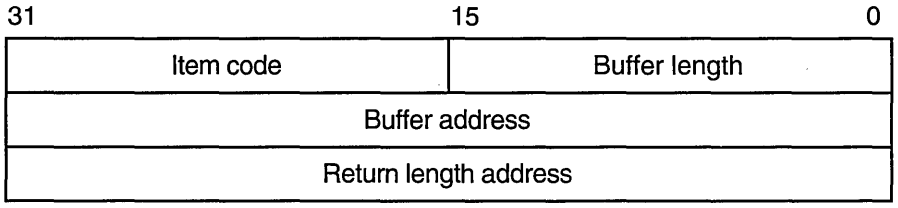
SYS$CHKPRO itmlst ,[objpro] ,[usrpro]

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.

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

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 Digital.

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 may 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 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.

CHP$_AUDITNAME
Address of a buffer to receive the audit name from any Audit ACE contained in the object's ACL. If the object does not have auditing enabled, $CHKPRO returns **retlenadr** as 0. If a matching Audit ACE exists, the string SECURITY will be returned.

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

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

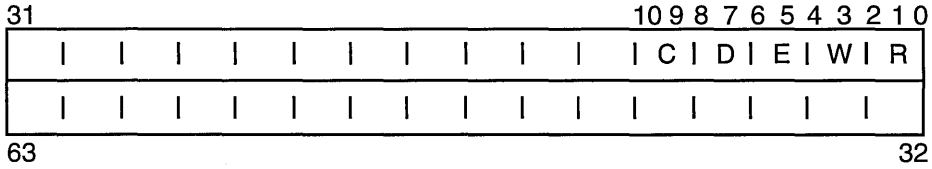
CHP$_MATCHEDACE
This output item is a variable-length data structure containing the first Identifier ACE in the object's ACL that allowed or denied the accessor to access the object. See the $FORMAT_ACL system service for a description of an Identifier ACE format.

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

Symbol	Access Mode
PSL$C_EXEC	Executive
PSL$C_KERNEL	Kernel

CHP$_MODES
A quadword that defines the object's access mode protection. You specify a 2-bit access mode as shown in CHP$_MODE for each possible access type. The following diagram illustrates the format of an access mode vector for bit numbers.



ZK-1943-GE

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

CHP$_OBJECT_CLASS
A character string containing the protected object class associated with the object. The object class string is used to determine whether any security auditing is enabled for the object access event. This item code is required when the CHP$_AUDIT flag is specified.

CHP$_OBJECT_NAME
A character string containing the object name associated with the protection check. The object name string is included in any resulting security audit. If an object name string is not specified, the string "<not available>" is substituted in any security audit for all protected object classes other than FILE. For FILE class audits, it is assumed that the caller has supplied an object name by using the auditing item list (NSA$_OBJECT_NAME).

CHP$_OWNER
A longword describing the object's owner identifier (UIC or general identifier). This might be either a UIC format identifier or a general identifier.

Note

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

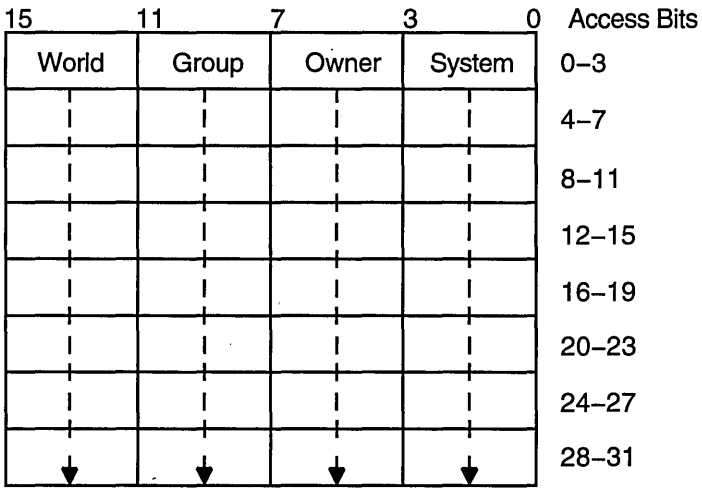
CHP$_PRIV
A quadword that defines an accessor's privilege mask. Each bit in the mask has a symbolic name, defined by the $PRVDEF macro. You form the bit array by specifying the symbolic name of each privilege in a logical OR operation. See the $SETPRV system service for the symbolic name and description of each privilege.

CHP$_PRIVUSED
A longword mask of flags representing privileges used to gain the requested access.

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

Symbol	Meaning
CHP$V_SYSPRV	SYSPRV was used to gain the requested access.
CHP$V_GRPPRV	GRPPRV was used to gain the requested access.
CHP$V_BYPASS	BYPASS was used to gain the requested access.
CHP$V_READALL	READALL was used to gain the requested access.
CHP$V_OPER	OPER was used to gain the requested access.
CHP$V_GRPNAM	GRPNAM was used to gain the requested access.
CHP$V_SYSNAM	SYSNAM was used to gain the requested access.
CHP$V_GROUP	GROUP was used to gain the requested access.
CHP$V_WORLD	WORLD was used to gain the requested access.
CHP$V_PRMCEB	PRMCEB was used to gain the requested access.
CHP$V_UPGRADE	UPGRADE was used to gain the requested access.
CHP$V_DOWNGRADE	DOWNGRADE was used to gain the requested access.

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



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 CHP$_PROT item in increments of words; if a short buffer is given, zeros are assumed for the remainder.

The $CHKPRO service compares the low-order four bits of CHP$_ACCESS against one of the 4-bit fields in the low-order word of CHP$_PROT, the next four bits of CHP$_ACCESS against one of the 4-bit fields in the next word of CHP$_PROT, and so on. The $CHKPRO service chooses a field of CHP$_PROT based on the privileges specified for the accessor (CHP$_PRIV), the UICs of the accessor (CHP$_RIGHTS or CHP$_ADDRIGHTS, or both), and the object's owner (CHP$_OWNER).

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

CHP$_RIGHTS
A vector that points to an accessor's rights list. The accessor's UIC is the identifier of the first entry in the rights list. The accessor's rights list consists of the rights list specified by CHP$_RIGHTS and, optionally, the rights list specified by the CHP$_ADDRIGHTS item codes.

CHP$_UIC
A longword specifying the accessor's owner UIC. This item code may be used to avoid having to pass an entire rights list segment via the CHP$_RIGHTS item code. If CHP$_RIGHTS and then CHP$_UIC are specified, in that order, $CHKPRO initializes the local rights list and then replaces just the owner UIC with the value of CHP$_UIC.

Description

The Check Access Protection service determines whether an accessor with the specified rights and privileges can access an object with the specified attributes. The service invokes the system's access protection check, which permits layered products and other subsystems to build protected structures that are consistent with the protection facilities provided by the base operating system. The service also allows a privileged subsystem to perform protection checks on behalf of a requester.

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

The item list arguments accepted by this service permit you to specify the protection of the object being accessed, the rights and privileges of the accessor, and the type of access desired.

At 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 may also request that an object access audit be performed if security auditing has been enabled for the object class or if auditing ACEs are contained in the object's ACL. The CHP$V_AUDIT flag requests an access audit. This

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

Normally, $CHKPRO generates an object access audit when an audit is required. The caller may specify the CHP$V_CREATE flag to force an object creation audit instead of an object access audit. Similarly, the CHP$V_DELETE flag forces an object deletion audit. The CHP$_AUDIT_LIST item code may be used to specify additional information to be included in any resulting audit records.

Required Access or Privileges

AUDIT privilege is required when requesting an audit.

Required Quota

None

Related Services

$AUDIT_EVENT, $CHECK_ACCESS, $CREATE_USER_PROFILE, $FORMAT_ACL

Condition Values Returned

SS$_NORMAL	The service completed successfully; the desired access is granted.
SS$_ACCVIO	The item list cannot be read by the caller, or one of the buffers specified in the item list cannot be written by the caller.
SS$_ACLFULL	More than 20 CHP$_ACL items were given.
SS$_BADPARAM	The argument is invalid.
SS$_BUFFEROVF	The output buffer is too small and the protection check succeeded.
SS$_IVACL	You supplied an invalid ACL segment with the CHP$_ACL item.
SS$_IVBUFLEN	The output buffer is too small and the protection check failed.
SS$_NOAUDIT	Caller lacks privilege to request audit.
SS$_NOPRIV	The desired access is not granted.
SS$_RIGHTSFULL	More than 11 CHP$_ADDRIGHTS items were given.

$CLRCLUEVT (Alpha Only) Clear Cluster Event

On Alpha systems, removes one or more notification requests previously established by a call to SYS$SETCLUEVT.

Format

SYS$CLRCLUEVT [handle] ,[acmode] ,[event]

Arguments

handle

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

Identification of the AST request to be canceled. The **handle** argument uniquely identifies the request and is returned when the $SETCLUEVT service is called.

acmode

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

Access mode of the cluster configuration event to be canceled. The **acmode** argument is a longword containing the access mode.

Each access mode has a symbolic name. The $PSLDEF macro defines the following symbols for the four access types.

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

event

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

Event code indicating the type of cluster configuration event for which an AST is no longer to be delivered. The **event** argument is a value indicating which type of event is no longer of interest.

Each event type has a symbolic name. The $CLUEVTDEF macro defines the following symbolic names.

Symbolic Name	Description
CLUEVT$C_ADD	One or more OpenVMS nodes have been added to the VMScluster system.
CLUEVT$C_REMOVE	One or more OpenVMS nodes have been removed from the VMScluster 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

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

$CMEXEC
Change to Executive Mode

Changes the access mode of the calling process to executive mode.

Format

SYS$CMEXEC routin ,[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.

Alpha

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.

Alpha

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

VAX

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.♦

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 **routin** argument executes in kernel mode, not executive mode.

Required Access or Privileges

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

Required Quota

None

Related Services

None

Condition Values Returned

SS$_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

Arguments

routin_64

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

Routine to be executed while the process is in executive mode. The **routin_64** argument is the 32-bit or 64-bit address of this routine.

arglst_64

OpenVMS usage:	arg_list
type:	quadword (unsigned)
access:	read only
mechanism:	by 32-bit 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-bit or 64-bit address of this argument list.

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 non-zero, must contain an accessible virtual address for an argument list, the first quadword of which must be a number between 0 and 255 specifying the number of quadwords that follow it on the list.

Description

The Change to Executive Mode with Quadword Argument List service allows a process to change its access mode to executive, execute a specified routine, and then return to the access mode in effect before the call was issued.

The $CMEXEC_64 service uses standard procedure calling conventions to pass control to the specified routine.

When you use the $CMEXEC_64 service, the system modifies the registers before entry into the target routine. The specified routine must exit with a RET instruction.

All of the Change Mode system services are intended to allow for the execution of a routine at an access mode more (not less) privileged than the access mode from which the call is made. If $CMEXEC_64 is called while a process is executing in kernel mode, the routine specified by the **routin_64** argument executes in kernel mode, not executive mode.

Required Access or Privileges

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

Required Quota

None.

Related Services

$CMEXEC, $CMKRNL, $CMKRNL_64

Condition Values Returned

SS$_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]

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.

Alpha

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 Kernel Mode 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 service uses standard procedure calling conventions to pass control to the specified routine.

Alpha

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

VAX

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. Programs should not use registers R2 through R11 to pass context between the calling and called procedures.♦

On Alpha and VAX systems, when you use the $CMKRNL 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.

The system loads R4 with the address of the process control block (PCB).

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

None

Condition Values Returned

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

$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

Arguments

routin_64

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

Routine to be executed while the process is in kernel mode. The **routin_64** argument is the 32-bit or 64-bit address of this routine.

arglst_64

OpenVMS usage:	arg_list
type:	quadword (unsigned)
access:	read only
mechanism:	by 32-bit 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-bit or 64-bit address of this routine.

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 non-zero, must contain an accessible virtual address for an argument list, the first quadword of which must be a number between 0 and 255 specifying the number of quadwords that follow it on the list.

Description

The Change to Kernel Mode with Quadword Argument List service allows a process to change its access mode to kernel, execute a specified routine, and then return to the access mode in effect before the call was issued.

The $CMKRNL_64 service uses standard procedure calling conventions to pass control to the specified routine.

When you use the $CMKRNL_64 service, the system modifies the registers before entry into the target routine. The system loads R4 with the address of the process control block (PCB). The specified routine (if programmed in MACRO-32) must exit with a RET instruction.

Required Access or Privileges

To call the $CMKRNL_64 service, a process must either have CMKRNL privilege or be currently executing in executive or kernel mode.

Required Quota

None.

Related Services

$CMEXEC, $CMEXEC_64, $CMKRNL

Condition Values Returned

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

$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]

Arguments

cpu_id

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

Identifier of the CPU whose user capability mask is to be modified or returned. The **cpu_id** argument is a longword containing this number, which is in the supported range of individual CPUs from 0 to SYI$_MAX_CPUS –1.

Specifying the constant CAP$K_ALL_ACTIVE_CPUS applies the current modification operation to all CPUs currently in the active set, and to the default CPU initialization context in SCH$GL_DEFAULT_CPU_CAP. If the **prev_mask** argument is also supplied, the previous default CPU initialization context in SCH$GL_DEFAULT_CPU_CAP will be returned rather than any specific CPU state.

To modify only the user capabilities in SCH$GL_DEFAULT_CPU_CAP, the **flags** argument has a bit constant CAP$M_FLAG_DEFAULT_ONLY. When this bit is set, all service operations are performed on the global cell rather than on an individual CPU specified in the **cpu_id** argument. This bit does not supersede the CAP$K_ALL_ACTIVE_CPUS constant, however. If both constants are specified, CAP$K_ALL_ACTIVE_CPUS take precedence; nevertheless, the operations to SCH$GL_DEFAULT_CPU are identical because that function is a direct subset of the other.

select_mask

OpenVMS usage:	mask_quadword
type:	quadword (unsigned)
access:	read only
mechanism:	by 32-bit 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-bit or 64-bit address of a quadword bit vector wherein a bit, when set, specifies that the corresponding user capability is to be modified.

The individual user capability bits in **select_mask** can be referenced by their symbolic constant names, CAP$M_USER1 through CAP$M_USER16. These constants (not zero-relative) specify the position in the mask quadword that corresponds to the bit name. Multiple capabilities can be selected by ORing together the appropriate bits.

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-bit or 64-bit reference

Mask specifying the settings for those capabilities selected in the **select_mask** argument. The **modify_mask** argument is the 32-bit or 64-bit address of a quadword bit vector wherein a bit, when set, specifies that the corresponding user capability is to be added to the specified 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 ORing together the appropriate bits.

To add a specific user capability to the specified CPU, that bit position must be set in both **select_mask** and **modify_mask**. To remove a specific user capability from the specified CPU, that bit position must be set in **select_mask** and clear in **modify_mask**.

The symbolic constant CAP$K_ALL_USER_ADD, when specified in **modify_mask**, indicates that all capabilities specified in **select_mask** are to be added to the current user capability set. The constant CAP$K_ALL_USER_REMOVE indicates that all capabilities specified are to be cleared from the set.

prev_mask

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

Options selected for the user capability modification. The **flags** argument is a quadword bit vector wherein a bit corresponds to an option. Only the bits specified below are used; the remainder of the quadword bits are reserved and must be 0.

Each option (bit) has a symbolic name, defined by the $CAPDEF macro. The **flags** argument is constructed by performing a logical OR operation using the symbolic names of each desired option. The following table describes the symbolic name of each option:

Symbolic Name	Description
CAP$M_FLAG_DEFAULT_ONLY	Indicates that the specified operations are to be performed on the global context cell instead of on a specific CPU. This bit supersedes any individual CPU specified in **cpu_id** but does not override the all active set behavior (CAP$K_ALL_ACTIVE_CPUS). Specifying this bit constant applies this operation to the default startup capabilities for all CPUs booted for the first time.
CAP$M_FLAG_CHECK_CPU	Determines whether the kernel thread can be left in a non-runnable state under some circumstances. No operation of this service will allow a transition from a runnable to blocked state; however, if the kernel thread is already at a blocked state, this bit determines whether the result of the operation must leave it runnable. If CAP$M_FLAG_CHECK_CPU is set or **flags** is not specified, the kernel thread will be checked to ensure it can safely run on one of the CPUs in the active set. If CAP$M_FLAG_CHECK_CPU is not set, any state operations on kernel threads already in a blocked state will be 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, then **select_mask** must be specified as an argument. If **modify_mask** is not specified, then no modifications are made to the user capability mask for the specified CPU. In this case, **select_mask** is ignored. If **prev_mask** is not specified, then no previous mask is returned.

No service state changes that will place any currently runnable kernel thread into a blocked state will be allowed.

If CAP$K_ALL_ACTIVE_CPUS is specified in **cpu_id**, the user capability modifications are performed on all CPUs currently in the active set, as well as the global initialization cell. If the bit constant CAP$M_FLAG_DEFAULT_ONLY is set in the **flags** argument, the user capability modifications are made only to

the global initialization cell, regardless of what individual CPU is specified in **cpu_id**.

Required Access or Privileges

The caller must have both ALTPRI and WORLD privileges to call SYS$CPU_CAPABILITIES to modify CPU user capabilities.

No privileges are required if SYS$CPU_CAPABILITIES is called only to retrieve the current user capabilities mask from the specified CPU or global default.

Related Services

$PROCESS_CAPABILITIES

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_BADPARAM	One of more arguments has an invalid value or the specified CPU is not in the configuration.
SS$_ACCVIO	The service cannot access the locations specified by one or more arguments.
SS$_NOPRIV	Insufficient privilege for attempted operation.
SS$_CPUCAP	Attempted operation would place one or more processes in an unrunnable state.
SS$_INSFARG	Fewer than the required number of arguments were specified or no operation was specified.

$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

Arguments

start_va_64

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

Starting virtual address of the pages to be included in the buffer object. The specified virtual address will be rounded down to a CPU-specific page boundary.

The virtual address space must already exist.

length_64

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

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

acmode

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

Access mode on behalf of which the request is being made. The **acmode** argument is a longword containing the access mode.

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

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

The most privileged access mode used is the access mode of the caller. For the $CREATE_BUFOBJ_64 service to complete successfully, the resultant access mode must be equal to or more privileged than the access mode already associated with the pages in the specified input range.

flags

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

Flag mask specifying the request options. The **flags** argument is a longword bit vector in which each bit corresponds to a flag. The $CBODEF macro in STARLET.MLB and CBODEF.H file in SYS$STARLET_C.TLB define a symbolic name for each flag. The following table describes each flag that is valid for the $CREATE_BUFOBJ_64 service:

Flag	Value	Description
CBO$M_RETSVA	1	If set, return the system virtual address in the **return_va_64** argument instead of the process virtual address range. (Valid for inner mode callers only.)
CBO$M_SVA_32	4	If set, create the buffer object window in 32-bit S0/S1 space. (By default, this service will create the window in 64-bit S2 space.)

return_va_64

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

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

return_length_64

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

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

buffer_handle_64

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

The 32-bit or 64-bit virtual address of a naturally aligned quadword into which a buffer handle is returned to be used when referencing the created buffer object.

Description

The Create Buffer Object service creates a buffer object for use by the I/O subsystem. The pages that constitute the buffer object are permanently locked into physical memory (but not the process's working set) and double mapped into system space. The net effect is:

- I/O can be initiated to or from the buffer without the need to probe or lock the buffer pages.
- The process is still fully swappable.

If the condition value SS$_ACCVIO is returned by this service, a value *cannot* be returned in the memory locations pointed to by the **return_va_64**, **return_length_64**, and **buffer_handle_64** arguments.

If a condition value other than SS$_ACCVIO is returned, the returned address and returned length indicate the pages that were successfully made part of the buffer object before the error occurred. If no pages were made part of the buffer object, the **return_va_64** argument will contain the value -1, and a value is *not* returned in the memory location pointed to by the **return_length_64** argument.

Required Privileges

No privileges are required if calling $CREATE_BUFOBJ_64 from an inner mode. If calling from user mode, the process must hold the rights identifier VMS$BUFFER_OBJECT_USER at the time of the call. This identifier is normally granted by the system administrator via the AUTHORIZE utility.

Required Quota

No process quota is charged but the pages are charged against a systemwide limit, system parameter MAXBOBMEM.

Related Services

$CRETVA_64, $DELETE_BUFOBJ, $EXPREG_64

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The **return_va_64**, **return_length_64**, or **buffer_handle_64** argument cannot be written by the caller.
SS$_BADPARAM	Invalid flags options specified.
SS$_EXBUFOBJLM	Buffer object cannot be created because it would bring the total number of buffer object pages above the systemwide limit MAXBOBMEM.
SS$_INSFMEM	Insufficient dynamic memory.
SS$_INSFSPTS	Insufficient system page table entries.
SS$_NOBUFOBJID	The process attempted to create a buffer object from user mode but was not holding required rights identifier VMS$BUFFER_OBJECT_USER.
SS$_NOPRIV	Valid flag options were specified but from user mode.
SS$_PAGNOTWRITE	A page within the address range is not writeable.

SS$_PAGOWNVIO	The pages could not put into the buffer object because the access mode associated with the call to $CREATE_BUFOBJ_64 was less privileged than the access mode associated with the pages.

$CREATE_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]

Arguments

gs_name_64

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

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

ident_64

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

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

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

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

If you specify the **ident_64** argument as 0, the version number and match control fields default to 0.

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

file_offset_64

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

Byte offset into the file that marks the beginning of the section. The **file_offset_64** argument is a quadword containing this number. If you do not specify the **file_offset_64** argument or specify it as 0, the section is created beginning with the first byte in the file.

The **file_offset_64** argument must be a multiple of virtual disk blocks.

length_64

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

Length, in bytes, of the global disk file section to be created. The length specified must be 0 or a multiple of virtual disk blocks. If the length specified is 0 or extends beyond end-of-file (EOF), the global disk file section is created up to and including the virtual block number that contains EOF.

chan

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

Number of the channel on which the file has been accessed. The **chan** argument is a longword containing this number. The access mode at which the channel was opened must be equal to or less privileged than the access mode of the caller.

You can use the OpenVMS Record Management Services (RMS) macro $OPEN to access a file; the file options parameter in the file access block must indicate a user file open (UFO keyword).

acmode

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

Access mode that is to be the owner of the pages created during the mapping. The **acmode** argument is a longword containing the access mode.

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

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

flags

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

Flag mask specifying the type of global section to be created as well as its characteristics. The **flags** argument is a longword bit vector in which each bit corresponds to a flag. The $SECDEF macro and the SECDEF.H file define a symbolic name for each flag. You construct the **flags** argument by performing a logical OR operation on the symbol names for all desired flags. The following table describes each flag that is valid for the $CREATE_GFILE service:

Flag	Description
SEC$M_CRF	Pages are copy-on-reference. By default, pages are shared.
SEC$M_DZRO	Pages are demand-zero pages. By default, they are not zeroed when copied. Note that SEC$M_DZRO and SEC$M_CRF cannot both be set and that SEC$M_DZRO set and SEC$M_WRT clear is an invalid combination.
SEC$M_GBL	Pages form a global section. By default, this flag is always present in this service and cannot be disabled.
SEC$M_PERM	Pages are 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 Digital and should be specified as 0. The condition value SS$_IVSECFLG is returned if any undefined bits are set or if an illegal combination of flags is set.

return_length_64

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

The length of the global section created. The **return_length_64** argument is the 32-bit or 64-bit virtual address of a naturally aligned quadword into which the service returns the length of the 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 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.

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

In order to create a global section, the process must have the following privileges:

- SYSGBL privilege to create a system global section (if flag SEC$M_SYSGBL is set)
- PRMGBL privilege to create a permanent global section

Required Quota

None.

Related Services

$CRMPSC, $CRMPSC_GFILE_64, $DGBLSC, $MGBLSC, $MGBLSC_64

Condition Values Returned

SS$_CREATED	The service completed successfully. The specified global section did not previously exist and has been created.
SS$_ACCVIO	The **gs_name_64** argument or the **return_length_64** argument cannot be read by the caller.
SS$_CHANVIO	The specified channel was assigned from a more privileged access mode.

SS$_DUPLNAM	A global section of the same name already exists; a new global section was not created.
SS$_ENDOFFILE	The **file_offset_64** argument specified is beyond the logical end-of-file.
SS$_EXBYTLM	Process has exceeded the byte count quota; the system was unable to map the requested file.
SS$_GPTFULL	There is no more room in the system global page table to set up page table entries for the section.
SS$_GSDFULL	There is no more room in the system space allocated to maintain control information for global sections.
SS$_IVCHAN	An invalid channel number was specified; the channel number specified was 0 or a channel that is unassigned.
SS$_IVCHNLSEC	The channel number specified is currently active, or there are no files opened on the specified channel.
SS$_IVIDENT	An invalid channel number was specified; the channel number specified is larger than the number of channels available.
SS$_IVLOGNAM	The specified global section name has a length of 0 or has more than 43 characters.
SS$_IVLVEC	The specified section was not installed using the /PROTECT qualifier.
SS$_IVSECFLG	An invalid flag, a reserved flag, or an invalid combination of flags was specified.
SS$_IVSECIDCTL	The match control field of the global section identification is invalid.
SS$_LEN_NOTBLKMULT	The **length_64** argument is not a multiple of virtual disk blocks.
SS$_NOPRMGBL	The process does not have the privileges to create or delete a permanent group global section (PRMGBL).
SS$_NOSYSGBL	The process does not have the privileges to create or delete a system global section (SYSGBL).
SS$_NOTFILEDEV	The device is not a file-oriented, random-access, or directory device.
SS$_NOWRT	The file is read-only, and the flag bit SEC$M_CRF is not set.
SS$_OFF_NOTBLKALGN	The **file_offset_64** argument is not a multiple of virtual disk blocks.
SS$_SECTBLFUL	There are no entries available in the system global section table.
SS$_TOOMANYLNAM	The logical name translation of the **gs_name_64** argument exceeded the allowed depth of 10.

$CREATE_GPFILE (Alpha Only) Create Permanent Global Page File

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

This service accepts 64-bit addresses.

Format

SYS$CREATE_GPFILE gs_name_64 ,ident_64 ,prot ,length_64 ,acmode ,flags

Arguments

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

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

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

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

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

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

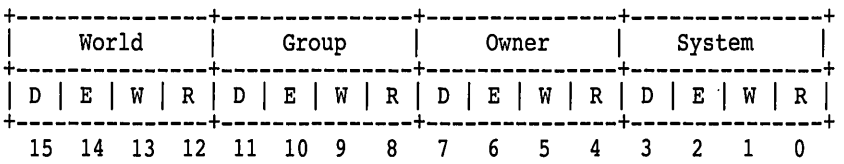
If you specify the **ident_64** argument as 0, the version number and match control fields default to 0.

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

prot

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

Protection to be applied to the global page file section. The mask contains four 4-bit fields. Bits are read from right to left in each field. The following diagram depicts the mask:



Cleared bits indicate that read, write, execute, and delete access, in that order, are granted to the particular category of user. Only read, write, and execute access are meaningful for section protection. Delete access bits are ignored. Read access also grants execute access for those situations where execute access applies. If 0 is specified, read access and write access are granted to all users.

length_64

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

Length, in bytes, of the global page file section to be created. The **length_64** argument must be specified as a multiple of the CPU-specific page size. A length of 0 cannot be specified.

acmode

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

Access mode that is to be the owner of the pages created during the mapping. The **acmode** argument is a longword containing the access mode.

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

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

flags

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

Flag mask specifying the type of global section to be created as well as its characteristics. The **flags** argument is a longword bit vector in which each bit corresponds to a flag. The $SECDEF macro and the SECDEF.H file define a symbolic name for each flag. You construct the **flags** argument by performing a logical OR operation on the symbol names for all desired flags. The following table describes the flags that are valid for the $CREATE_GPFILE service:

Flag	Description
SEC$M_DZRO	Pages are demand-zero pages.
SEC$M_GBL	Pages form a global section. By default, this flag is always present in this service and cannot be disabled.
SEC$M_PAGFIL	Pages form a global page-file section. SEC$M_PAGFIL also implies SEC$M_WRT and SEC$M_DZRO. By default, this flag is always present in this service and cannot be disabled.
SEC$M_PERM	Pages are permanent. By default, this flag is always present in this service and cannot be disabled.
SEC$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 Digital and should be specified as 0. The condition value SS$_IVSECFLG is returned if any undefined bits are set.

Description

The Create Permanent Global Page File Section service allows a process to create a permanent global page file section. Global page file sections contain demand-zero allocation pages that are writable and backed up by the system page file. All pages in the global page file section are shared by all processes that map to the global section.

Required Privileges

In order to create a permanent global page file section, the process must have the following privileges:

- SYSGBL privilege to create a system global section (if flag SEC$M_SYSGBL is set)
- PRMGBL privilege to create a permanent global section

Required Quota

The systemwide number of global page file pages is limited by the system parameter GBLPAGFIL.

Related Services

$CRMPSC, $CRMPSC_GPFILE_64, $DGBLSC, $MGBLSC, $MGBLSC_64

Condition Values Returned

SS$_CREATED	The service completed successfully. The specified global section did not previously exist and has been created.
SS$_ACCVIO	The **gs_name_64** descriptor cannot be read by the caller.
SS$_DUPLNAM	A global section of the same name already exists; a new global section was not created.
SS$_GPTFULL	There is no more room in the system global page table to set up page table entries for the section.
SS$_GSDFULL	There is no more room in the system space allocated to maintain control information for global sections.
SS$_IVLOGNAM	The specified global section name has a length of 0 or has more than 43 characters.
SS$_IVSECFLG	An invalid flag, a reserved flag, or an invalid combination of flags was specified.
SS$_IVSECIDCTL	The match control field of the global section identification is invalid.
SS$_LEN_NOTPAGMULT	The **length_64** argument is not a multiple of CPU-specific pages or was specified as 0.
SS$_NOPRMGBL	The process does not have the privileges to create or delete a permanent group global section (PRMGBL).
SS$_NOSYSGBL	The process does not have the privileges to create or delete a system global section (SYSGBL).
SS$_SECTBLFUL	There are no entries available in the system global section table.
SS$_TOOMANYLNAM	The logical name translation of the **gs_name_64** argument exceeded the allowed depth of 10.

$CREATE_GPFN (Alpha Only) Create 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

Arguments

gs_name_64

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

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

ident_64

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

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

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

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

If you specify the **ident_64** argument as 0, the version number and match control fields default to 0.

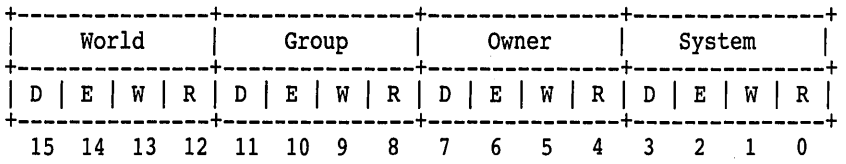
The version number is in the second longword. The version number contains two fields: a minor identification in the low-order 24 bits and a major identification in the high-order 8 bits. You can assign values for these fields by installation convention to differentiate versions of global sections. If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

prot

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

Protection to be applied to the global page frame section.

The mask contains four 4-bit fields. Bits are read from right to left in each field. The following diagram depicts the mask:



Cleared bits indicate that read, write, execute, and delete access, in that order, are granted to the particular category of user. Only read, write, and execute access are meaningful for section protection. Delete access bits are ignored. Read access also grants execute access for those situations where execute access applies. If zero is specified, read access and write access are granted to all users.

start_pfn

OpenVMS usage:	page frame number
type:	longword (unsigned)
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.

acmode

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

Access mode that is to be the owner of the pages created during the mapping. The **acmode** argument is a longword containing the access mode.

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

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

flags

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

Flag mask specifying the characteristics of the page frame section to be created. The **flags** argument is a longword bit vector in which each bit corresponds to a flag. The $SECDEF macro and the SECDEF.H file define a symbolic name for each flag. You construct the **flags** argument by performing a logical OR operation on the symbol names for all desired flags. The following table describes the flags that are valid for the $CREATE_GPFN service:

Flag	Description
SEC$M_GBL	Pages form a global section. By default, this flag is always present in this service and cannot be disabled.
SEC$M_PERM	Pages are permanent. By default, this flag is always present in this service and cannot be disabled.
SEC$M_PFNMAP	Pages form a page frame section. By default, this flag is always present in this service and cannot be disabled.
SEC$M_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 Digital and should be specified as 0. The condition value SS$_IVSECFLG is returned if any undefined bits are set or if an illegal combination of flags is set.

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

In order to create a permanent global page frame section, the process must have the following privileges:

- SYSGBL privilege to create a system global section. (if flag SEC$M_SYSGBL is set)
- PRMGBL privilege to create a permanent global section.
- PFNMAP privilege to create a page frame section.

Required Quota

None.

Related Services

$CRMPSC, $CRMPSC_GPFN_64, $DGBLSC, $MGBLSC, $MGBLSC_GPFN_64

Condition Values Returned

SS$_CREATED	The service completed successfully. The specified global section did not previously exist and has been created.
SS$_ACCVIO	The **gs_name_64** argument cannot be read by the caller.
SS$_DUPLNAM	A global section of the same name already exists; a new global section was not created.
SS$_GPTFULL	There is no more room in the system global page table to set up page table entries for the section.
SS$_GSDFULL	There is no more room in the system space allocated to maintain control information for global sections.
SS$_IVLOGNAM	The specified global section name has a length of 0 or has more than 43 characters.
SS$_IVSECFLG	An invalid flag, a reserved flag, or an invalid combination of flags was specified.
SS$_IVSECIDCTL	The match control field of the global section identification is invalid.
SS$_NOPRMGBL	The process does not have the privileges to create or delete a permanent group global section (PRMGBL).
SS$_NOSYSGBL	The process does not have the privileges to create or delete a system global section (SYSGBL).
SS$_TOOMANYLNAM	The logical name translation of the **gs_name_64** argument exceeded the allowed depth of 10.

$CREATE_RDB
Create Rights Database

Initializes a rights database.

Format

SYS$CREATE_RDB [sysid]

Argument

sysid

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

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

Description

The Create Rights Database service initializes a rights database. The database name is the file equated to the logical name RIGHTSLIST, which must be defined as a system logical name from executive mode. If the logical name does not exist, the database is created in SYS$COMMON:[SYSEXE] with the file name RIGHTSLIST.DAT. If the database already exists, $CREATE_RDB fails with the error RMS$_FEX.

The rights database is created with an owner of [1,4] and a protection of (RWED, RWED, R).

Required Access or Privileges

Write access to the directory in which the file is being created is required.

Required Quota

None

Related Services

$ADD_HOLDER, $ADD_IDENT, $ASCTOID, $CHANGE_ACL, $FIND_HELD, $FIND_HOLDER, $FINISH_RDB, $FORMAT_ACL, $GRANTID, $IDTOASC, $MOD_HOLDER, $MOD_IDENT, $PARSE_ACL, $REM_HOLDER, $REM_IDENT, $REVOKID

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The **sysid** argument cannot be read by the caller.

SS$_INSFMEM	The process dynamic memory is insufficient for opening the rights database.
RMS$_FEX	A rights database already exists. To create a new one, you must explicitly delete or rename the old one.
RMS$_PRV	The user does not have write access to SYS$SYSTEM.

Because the rights database is an indexed file accessed with OpenVMS RMS, this service can also return RMS status codes associated with operations on indexed files. For descriptions of these status codes, refer to the *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. Virtual regions can be created only within the process's P2 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]

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.

region_prot

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

Region protection associated with the call to $CREATE_REGION_64. The **region_prot** argument is a longword containing the create and owner mode.

The file VADEF.H in SYS$STARLET_C.TLB and the $VADEF macro in STARLET.MLB define the following symbols for valid combinations of create and owner modes:

Symbol	Create and Owner Modes
VA$C_REGION_UCREATE_UOWN	User create mode and user owner mode
VA$C_REGION_UCREATE_SOWN	User create mode and supervisor owner mode
VA$C_REGION_UCREATE_EOWN	User create mode and executive owner mode
VA$C_REGION_UCREATE_KOWN	User create mode and kernel owner mode
VA$C_REGION_SCREATE_SOWN	Supervisor create mode and supervisor owner mode
VA$C_REGION_SCREATE_EOWN	Supervisor create mode and executive owner mode
VA$C_REGION_SCREATE_KOWN	Supervisor create mode and kernel owner mode

Symbol	Create and Owner Modes
VA$C_REGION_ECREATE_EOWN	Executive create mode and executive owner mode
VA$C_REGION_ECREATE_KOWN	Executive create mode and kernel owner mode
VA$C_REGION_KCREATE_KOWN	Kernel create mode and kernel owner mode

For both create and owner mode, the $CREATE_REGION_64 service uses whichever of the following two access modes is least privileged:

- The access mode specified by the **acmode** argument.
- The access mode of the caller.

A subsequent call to any system service that created address space within a region must be made from an access mode that is the same or more privileged than the create mode associated with the region.

A subsequent call to $DELETE_REGION_64 to delete the region must be made from an access mode that is the same or more privileged than the owner mode associated with the region.

All regions created by $CREATE_REGION_64 are automatically deleted when the image is run down on image exit.

flags

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

Flag mask specifying the characteristics of the region to be created. The **flags** argument is a longword bit vector in which each bit corresponds to a flag. The file VADEF.H in SYS$STARLET_C.TLB and the $VADEF macro in STARLET.MLB define a symbolic name for each flag. You construct the **flags** argument by performing a logical OR operation on the symbol names for all desired flags.

The following table describes the flag that is valid for the $CREATE_REGION_64 service:

Flag	Description
VA$M_DESCEND	Created region is a descending region; that is, allocation occurs toward decreasing virtual addresses. If VA$M_DESCEND is not specified, the region allocation occurs toward increasing virtual addresses.

All other bits in the **flags** argument are reserved for future use by Digital. The condition value SS$_IVREGFLG is returned if any undefined bits are set.

return_region_id_64

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

The region ID associated with the created region. The **return_region_id_64** argument is the 32-bit 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-bit or 64-bit reference

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

return_length_64

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

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

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 P2 address space.

Description

The Create Virtual Region service is a kernel mode service that can be called from any mode. This service allows a process to create a virtual region within its P2 private address space. Virtual regions in P0 and P1 space are not supported.

The Create Virtual Region service creates the virtual region on a page-aligned boundary.

The Create Virtual Region service returns the lowest virtual address within the region whether the region expands toward higher or lower virtual addresses.

If the **start_va_64** argument is not specified by the caller, no assumptions should be made about the relative placement of the region within the overall process address space.

If the returned value of the service is not a successful condition value, a value *cannot* be 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_FILE_64, $CRMPSC_GFILE_64, $CRMPSC_GPFILE_64, $CRMPSC_GPFN_64, $CRMPSC_PFN_64, $DELETE_REGION_64, $DELTVA_64, $EXPREG_64, $GET_REGION_INFO, $MGBLSC_64, $MGBLSC_GPFN_64

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The **return_region_id_64** argument, the **return_va_64** argument, or the **return_length_64** argument cannot be written by the caller.
SS$_IVREGFLG	One or more of the reserved bits in the **flags** argument is set.
SS$_LEN_NOTPAGMULT	The **length_64** argument is not a multiple of CPU-specific pages.
SS$_VASFULL	The process private address space is full or no space is available in the process private address space for a region of the specified size.
SS$_VA_IN_USE	A page in the specified virtual address range is within another virtual region or is otherwise inaccessible.
SS$_VA_NOTPAGALGN	The **start_va_64** argument is not CPU-specific page-aligned.

$CREATE_USER_PROFILE
Create User Profile

Returns an encoded security profile for the specified user.

Format

SYS$CREATE_USER_PROFILE usrnam ,[itmlst] ,[flags] ,usrpro ,usrprolen ,[contxt]

Arguments

usrnam

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

Name of the user whose security profile is to be returned. The **usrnam** argument is the address of a descriptor pointing to a text string containing the user name. The user name string can contain a maximum of 12 alphanumeric characters.

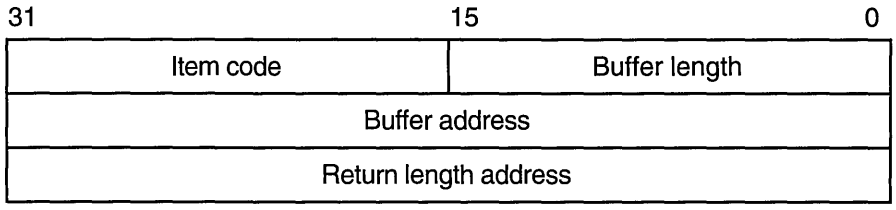
For more information about user names, see the *OpenVMS Guide to System Security*.

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

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 upon the item code specified in the item code field of the item descriptor.
Item code	A word containing a user-supplied symbolic code specifying the item of information.
Buffer address	A longword containing the user-supplied address of the buffer.
Return length address	A longword that normally contains the user-supplied address of a word in which the service writes the length (in bytes) of the information it returned. This is not used by $CREATE_USER_PROFILE and should contain a 0.

flags

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

The **flags** argument is used for controlling the behavior of the $CREATE_USER_PROFILE service. The following table describes each flag.

Symbol	Description
CHP$M_DEFCLASS	By default, $CREATE_USER_PROFILE initializes the security profile with the user's maximum authorized classification. When this flag is set, the service initializes the security profile from the user's default classification instead. This flag is reserved to Digital.
CHP$M_DEFPRIV	By default, $CREATE_USER_PROFILE initializes the security profile with the user's authorized privilege mask. When this flag is set, the service initializes the security profile from the user's default privilege mask instead.
CHP$M_NOACCESS	Instructs the service not to access the user authorization file (SYSUAF.DAT) or rights database (RIGHTSLIST.DAT) to build the security profile. This flag can be used as an optimization when all the information necessary to build the security profile is known to the caller.

usrpro

OpenVMS usage: char_string
type: opaque byte stream
access: write only
mechanism: by descriptor

Buffer to receive the security profile. The **usrpro** argument is the address of a buffer to receive the encoded security profile. If an address of 0 is specified, $CREATE_USER_PROFILE returns the size of the buffer needed in the **usrprolen** argument.

usrprolen

OpenVMS usage:	word
type:	word (unsigned)
access:	read/write
mechanism:	by reference

Word to receive the full size of the security profile. On input, the **usrprolen** argument specifies the length of the buffer pointed to by the **usrpro** argument. The **usrprolen** argument is the address of a word to which $CREATE_USER_PROFILE writes the actual length of the security profile. If the caller specifies a **usrpro** address of 0, $CREATE_USER_PROFILE returns the anticipated size, in bytes, of the buffer needed to hold the user's security profile in the **usrprolen** argument.

contxt

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

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

Using the **contxt** argument keeps the UAF open across all calls, thereby improving the performance of the system on subsequent calls. To close the UAF, you must run down the image.

The resulting context value from a $CREATE_USER_PROFILE call may also be used as the input **contxt** argument to the $GETUAI system service, and vice versa.

Item Codes

CHP$_ADDRIGHTS

A rights list segment containing additional identifiers to be appended to the set of identifiers held by the user. A rights list segment is a list of quadword identifier/attributes pairs, each containing a longword identifier value, followed by a longword mask identifying the attributes of the holder. The **buflen** argument should be set to the total size, in bytes, of the rights list segment. The **bufadr** argument points to a descriptor that points to the first byte in the rights list segment (that is, the first byte of the first identifier value).

This item code can be repeated to add up to 256 additional rights list segments. If more than 256 identifiers are granted to the user, $CREATE_USER_PROFILE returns SS$_INSFMEM.

CHP$_CLASS

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

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).

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 may be specified for the **usrnam** argument.

In either case, the newly created security profile may be passed as input to the $CHKPRO and $CHECK_ACCESS system services using the **usrpro** argument.

$CREATE_USER_PROFILE returns the set of identifiers associated with the user's owner identifier. The CHP$_ADDRIGHTS item code can be used to add additional identifiers to this set.

Required Access or Privileges
Access to SYSUAF.DAT and RIGHTSLIST.DAT is required unless you are constructing the security profile for your own user name.

Required Quota
None

Related Services
$CHECK_ACCESS, $CHKPRO, $FIND_HELD, $FINISH_RDB, $GETUAI

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	Parameter or item list buffer not accessible.
SS$_BADPARAM	Item code invalid.
SS$_INSFARG	A required item code or parameter is missing.
SS$_INSFMEM	Insufficient process memory to construct profile.

$CREATE_USER_PROFILE may 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.

Format

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

Arguments

attr

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

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

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

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

The attributes are as follows.

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

tabnam

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

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

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

lognam

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

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

acmode

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

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

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

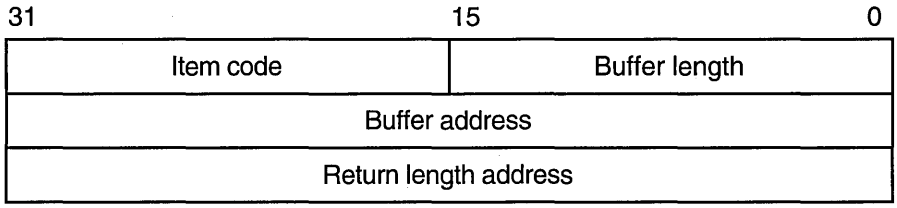
You cannot specify an access mode more privileged than that of the containing table. However, if the caller has SYSNAM privilege, then the specified access mode is associated with the logical name regardless of the access mode of the caller.

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

itmlst

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

Item list describing the equivalence names to be defined for the logical name and information to be returned to the caller. The **itmlst** argument is the address of a list of item descriptors, each of which specifies information about an equivalence name. The list of item descriptors is terminated by a longword of 0. 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 number of bytes in the buffer pointed to by the buffer address field. The length of the buffer needed depends upon the item code specified in the item code field of the item descriptor. If the value of buffer length is too small, the service truncates the data.
Item code	A word containing a symbolic code that describes the information in the buffer or the information to be returned to the buffer, pointed to by the buffer address field. The item codes are listed in the Item Codes section.
Buffer address	A longword containing the address of the buffer that receives or passes information.
Return length address	A longword containing the address of a word specifying the actual length in bytes of the information returned by $CRELNM in the buffer pointed to by the buffer address field. The return length address field is used only when the item code specified is LNM$_TABLE. Although this field is ignored for all other item codes, it must nevertheless be present as a placeholder in each item descriptor.

Item Codes

LNM$_ATTRIBUTES

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

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

LNM$_CHAIN

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

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

LNM$_STRING

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

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

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

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

LNM$_TABLE

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

This item code can appear anywhere in the item list.

Description

The Create Logical Name service creates a logical name and specifies its equivalence name. Note that logical names are case sensitive.

Required Access or Privileges

The calling process must have the following:

- Write access to shareable tables to create logical names in those tables
- GRPNAM privilege to enter a logical name into the group logical name table
- SYSNAM 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$_NORMAL	The service completed successfully; the logical name has been created.
SS$_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$_BADPARAM	One or more arguments have an invalid value, or a logical name table name or logical name was not specified.
SS$_DUPLNAM	An attempt was made to create a logical name with the same name as an already existing logical name, and the existing logical name was created at a more privileged access mode and with the LNM$M_NO_ALIAS attribute.
SS$_EXLNMQUOTA	The quota associated with the specified logical name table for the creation of the logical name is insufficient.
SS$_INSFMEM	The dynamic memory is insufficient for the creation of the logical name.
SS$_IVLOGNAM	The **tabnam** argument, the **lognam** argument, or the equivalence string specifies a string whose length is not in the required range of 1 through 255 characters. The **lognam** argument specifies a string whose length is not in the required range of 1 to 31 characters for directory table entries.

SS$_IVLOGTAB	The **tabnam** argument does not specify a logical name table.
SS$_NOLOGTAB	Either the specified logical name table does not exist or the logical name translation of the table name exceeded the allowable depth of 10 translations.
SS$_NOPRIV	The caller lacks the necessary privilege to create the logical name.

$CRELNT
Create Logical Name Table

Creates a process-private or shareable logical name table.

Format

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

Arguments

attr

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

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

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

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

The following table describes each attribute.

Attribute	Description
LNM$M_CONFINE	If set, the logical name table is not copied from the process to its spawned subprocesses. You create a subprocess with the DCL command SPAWN or the Run-Time Library LIB$SPAWN routine. You can specify this attribute only for process-private logical name tables; it is ignored for shareable tables. The state of this bit is also propagated from the parent table to the newly created table and can be overridden only if the parent table does not have the bit set. Thus, if the parent table has the LNM$M_CONFINE attribute, the newly created table will also have it, no matter what is specified in the **attr** argument. On the other hand, if the parent table does not have the LNM$M_CONFINE attribute, the newly created table can be given this attribute through the **attr** argument. The process-private directory table LNM$PROCESS_DIRECTORY does not have the LNM$M_CONFINE attribute.

Attribute	Description
LNM$M_CREATE_IF	If set, a new logical name table is created only if the specified table name is not already entered at the specified access mode in the appropriate directory table. If the table name exists, a new table is not created and no modification is made to the existing table name. This holds true even if the existing name has differing attributes or quota values, or even if it is not the name of a logical name table. If LNM$M_CREATE_IF is not set, the new logical name table will supersede any existing table name with the same access mode within the appropriate directory table. Setting this attribute is useful when two or more users want to create and use the same table but do not want to synchronize its creation.
LNM$M_NO_ALIAS	If set, the name of the logical name table cannot be duplicated at an outer access mode within the appropriate directory table. If this name already exists at an outer access mode, it is deleted.

resnam

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

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

reslen

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

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

quota

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

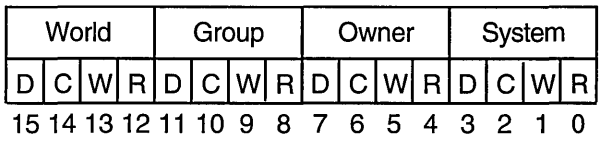
Maximum number of bytes of memory to be allocated for logical names contained in this logical name table. The **quota** argument is the address of a longword specifying this value.

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

promsk

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

Protection mask to be associated with the newly created shareable logical name table. The **promsk** argument is the address of a word that contains a value that represents four 4-bit fields. 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.



ZK-5893A-GE

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

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

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

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 descriptor–fixed length string descriptor

The name of the new logical name table. The **tabnam** argument is the address of a character string descriptor pointing to this name string. Table names are contained in either the process or system directory table (LNM$PROCESS_DIRECTORY or LNM$SYSTEM_DIRECTORY). Therefore, table names must consist of alphanumeric characters, dollar signs ($), and underscores (_); the maximum length is 31 characters.

If you do not specify this argument, a default name in the format LNM$*xxxx* is used, where *xxxx* is a unique hexadecimal number. You also need SYSPRV privilege 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 descriptor–fixed length string descriptor

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

acmode

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

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

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

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

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

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

A logical name table with supervisor mode access 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 write access to the system directory are required.

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

Required Quota

The parent table must have sufficient quota for the creation of the new table.

Related Services

$CRELNM, $DELLNM, $TRNLNM

Condition Values Returned

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

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

$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]

Arguments

prmflg

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

Indicator specifying whether the created mailbox is to be permanent or temporary. The **prmflg** argument is a 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. If you do not specify **maxmsg** or specify it as 0, the operating system provides a default value.

bufquo

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

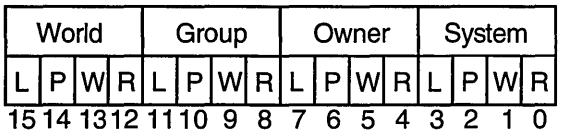
Number of bytes of system dynamic memory that can be used to buffer messages sent to the mailbox. The **bufquo** argument is a word value containing this number. If you do not specify the **bufquo** argument or specify it as 0, the operating system provides a default value.

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.



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.

flags

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

The **flags** argument is used for specifying options for the assign operation that occurs in $CREMBX. The **flags** argument is a longword bit mask that enables the user to specify that the channel assigned to the mailbox is a READ ONLY or WRITE ONLY channel. If the **flags** argument is not specified, then the default channel behavior is READ/WRITE. The $CMBDEF macro defines a symbolic name for each flag bit. The following table describes each flag.

Flag	Description
CMB$M_READONLY	When this flag is specified, $CREMBX assigns a read-only channel to the mailbox device. An attempt to issue a QIO WRITE operation on the mailbox channel results in an illegal I/O operation error.
CMB$M_WRITEONLY	When this flag is specified, $CREMBX assigns a write-only channel to the mailbox device. An attempt to issue a QIO READ operation on the mailbox channel results in an illegal I/O operation error.

For more information about the **flags** argument, see the *OpenVMS I/O User's Reference Manual.*

nullarg

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

Placeholding argument reserved to Digital.

Description

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

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

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

The initial security profile created for a mailbox is taken from the mailbox template for the device class. The owner is then set to the process UIC and the **promsk** argument replaces the protection mask.

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

A channel assigned to the mailbox READ ONLY is considered a READER. A channel assigned to the mailbox WRITE ONLY is considered a WRITER. A channel assigned to the mailbox READ/WRITE is considered both a WRITER and READER.

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

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

The mailbox unit number is determined when the mailbox is created. A process can obtain the unit number of the created mailbox by calling the Get Device/Volume Information ($GETDVI) service using the channel returned by $CREMBX.

Mailboxes are assigned sequentially increasing numbers (from 1 to a maximum of 9999) as they are created. When all unit numbers have been used, the system starts numbering again at unit 1. Logical names or mailbox names should be used to identify a mailbox between cooperating processes.

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

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

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

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

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

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

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

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

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

Required Access or Privileges

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

- TMPMBX privilege whenever the **prmflg** argument is specified as 0. However, a process that has PRMMBX privilege will also meet this requirement.
- PRMMBX privilege whenever the **prmflg** argument is specified as 1.
- SYSNAM privilege to place a logical name for a mailbox in the system logical name table.

- GRPNAM privilege to place a logical name for a mailbox in the group logical name table.

Required Quota

The calling process must have sufficient buffer I/O byte count (BYTLM) quota to allocate the mailbox 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 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.

Related Services

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

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The logical name string or string descriptor cannot be read by the caller, or the channel number cannot be written by the caller.
SS$_BADPARAM	The **bufquo** argument specified a value greater than approximately 65324, which is 65535 minus the size of a mailbox unit control block (UCB).
SS$_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 **prmflg** argument is undefined; this argument can have a value of 1 or 0.
SS$_NOIOCHAN	No I/O channel is available for assignment.
SS$_NOPRIV	The process does not have the privilege to create a temporary mailbox, a permanent mailbox, a mailbox in memory that is shared by multiple processors, or a logical name.
SS$_NOSHMBLOCK	No shared memory mailbox control block is available for use to create a new mailbox.

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

$CREPRC
Create Process

Creates on behalf of the calling process a subprocess or detached process on the current node, or a detached process on another VMScluster node.

Format

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

Arguments

pidadr

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

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

image

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

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

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

To create a process that will run under the control of a command language interpreter (CLI), specify SYS$SYSTEM:LOGINOUT.EXE as the image name.

input

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

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

output

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

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

error

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

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

Note that the **error** argument is ignored if the **image** argument specifies SYS$SYSTEM:LOGINOUT.EXE; in this case, SYS$ERROR has the same equivalence name as SYS$OUPUT.

prvadr

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

Privileges to be given to the created process. The **prvadr** argument is the address of a quadword bit vector wherein each bit corresponds to a privilege; setting a bit gives the privilege. If the **prvadr** argument is not specified, the current privileges are used.

Each bit has a symbolic name; the $PRVDEF macro defines these names. You form the bit vector by specifying the symbolic name of each desired privilege in a logical OR operation. Table SYS1–2 gives the symbolic name and description of each privilege.

Table SYS1–2 User Privileges

Privilege	Symbolic Name	Description
ACNT	PRV$V_ACNT	Create processes for which no accounting is done
ALLSPOOL	PRV$V_ALLSPOOL	Allocate a spooled device
ALTPRI	PRV$V_ALTPRI	Set (alter) any process priority
AUDIT	PRV$V_AUDIT	Generate audit records
BUGCHK	PRV$V_BUGCHK	Make bugcheck error log entries
BYPASS	PRV$V_BYPASS	Bypass UIC-based protection
CMEXEC	PRV$V_CMEXEC	Change mode to executive
CMKRNL	PRV$V_CMKRNL	Change mode to kernel
DETACH	PRV$V_DETACH	Create detached processes
DIAGNOSE	PRV$V_DIAGNOSE	Can diagnose devices
DOWNGRADE	PRV$V_DOWNGRADE	Can downgrade classification

(continued on next page)

Table SYS1–2 (Cont.) User Privileges

Privilege	Symbolic Name	Description
EXQUOTA	PRV$V_EXQUOTA	Can exceed quotas
GROUP	PRV$V_GROUP	Group process control
GRPNAM	PRV$V_GRPNAM	Place name in group logical name table
GRPPRV	PRV$V_GRPPRV	Group access via system protection field
IMPORT	PRV$V_IMPORT	Mount a nonlabeled tape volume
LOG_IO	PRV$V_LOG_IO	Perform logical I/O operations
MOUNT	PRV$V_MOUNT	Issue mount volume QIO
NETMBX	PRV$V_NETMBX	Create a network device
OPER	PRV$V_OPER	All operator privileges
PFNMAP	PRV$V_PFNMAP	Map to section by physical page frame number
PHY_IO	PRV$V_PHY_IO	Perform physical I/O operations
PRMCEB	PRV$V_PRMCEB	Create permanent common event flag clusters
PRMGBL	PRV$V_PRMGBL	Create permanent global sections
PRMMBX	PRV$V_PRMMBX	Create permanent mailboxes
PSWAPM	PRV$V_PSWAPM	Change process swap mode
READALL	PRV$V_READALL	Possess read access to everything
SECURITY	PRV$V_SECURITY	Can perform security functions
SETPRV	PRV$V_SETPRV	Set any process privileges
SHARE	PRV$V_SHARE	Can assign a channel to a non-shared device
SYSGBL	PRV$V_SYSGBL	Create system global sections
SYSLCK	PRV$V_SYSLCK	Queue systemwide locks
SYSNAM	PRV$V_SYSNAM	Place name in system logical name table
SYSPRV	PRV$V_SYSPRV	Access files and other resources as if you have a system UIC
TMPMBX	PRV$V_TMPMBX	Create temporary mailboxes
UPGRADE	PRV$V_UPGRADE	Can upgrade classification
VOLPRO	PRV$V_VOLPRO	Override volume protection
WORLD	PRV$V_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.

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

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

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

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

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

Minimum values are named PQL_M*xxxxx*, where *xxxxx* are the characters of the quota name that follow "PQL$_" in the quota name.

Default values are named PQL_D*xxxxx*, where *xxxxx* are the characters of the quota name that follow "PQL$_" in the quota name.

Individual Quotas

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

Minimum: PQL_MASTLM
Default: PQL_DASTLM
Nondeductible

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

Minimum: PQL_MBIOLM
Default: PQL_DBIOLM
Nondeductible

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

Minimum: PQL_MBYTLM
Default: PQL_DBYTLM
Pooled

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

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

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

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

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

Minimum: PQL_MCPULM
Default: PQL_DCPULM
Deductible

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

Minimum: PQL_MDIOLM
Default: PQL_DDIOLM
Nondeductible

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

Minimum: PQL_MENQLM
Default: PQL_DENQLM
Pooled

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

Minimum: PQL_MFILLM
Default: PQL_DFILLM
Pooled

PQL$_JTQUOTA
Job table quota. This quota limits the number of bytes of system paged pool used for the job logical name table. If the process being created is a subprocess, this item is ignored. A value of 0 represents an unlimited number of bytes.

Minimum: PQL_MJTQUOTA
Default: PQL_DJTQUOTA
Nondeductible

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

Minimum: PQL_MWSQUOTA
Default: PQL_DWSQUOTA
Nondeductible

Use of the Quota List The values specified in the quota list are not necessarily the quotas that are actually assigned to the created process. The $CREPRC service performs the following steps to determine the quota values that are assigned when you create a process on the same node:

1. It constructs a default quota list for the process being created, assigning it the default values for all quotas. Default values are SYSGEN 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 SYSGEN 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 DETACH or CMKRNL privilege, the resulting value is compared with the current value of the corresponding quota of the creating process and the lesser value is used.

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

     Pooled quota values are ignored.

   - If a detached process is being created and the creating process has DETACH 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 DETACH 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 VMScluster 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 DETACH or CMKRNL privilege, the service compares each value in the list with the current value of the corresponding quota of the creating process and uses the lesser value. It sends the resulting quota list to the node on which the new process is to be created.
2. On that node, the $CREPRC service constructs a default quota list for the process being created, assigning it default values for all quotas based on that node's SYSGEN 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 SYSGEN 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 DETACH or CMKRNL privilege to create a detached process with a UIC that is different from the UIC of the creating process.

If the **image** argument specifies the SYS$SYSTEM:LOGINOUT.EXE, the UIC of the created process will be the UIC of the caller of $CREPRC, and the UIC parameter is ignored.

mbxunt

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

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

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

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

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

If you specify this argument when you create a process on another node, an accounting message will be written to the mailbox when the process terminates. If the node is removed from the cluster before the created process terminates, an accounting message will be simulated. The simulated message will contain the created process's PID and name and a final status of SS$_NODELEAVE, but will lack execution statistics.

Note that two processes on different nodes cannot use the termination mailbox for general interprocess communication.

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

To receive the accounting message, the caller must issue a read to the mailbox. When the I/O completes, the second longword of the I/O status block, if one is specified, contains the process identification of the deleted process if the process was created on the same node. If it was created on a different VMScluster node, the second longword of the I/O status block contains 0.

The $ACCDEF macro defines symbolic names for offsets of fields within the accounting message. The offsets, their symbolic names, and the contents of each field are shown in the following table. Unless stated otherwise, the length of the field is 4 bytes.

Offset	Symbolic Name	Contents
0	ACC$W_MSGTYP	MSG$_DELPROC (2 bytes)
2		Not used (2 bytes)
4	ACC$L_FINALSTS	Exit status code
8	ACC$L_PID	External process identification
12		Not used (4 bytes)
16	ACC$Q_TERMTIME	Current time in system format at process termination (8 bytes)
24	ACC$T_ACCOUNT	Account name for process, blank filled (8 bytes)
32	ACC$T_USERNAME	User name, blank filled (12 bytes)
44	ACC$L_CPUTIM	CPU time used by the process, in 10-millisecond units
48	ACC$L_PAGEFLTS	Number of page faults incurred by the process
52	ACC$L_PGFLPEAK	Peak paging file usage
56	ACC$L_WSPEAK	Peak working set size

Offset	Symbolic Name	Contents
60	ACC$L_BIOCNT	Count of buffered I/O operations performed by the process
64	ACC$L_DIOCNT	Count of direct I/O operations performed by the process
68	ACC$L_VOLUMES	Count of volumes mounted by the process
72	ACC$Q_LOGIN	Time, in system format, that process logged in (8 bytes)
80	ACC$L_OWNER	Process identification of owner

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

stsflg

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

Options selected for the created process. The **stsflg** argument is a longword bit vector wherein a bit corresponds to an option. Only bits 0 to 18 are used; the others are reserved and must be 0.

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

Symbolic Name	Description
PRC$M_BATCH	Create a batch process. DETACH privilege is required.
PRC$M_DETACH	Create a detached process.
PRC$M_DISAWS	Disable system initiated working set adjustment.
PRC$M_HIBER	Force process to hibernate before it executes the image.
PRC$M_IMGDMP	Enable image dump facility. If an image terminates due to an unhandled condition, the image dump facility writes the contents of the address space to a file in your current default directory. The file name is the same as the name of the terminated image. The file type is .DMP.

Symbolic Name	Description
PRC$M_INTER	Create an interactive process. This option is meaningful only if the **image** argument specifies SYS$SYSTEM:LOGINOUT.EXE. The purpose of this option is to provide you with information about the process. When you specify this option, it identifies the process as one that is in communication with another user (an interactive process). For example, if you use the DCL lexical function F$MODE to make an inquiry about a process that has specified the PRC$M_INTER option, F$MODE returns the value INTERACTIVE.
PRC$M_NETWRK	Create a process that is a network connect object. DETACH privilege required.
PRC$M_NOACNT	Do not perform accounting. ACNT privilege is required.
PRC$M_NOPASSWORD	Do not display the *Username:* and *Password:* prompts if the process is interactive and detached and the image is SYS$SYSTEM:LOGINOUT.EXE. If you specify this option in your call to $CREPRC, the process created by the call is logged in under the user name associated with the creating process. If you do not specify this option for an interactive process, SYS$SYSTEM:LOGINOUT.EXE prompts you for the user name and password to be associated with the process. The prompts are displayed at the SYS$INPUT device.
PRC$M_NOUAF	Do not check authorization file if the process is detached and the image is SYS$SYSTEM:LOGINOUT.EXE. You should not specify this option if a subprocess is being created. In previous versions of the operating system, the symbolic name of this option was PRC$M_LOGIN. The symbolic name has been changed to more accurately denote the effect of setting this bit. For compatibility with existing user programs, you can still specify this bit as PRC$M_LOGIN.
PRC$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. DETACH privilege is required.

Note that options PRCM_BATCH, PRCM_INTER, PRCM_NOUAF, PRCM_NETWRK, and PRC$M_NOPASSWORD are intended for use by Digital software.

itmlst
OpenVMS usage: reserved
type: longword (unsigned)

The **itmlst** argument is reserved to Digital.

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

Name of the VMScluster 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.

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 VMScluster node. A detached process can be created on the current VMScluster 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_DETACH flag specifies that the created process is detached.

Creating a process is synchronous in that the process has actually been created and its PID determined before control returns to the program that requested the system service. Note, however, that the new process has not necessarily begun to execute at that point. Some error conditions are not detected until the created process executes. These conditions include an invalid or nonexistent image; invalid SYS$INPUT, SYS$OUTPUT, or SYS$ERROR logical name equivalence; inadequate quotas; or insufficient privilege to execute the requested image.

In creating a detached or subprocess, you can specify that the process run the image SYS$SYSTEM:LOGINOUT.EXE. During interactive logins, LOGINOUT performs the following functions:

1. It validates user name and password.
2. It reads the system authorization file record associated with that user and redefines the process environment based on information from the record.
3. It maps a command language interpreter (CLI) into the process and passes control to it.

The CLI reads a command from SYS$INPUT, processes it, and reads another command. The presence of the CLI enables the process to execute multiple images. It also enables an image running in the process to use run-time library procedures, such as LIB$SPAWN, LIB$DO_COMMAND, and LIB$SET_LOGICAL, that require a CLI.

Running in the context of a process you create through $CREPRC, LOGINOUT can perform some or all of the preceding steps, depending on whether the process is a subprocess or a detached process and on the values of PRC$M_NOPASSWORD and PRC$M_NOUAF in the **stsflg** argument.

Certain characteristics of a created process can be specified explicitly through $CREPRC system service arguments, while other characteristics are propagated implicitly from the $CREPRC caller. Implicit characteristics include the following:

- Current default directory
- Creator's equivalence name for SYS$DISK
- User and account names
- Command language interpreter (CLI) name and command table file name

Note, however, that after the process has been created, if it runs LOGINOUT and LOGINOUT redefines the process environment, those characteristics will be overridden by information from the system authorization file.

Several process characteristics are relevant to the creation of a process an another VMScluster 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 VMScluster 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 SYSGEN parameters on the other VMScluster 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 that 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:

- DETACH or CMKRNL privilege to create any of the following types of process:
  - A detached process with a UIC that is different from the UIC of the calling process

  - A detached process with a larger value specified for some quota than is authorized for the caller
  - A detached process on another node if the SYSGEN parameter CWCREPRC_ENABLE has a value of 0
- DETACH 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 VMScluster 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 VMScluster node that a process can create with the same user name is controlled by the MAXDETACH entry in the user authorization file (UAF).

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

Related Services

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

Condition Values Returned

SS$_ACCVIO	The caller cannot read a specified input string or string descriptor, the privilege list, or the quota list; or the caller cannot write the process identification.
SS$_DUPLNAM	The specified process name duplicates one already specified within that group.

SS$_EXPRCLM	The creation of a detached process failed because the creating process already reached its limit for the creation of detached processes. This limit is established by the MAXDETACH quota in the user authorization file (UAF) of the creating process.
SS$_EXQUOTA	At least one of the following conditions is true: • The process has exceeded its quota for the creation of subprocesses. • A quota value specified for the creation of a subprocess exceeds the creating process's corresponding quota. • The quota is deductible and the remaining quota for the creating process would be less than the minimum.
SS$_INCOMPAT	The remote node is running an incompatible version of the operating system, namely, one that does not support remote process creation.
SS$_INSFMEM	The system dynamic memory is insufficient for the requested operation.
SS$_INVARG	An invalid argument was specified.
SS$_IVLOGNAM	At least one of the following two conditions is true: • The specified process name, has a length of 0 or has more than 15 characters. • The specified image name, input name, output name, or error name has more than 255 characters.
SS$_IVQUOTAL	The quota list is not in the proper format.
SS$_IVSTSFLG	A reserved status flag was specified.
SS$_NODELEAVE	The specified node was removed from the VMScluster 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]

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.

Alpha

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 Digital-supplied procedures (including Run-Time Library procedures). This system service provides no means to communicate a change in virtual address space with other routines. Digital recommends that you use either $EXPREG or the Run-Time Library procedure Allocate Virtual Memory (LIB$GET_VM) to get memory. You can find documentation on LIB$GET_VM in the *OpenVMS*

RTL Library (LIB$) Manual. When using $DELTVA, you should take care to delete only pages that you have specifically created.

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The **inadr** argument cannot be read by the caller, or the **retadr** argument cannot be written by the caller.
SS$_EXQUOTA	The process has exceeded its paging file quota.
SS$_INSFWSL	The process's working set limit is not large enough to accommodate the increased size of the virtual address space.
SS$_NOPRIV	A page in the specified range is in the system address space.
SS$_PAGOWNVIO	A page in the specified range already exists and cannot be deleted because it is owned by a more privileged access mode than that of the caller.
SS$_VASFULL	The process's virtual address space is full; no space is available in the page tables for the requested pages.

$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

Arguments

region_id_64

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

The region ID associated with the region to create the virtual address range. The file VADEF.H in SYS$STARLET_C.TLB and the $VADEF macro in STARLET.MLB define a symbolic name for each of the three default regions in P0, P1, and P2 space.

The following region IDs are defined:

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

Other region IDs, as returned by the $CREATE_REGION_64 service, can be specified. Also, given a particular virtual address, the region ID for the region it is in can be obtained by calling the $GET_REGION_INFO system service specifying the VA$_REGSUM_BY_VA function.

start_va_64

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

The starting address for the created virtual address range. The specified virtual address must be a CPU-specific page-aligned address.

length_64

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

Length of the virtual address space to be created. The length specified must be a multiple of CPU-specific pages.

acmode

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

Access mode associated with the call to $CRETVA_64. The access mode determines the owner mode of the pages as well as the read and write protection on the pages. The **acmode** argument is a longword containing the access mode.

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

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

The $CRETVA_64 service uses whichever of the following access modes is least privileged:

- The access mode specified by the **acmode** argument.
- The access mode of the caller.

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

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

flags

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

Flag mask controlling the characteristics of the demand-zero pages created. The **flags** argument is a longword bit vector in which each bit corresponds to a flag. The $VADEF macro and the VADEF.H file define a symbolic name for each flag. You construct the **flags** argument by performing a logical OR operation on the symbol names for all desired flags.

The following table describes the flag that is valid for the $CRETVA_64 service:

Flag	Description
VA$M_NO_OVERMAP	Pages cannot overmap existing address space. By default, pages can overmap existing address space.

All other bits in the flags argument are reserved for future use by Digital and should be specified as 0. The condition value SS$_IVVAFLG is returned if any undefined bits are set.

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

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

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

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

Description

The Create Virtual Address Space service is a kernel mode service that can be called from any mode. The service adds a range of demand-zero allocation pages, starting at the virtual address specified by the **start_va_64** argument. The pages are added to a process's virtual address space for the execution of the current image. Expansion occurs at the next free available address within the specified region if the range of addresses is beyond the next free available address.

The new pages, which were previously inaccessible to the process, are created as demand-zero pages.

The returned address is always the lowest virtual address in the range of pages created. The returned length is always an unsigned byte count indicating the length of the range of pages created.

Successful return status from $CRETVA means that the specified address space was created of the size specified in the **length_64** argument.

If $CRETVA_64 creates pages that already exist, the service deletes those pages if they are not owned by a more privileged access mode than that of the caller. Any such deleted pages are reinitialized as demand-zero pages.

If the condition value SS$_ACCVIO is returned by this service, a value *cannot* be returned in the memory locations pointed to by the **return_va_64** and **return_length_64** arguments.

If an address within the specified address range is not within the bounds of the specified region, the condition value SS$_PAGNOTINREG is returned.

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

Required Privileges

None.

Required Quota

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

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

Related Services

$CREATE_BUFOBJ_64, $CREATE_REGION_64, $DELETE_REGION_64, $DELTVA_64, $EXPREG_64, $LCKPAG_64, $LKWSET_64, $PURGE_WS, $SETPRT_64, $ULKPAG_64, $ULWSET_64

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The **return_va_64** or **return_length_64** argument cannot be written by the caller.
SS$_EXPGFLQUOTA	The process has exceeded its paging file quota.
SS$_INSFWSL	The process's working set limit is not large enough to accommodate the increased virtual address space.
SS$_IVACMODE	The caller's mode is less privileged than the create mode associated with the region.
SS$_IVREGID	Invalid region ID specified.
SS$_IVVAFLG	An invalid flag, a reserved flag, or an invalid combination of flags and arguments was specified.
SS$_LEN_NOTPAGMULT	The **length_64** argument is not a multiple of CPU-specific pages.
SS$_PAGNOTINREG	A page in the specified range is not within the specified region.
SS$_PAGOWNVIO	A page in the specified range already exists and cannot be deleted because it is owned by a more privileged access mode than that of the caller.
SS$_REGISFULL	The specified virtual region is full.
SS$_VA_IN_USE	A page in the specified range is already mapped and the VA$M_NO_OVERLAP flag was set.
SS$_VA_NOTPAGALGN	The **start_va_64** argument is not CPU-specific page-aligned.

$CRMPSC
Create and Map Section

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

Format

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

Arguments

inadr

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

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

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

Alpha

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

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

Caution

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

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

In general, the **inadr** argument should be specified. However, it may be omitted to request a special feature: for permanent global sections, you may omit the **inadr** argument, or specify it as 0, to request that the section be created but not mapped. Such a request will be granted regardless of the setting of the SEC$M_EXPREG flag. However, to ensure compatibility between VAX and Alpha systems, Digital recommends that the SEC$M_EXPREG flag be clear when the **inadr** argument is omitted.♦

VAX

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

Note

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

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

Caution

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

Alpha

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

acmode

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

Access mode 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 **inadr** argument. See the **inadr** 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$M_PERM	Pages are permanent. By default, pages are temporary.
SEC$M_PFNMAP	Pages form a page frame section. By default, pages form a disk-file section. Pages mapped by SEC$M_PFNMAP are not included in or charged against the process's working set; they are always valid. Do not lock these pages in the working set by using $LKWSET; this can result in a machine check if they are in I/O space. ‡On Alpha systems, when the SEC$M_PFNMAP flag is set, the **pagcnt** and **relpag** arguments are interpreted in CPU-specific pages, *not* as pagelets.
SEC$M_SYSGBL	Pages form a system global section. By default, pages form a group global section.
SEC$M_PAGFIL	Pages form a global page-file section. By default, pages form a disk-file section. SEC$M_PAGFIL also implies SEC$M_WRT and SEC$M_DZRO.
SEC$M_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.

‡Alpha specific

gsdnam

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

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

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

ident

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

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

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

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

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

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

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

relpag

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

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

Alpha

On Alpha systems, the **relpag** argument is interpreted as an index into the section file, measured in pagelets for a file-backed section or 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

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

Alpha

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 SYS1–3 shows which arguments are required and which are optional for three different uses of the $CRMPSC service.

Table SYS1–3 Required and Optional Arguments for the $CRMPSC Service

Argument	Create/Map Global Section	Map Global[1] Section	Create/Map Private Section
inadr	Optional[2]	Required	Required
retadr	Optional	Optional	Optional
acmode	Optional	Optional	Optional
flags			
SEC$M_GBL	Required	Ignored	Not used
SEC$M_CRF[3]	Optional	Not used	Optional
SEC$M_DZRO[3]	Optional	Not used	Optional
SEC$M_EXPREG	Optional	Optional	Optional
SEC$M_PERM	Optional[2]	Not used	Not used
SEC$M_PFNMAP	Optional	Not used	Optional
SEC$M_SYSGBL	Optional	Optional	Not used
SEC$M_WRT	Optional	Optional	Optional
SEC$M_PAGFIL	Optional	Not used	Not used
gsdnam	Required	Required	Not used
ident	Optional	Optional	Not used
relpag[3]	Optional	Optional	Not used
chan[3]	Required		Required
pagcnt	Required		Required
vbn[3]	Optional		Optional
prot	Optional		Not used
pfc[3]	Optional		Optional

[1]The Map Global Section ($MGBLSC) service maps an existing global section.

[2]See the description of **inadr** for the rules governing the omission of the argument.

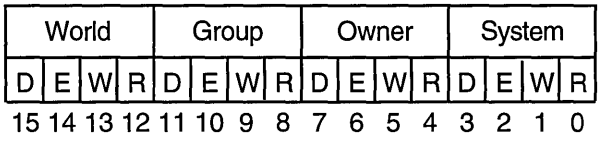
[3]For physical page frame sections: **vbn** specifies the starting page frame number; **chan** must be 0; **pfc** is not used; and the SEC$M_CRF and SEC$M_DZRO flag bit settings are invalid. For page-file sections, **chan** must be 0 and **pfc** not used.

prot

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

Protection to be applied to the global page-file and PFN sections. For file-backed sections, the protection is taken from the backing file and the **prot** argument is ignored.

The mask contains four 4-bit fields. Bits are read from right to left in each field. The following diagram depicts the mask.



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.

Alpha

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.♦

VAX

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. See the memory management section in the *OpenVMS Programming Concepts Manual*.

Depending on the actual operation requested, certain arguments are required or optional. Table SYS1–3 summarizes how the $CRMPSC service interprets the arguments passed to it and under what circumstances it requires or ignores arguments.

The $CRMPSC service returns the virtual addresses of the virtual address space created in the **retadr** argument, if specified. The section is mapped from a low address to a high address, whether the section is mapped in the program or control region.

If an error occurs during the mapping of a global section, the **retadr** argument, if specified, indicates the pages that were successfully mapped when the error occurred. If no pages were mapped, the value of the longwords is indeterminate. In this case, either both longwords of the **retadr** argument will contain the value –1, or the value of the longwords will be unaltered.

The SEC$M_PFNMAP flag setting identifies the memory for the section as starting at the page frame number specified in the **vbn** argument and extending for the number of CPU-specific pages specified in the **pagcnt** argument. Setting the SEC$M_PFNMAP flag places restrictions on the following arguments.

Argument	Restriction
chan	Must be 0
pagcnt	Must be specified; cannot be 0
vbn	Specifies first page frame to be mapped
pfc	Does not apply
SEC$M_CRF	Must be 0
SEC$M_DZRO	Must be 0
SEC$M_PERM	Must be 1 if the flags SEC$M_GBL or SEC$M_SYSGBL are set

Setting the SEC$M_PAGFIL flag places the following restrictions on the following flags.

Flag	Restriction
SEC$M_CRF	Must be 0
SEC$M_DZRO	Assumed to be 0
SEC$M_GBL	Must be 1
SEC$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.

VAX

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.

Required Access or Privileges

If $CRMPSC specifies a global section and the SS$_NOPRIV condition value is returned, the process does not have the required privilege to create that section. In order to create global sections, the process must have the following privileges:

- SYSGBL privilege to create a system global section
- PRMGBL privilege to create a permanent global section
- PFNMAP privilege to create a page frame section
- SHMEM privilege to create a global section in memory shared by multiple processors (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 SYSGEN parameter GBLPAGFIL.

Related Services

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

Condition Values Returned

SS$_NORMAL	The service completed successfully. The specified global section already exists and has been mapped.
SS$_CREATED	The service completed successfully. The specified global section did not previously exist and has been created.
SS$_ACCVIO	The **inadr** argument, **gsdnam** argument, or name descriptor cannot be read by the caller; the **inadr** argument was omitted; or the **retadr** argument cannot be written by the caller.

SS$_ENDOFFILE	The starting virtual block number specified is beyond the logical end-of-file, or the value in the **relpag** argument is greater than or equal to the actual size of the global section.
SS$_EXBYTLM	The process has exceeded the byte count quota; the system was unable to map the requested file.
SS$_EXGBLPAGFIL	The process has exceeded the systemwide limit on global page-file pages; no part of the section was mapped.
SS$_EXQUOTA	The process exceeded its paging file quota while creating copy-on-reference or page-file-backing-store pages.
SS$_GPTFULL	There is no more room in the system global page table to set up page table entries for the section.
SS$_GSDFULL	There is no more room in the system space allocated to maintain control information for global sections.
SS$_ILLPAGCNT	The page count value is negative or is 0 for a physical page frame section.
SS$_INSFMEM	Not enough pages are available in the specified shared memory to create the section.
SS$_INSFWSL	The process's working set limit is not large enough to accommodate the increased size of the address space.
SS$_IVCHAN	An invalid channel number was specified, that is, a channel number of 0 or a number larger than the number of channels available.
SS$_IVCHNLSEC	The channel number specified is currently active.
SS$_IVLOGNAM	The specified global section name has a length of 0 or has more than 43 characters.
SS$_IVLVEC	The specified section was not installed using the /PROTECT qualifier.
SS$_IVSECFLG	An invalid flag, a reserved flag, a flag requiring a privilege you lack, or an invalid combination of flags was specified.
SS$_IVSECIDCTL	The match control field of the global section identification is invalid.

SS$_NOPRIV	The process does not have the privileges to create a system global section (SYSGBL) or a permanent group global section (PRMGBL). The process does not have the privilege to create a section starting at a specific physical page frame number (PFNMAP). The process does not have the privilege to create a global section in memory shared by multiple processors (SHMEM). A page in the input address range is in the system address space. The specified channel is not assigned or was assigned from a more privileged access mode.
SS$_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$_SECTBLFUL	There are no entries available in the system global section table or in the process section table.
SS$_TOOMANYLNAM	The logical name translation of the **gsdnam** argument exceeded the allowed depth.
SS$_VA_IN_USE	A page in the specified input address range is already mapped and the flag SEC$M_NO_OVERMAP is set.
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]]

Arguments

region_id_64

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

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

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

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

file_offset_64

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

Byte offset into the file that marks the beginning of the section. The **file_offset_64** argument is a quadword containing this number. If you specify the **file_offset_64** argument as 0, the section is created beginning with the first byte in the file.

The **file_offset_64** argument must be a multiple of virtual disk blocks.

length_64

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

Length, in bytes, of the private disk file section to be created and mapped to. The length specified must be 0 or a multiple of virtual disk blocks. If the length specified is 0 or extends beyond end-of-file (EOF), the disk file is mapped up to and including the virtual block number that contains EOF.

chan

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

Number of the channel on which the file has been accessed. The **chan** argument is a longword containing this number. The access mode at which the channel was opened must be equal to or less privileged than the access mode of the caller.

Use the OpenVMS Record Management Services (RMS) macro $OPEN to access a file; the file options parameter in the file access block must indicate a user file open (UFO keyword).

acmode

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

Access mode that is to be the owner of the pages created during the mapping. The **acmode** argument is a longword containing the access mode.

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

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

The most privileged access mode used is the access mode of the caller. The calling process can delete pages only if those pages are owned by an access mode equal to or less privileged than the access mode of the calling process.

flags

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

Flag mask specifying the characteristics of the private section to be created. The **flags** argument is a longword bit vector in which each bit corresponds to a flag. The $SECDEF macro and the SECDEF.H file define a symbolic name for each flag. You construct the flags argument by performing a logical OR operation on the symbol names for all desired flags.

The following table describes each flag that is valid for the $CRMPSC_FILE_64 service:

Flag	Description
SEC$M_CRF	Pages are copy-on-reference.
SEC$M_DZRO	Pages are demand-zero pages. By default, they are not zeroed when copied. Note that SEC$M_DZRO and SEC$M_CRF cannot both be set and that SEC$M_DZRO set and SEC$M_WRT clear is an invalid combination.
SEC$M_EXPREG	Pages are mapped into the first available space at the current end of the specified region.
SEC$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 Digital and should be specified as 0. The condition value SS$_IVSECFLG is returned if any undefined bits are set or if an illegal combination of flags is set.

return_va_64

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

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

return_length_64

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

The 32-bit or 64-bit virtual address of a naturally aligned quadword into which the service returns the length of the usable virtual address range mapped in bytes. This length might differ from the total amount mapped. If the section being mapped does not completely fill the last page used to map the section, the **return_va_64** and **return_length_64** arguments indicate the highest address that actually maps the section.

fault_cluster

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

Page fault cluster in byte units indicating how many pages are to be brought into memory when a page fault occurs for a single page. The fault cluster specified will be rounded up to a multiple of CPU-specific pages.

If this argument is specified as 0, the process default page fault cluster will be used. If this argument is specified as more than the maximum allowed for the system, no condition value will be returned. The systemwide maximum will be used.

start_va_64

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

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

Description

The Create and Map Private Disk File Section service allows a process to create a map to a private disk file section. Creating a private disk file section involves mapping all or part of a disk file as a section. The section is mapped from a low address to a high address whether the section is mapped in a region that grows from low to high addresses or from high to low addresses.

The flag SEC$M_WRT applies only to the way in which the newly created section is mapped. For a file to be made writable, the channel used to open the file must allow write access to the file.

If the condition value SS$_ACCVIO is returned by this service, a value *cannot* be returned in the memory locations pointed to by the **return_va_64** and **return_length_64** arguments.

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

Required Privileges

None

Required Quota

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

The process must have sufficient byte count quota to satisfy the request.

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

Related Services

$CREATE_REGION_64, $CRMPSC, $CRMPSC_GFILE_64, $CRMPSC_GPFILE_64, $CRMPSC_GPFN_64, $CRMPSC_PFN_64, $DELETE_REGION_64, $DELTVA_64, $LCKPAG_64, $LKWSET_64, $PURGE_WS, $SETPRT_64, $ULKPAG_64, $ULWSET_64, $UPDSEC_64, $UPDSEC_64W

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The **return_va_64** argument or the **return_length_64** argument cannot be written by the caller.
SS$_CHANVIO	The specified channel was assigned from a more privileged access mode.
SS$_ENDOFFILE	The **file_offset_64** argument specified is beyond the logical end-of-file.
SS$_EXBYTLM	The process has exceeded the byte count quota; the system was unable to map the requested file.
SS$_EXPGFLQUOTA	The process exceeded its paging file quota.
SS$_INSFWSL	The process's working set limit is not large enough to accommodate the increased virtual address space.
SS$_IVCHAN	An invalid channel number was specified; the channel number specified was 0 or a channel that is unassigned.
SS$_IVCHNLSEC	The channel number specified is currently active, or there are no files opened on the specified channel.
SS$_IVIDENT	An invalid channel number was specified; the channel number specified is larger than the number of channels available.
SS$_IVLOGNAM	The specified global section name has a length of 0 or has more than 43 characters.
SS$_IVREGID	Invalid region ID specified.
SS$_IVSECFLG	An invalid flag, a reserved flag, or an invalid combination of flags and arguments was specified.
SS$_LEN_NOTBLKMULT	The **length_64** argument is not a multiple of virtual disk blocks.
SS$_NOTFILEDEV	The device is not a file-oriented, random-access, or directory device.
SS$_OFF_NOTBLKALGN	The **file_offset_64** argument is not a multiple of virtual disk blocks.
SS$_NOWRT	The file is read-only, the flag bit SEC$M_WRT was set, and the flag bit SEC$M_CRF is not set.
SS$_PAGNOTINREG	A page in the specified range is not within the specified region.
SS$_PAGOWNVIO	A page in the specified range already exists and cannot be deleted because it is owned by a more privileged access mode than that of the caller.
SS$_REGISFULL	The specified virtual region is full; no space is available in the region for the pages created to contain the mapped section.

SS$_VA_IN_USE	A page in the specified input address range is already mapped, and the flag SEC$M_NO_OVERMAP is set.
SS$_VA_NOTPAGALGN	The **start_va_64** argument is not CPU-specific 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]]]

Arguments

gs_name_64

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

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

ident_64

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

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

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

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

When a section is mapped at creation time, the match control field is ignored. If you specify the **ident_64** argument as 0, the version number and match control fields default to 0.

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

file_offset_64

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

Byte offset into the file that marks the beginning of the section. The **file_offset_64** argument is a quadword containing this number. If you specify the **file_offset_64** argument as 0, the section is created beginning with the first byte in the file.

The file offset specified must be a multiple of virtual disk blocks.

length_64

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

Length, in bytes, of the global disk file section to be created. The length specified must be 0 or a multiple of virtual disk blocks. If the length specified is 0 or extends beyond the end-of-file (EOF), the global disk file section is created up to and including the virtual block number that contains EOF.

chan

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

Number of the channel on which the file has been accessed. The **chan** argument is a longword containing this number. The access mode at which the channel was opened must be equal to or less privileged than the access mode of the caller.

You can use the OpenVMS Record Management Services (RMS) macro $OPEN to access a file; the file options parameter in the file access block must indicate a user file open (UFO keyword).

region_id_64

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

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

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

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

section_offset_64

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

Offset into the global section to start mapping into the process's virtual address space. The offset specified must be a multiple of virtual disk blocks.

acmode

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

Access mode that is to be the owner of the pages created during the mapping. The **acmode** argument is a longword containing the access mode.

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

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

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

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

flags

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

Flag mask specifying the characteristics of the global section to be created. The **flags** argument is a longword bit vector in which each bit corresponds to a flag. The $SECDEF macro and the SECDEF.H file define a symbolic name for each flag. You construct the **flags** argument by performing a logical OR operation on the symbol names for all desired flags.

The following table describes each flag that is valid for the $CRMPSC_GFILE_64 service:

Flag	Description
SEC$M_CRF	Pages are copy-on-reference.
SEC$M_GBL	Pages form a global section. By default, this flag is always present in this service and cannot be disabled.
SEC$M_WRT	Pages form a read/write section. By default, pages form a read-only section.
SEC$M_DZRO	Pages are demand-zero pages. By default, they are not zeroed when copied. Note that SEC$M_DZRO and SEC$M_CRF cannot both be set and that SEC$M_DZRO set and SEC$M_WRT clear is an invalid combination.
SEC$M_EXPREG	Pages are mapped into the first available space at the current end of the specified region.
SEC$M_NO_ OVERMAP	Pages cannot overmap existing address space. By default, pages can overmap existing address space.
SEC$M_PERM	Pages are permanent. By default, pages are temporary.
SEC$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 Digital and should be specified as 0. The condition value SS$_IVSECFLG is returned if any undefined bits are set or if an illegal combination of flags is set.

return_va_64

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

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

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

return_length_64

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

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

Upon successful completion of this service, the value in the **return_length_64** argument indicates the amount of created address space backed by the section file.

If the number of disk blocks mapped does not represent an exact multiple of CPU-specific pages, the last page in the mapped address space will not be completely mapped by the section file. In this case, modifying memory beyond the amount indicated by **return_length_64** can result in the loss of this data.

Unlike the **return_length_64** argument for the $CREATE_GFILE service, upon successful completion of this service, the **return_length_64** argument does not represent the total length of the global section created if the **section_offset_64** argument was specified as non-zero. The value in the **section_offset_64** argument plus the value in the **return_length_64** argument is the total length of the global disk file section created.

fault_cluster

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

Page fault cluster in byte units indicating how many pages are to be brought into memory when a page fault occurs for a single page. The fault cluster specified will be rounded up to a multiple of CPU-specific pages.

If this argument is specified as 0, the system default page fault cluster will be used. If this argument is specified as more than the maximum allowed for the system, no error will be returned. The systemwide maximum will be used.

start_va_64

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

The starting virtual address to map the global disk file section. The specified virtual address must be a CPU-specific page aligned address. If the flag SEC$M_EXPREG is specified, this argument will not be used. If SEC$M_EXPREG is clear and the **start_va_64** argument is not specified or is specified as 0, the condition value SS$_IVSECFLG will be returned.

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

map_length_64

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

Length of the global disk file section to be mapped. The length specified must be a multiple of virtual disk blocks. If this argument is not specified as zero, the global disk section is mapped up to and including the last disk block in the section.

Description

The Create and Map Global Disk File Section service allows a process to create and map to a global disk file section. Creating a global disk file section involves defining all or part of a disk file as a section. The section is mapped from a low address to a high address whether the section is mapped in a region that grows from low to high addresses or from high to low addresses. If the $CRMPSC_GFILE_64 service specifies a global disk file section that already exists, the service maps it.

If the condition value SS$_ACCVIO is returned by this service, a value *cannot* be returned in the memory locations pointed to by the **return_va_64** and **return_length_64** arguments.

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

The flag SEC$M_WRT applies only to the way in which the newly created section is mapped. For a file to be made writable, the channel used to open the file must allow write access to the file.

Required Privileges

In order to create a global section, the process must have the following privileges:

- SYSGBL privilege to create a system global section (if flag SEC$M_SYSGBL is set)
- PRMGBL privilege to create a permanent global section

Required Quota

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

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

Related Services

$CREATE_REGION_64, $CRMPSC, $CRMPSC_FILE_64, $CRMPSC_GPFILE_64, $CRMPSC_GPFN_64, $CRMPSC_PFN_64, $DELETE_REGION_64, $DELTVA_64, $DGBLSC, $LCKPAG_64, $LKWSET_64, $MGBLSC_64, $PURGE_WS, $SETPRT_64, $ULKPAG_64, $ULWSET_64, $UPDSEC_64, $UPDSEC_64W

Condition Values Returned

SS$_NORMAL	The service completed successfully. The specified global section already exists and has been mapped.
SS$_CREATED	The service completed successfully. The specified global section did not previously exist and has been created.

SS$_ACCVIO	The **gs_name_64** argument cannot be read by the caller, or the **return_va_64** or **return_length_64** argument cannot be written by the caller.
SS$_CHANVIO	The specified channel was assigned from a more privileged access mode.
SS$_ENDOFFILE	The **file_offset_64** argument specified is beyond the logical end-of-file.
SS$_EXBYTLM	The process has exceeded the byte count quota; the system was unable to map the requested file.
SS$_EXPGFLQUOTA	The process exceeded its paging file quota, creating copy-on-reference pages.
SS$_GBLSEC_MISMATCH	Global section type mismatch. The specified global section was found; however, it was not a global disk file section.
SS$_GPTFULL	There is no more room in the system global page table to set up page table entries for the section.
SS$_GSDFULL	There is no more room in the system space allocated to maintain control information for global sections.
SS$_INSFWSL	The process's working set limit is not large enough to accommodate the increased virtual address space.
SS$_IVACMODE	The caller's mode is less privileged than the create mode associated with the region.
SS$_IVCHAN	An invalid channel number was specified; the channel number specified was 0 or a channel that is unassigned.
SS$_IVCHNLSEC	The channel number specified is currently active or there are no files opened on the specified channel.
SS$_IVIDENT	An invalid channel number was specified; the channel number specified is larger than the number of channels available.
SS$_IVLOGNAM	The specified global section name has a length of 0 or has more than 43 characters.
SS$_IVREGID	Invalid region ID specified.
SS$_IVSECFLG	An invalid flag, a reserved flag, or an invalid combination of flags and arguments was specified.
SS$_IVSECIDCTL	The match control field of the global section identification is invalid.
SS$_LEN_NOTBLKMULT	The **length_64** or the **map_length_64** argument is not a multiple of virtual disk blocks.
SS$_NOPRMGBL	The process does not have the privileges to create or delete a permanent group global section (PRMGBL).

SS$_NOSYSGBL	The process does not have the privileges to create or delete a system global section (SYSGBL).
SS$_NOTFILEDEV	The device is not a file-oriented, random-access, or directory device.
SS$_NOWRT	The file is read-only, and the flag bit SEC$M_CRF is not set.
SS$_OFF_NOTBLKALGN	The **file_offset_64** or **section_offset_64** argument is not virtual disk block aligned.
SS$_OFFSET_TOO_BIG	The **section_offset_64** argument specified is beyond the logical end-of-file.
SS$_PAGNOTINREG	A page in the specified range is not within the specified region.
SS$_PAGOWNVIO	A page in the specified input address range is owned by a more privileged access mode.
SS$_REGISFULL	The specified virtual region is full; no space is available in the region for the pages created to contain the mapped section.
SS$_SECTBLFUL	There are no entries available in the system global section table.
SS$_TOOMANYLNAM	The logical name translation of the **gs_name_64** argument exceeded the allowed depth of 10.
SS$_VA_IN_USE	A page in the specified input address range is already mapped, and the flag SEC$M_NO_OVERMAP is set.
SS$_VA_NOTPAGALGN	The **start_va_64** argument is not CPU-specific page-aligned.

$CRMPSC_GPFILE_64 (Alpha 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]]

Arguments

gs_name_64

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

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

ident_64

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

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

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

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

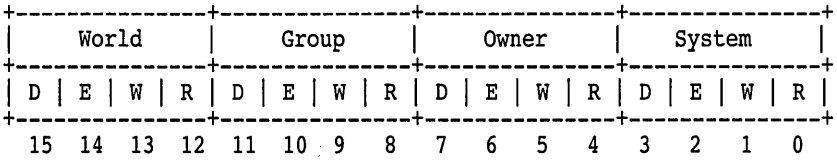
When a section is mapped at creation time, the match control field is ignored. If you specify the **ident_64** argument as 0, the version number and match control fields default to 0.

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

prot

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

Protection to be applied to the global page file section. The mask contains four 4-bit fields. Bits are read from right to left in each field. The following diagram depicts the mask:



Cleared bits indicate that read, write, execute, and delete access, in that order, are granted to the particular category of user. Only read, write, and execute access are meaningful for section protection. Delete access bits are ignored. Read access also grants execute access for those situations where execute access applies. If zero is specified, read access and write access are granted to all users.

length_64

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

Length, in bytes, of the global page file section to be created. The length specified must be a multiple of CPU-specific pages. A length of 0 cannot be specified.

region_id_64

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

The region ID associated with the region to map the global page file section.

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

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

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

section_offset_64
OpenVMS usage: byte offset
type: quadword (unsigned)
access: read only
mechanism: by value

Offset into the global section to start mapping into the process's virtual address space. The offset specified must be a multiple of virtual disk blocks.

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

Access mode that is to be the owner of the pages created during the mapping. The **acmode** argument is a longword containing the access mode.

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

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

The most privileged access mode used is the access mode of the caller. The calling process can delete pages only if those pages are owned by an access mode equal to or less privileged than the access mode of the calling process.

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

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

Flag mask specifying the characteristics of the global section to be created. The **flags** argument is a longword bit vector in which each bit corresponds to a flag. The $SECDEF macro and the SECDEF.H file define a symbolic name for each flag. You construct the flags argument by performing a logical OR operation on the symbol names for all desired flags.

The following table describes each flag that is valid for the $CRMPSC_GPFILE_64 service:

Flag	Description
SEC$M_DZRO	Pages are demand-zero pages. By default, this flag is always present in this service and cannot be disabled.
SEC$M_EXPREG	Pages are mapped into the first available space at the current end of the specified region. SEC$M_EXPREG cannot be specified with the SEC$M_NO_OVERMAP flag.
SEC$M_GBL	Pages form a global section. By default, this flag is always present in this service and cannot be disabled.
SEC$M_NO_ OVERMAP	Pages cannot overmap existing address space. By default, pages can overmap existing address space. SEC$M_NO_OVERMAP cannot be specified with the SEC$M_EXPREG flag.
SEC$M_PAGFIL	Pages form a global page-file section. By default, this flag is always present in this service and cannot be disabled.
SEC$M_PERM	Pages are permanent. By default, pages are temporary.
SEC$M_SYSGBL	Pages form a system global section. By default, pages form a group global section.
SEC$M_WRT	Pages form a read/write section. By default, this flag is always present in this service and cannot be disabled.

All other bits in the **flags** argument are reserved for future use by Digital and should be specified as 0. The condition value SS$_IVSECFLG is returned if any undefined bits are set or if an invalid combination of flags is set.

return_va_64

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

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

return_length_64

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

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

start_va_64

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

The starting virtual address to map the global page file section. The specified virtual address must be a CPU-specific page-aligned address. If the flag

SEC$M_EXPREG is specified, the **start_va_64** argument must not be specified or must be specified as 0. If SEC$M_EXPREG is set and the **start_va_64** argument is non-zero, the condition value SS$_IVSECFLG is returned.

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

map_length_64

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

Length of the global page file section to be mapped. The length specified must be a multiple of CPU-specific pages. If this argument is not specified or is specified as zero, the global file section is mapped up to and including the last page in that section.

Description

The Create and Map Global Page File Section service allows a process to create a map to global page file section. Creating a global page file section involves defining a global section backed up by the system page file. The section is mapped from a low address to a high address whether the section is mapped in a region that grows from low to high addresses or from high to low addresses. If the $CRMPSC_GPFILE_64 service specifies a global section that already exists, the service maps it.

If the condition value SS$_ACCVIO is returned by this service, a value *cannot* be returned in the memory locations pointed to by the **return_va_64** and **return_length_64** arguments.

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

Required Privileges

In order to create a global section, the process must have the following privileges:

- SYSGBL privilege to create a system global section (if flag SEC$M_SYSGBL is set)
- PRMGBL privilege to create a permanent global section

Required Quota

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

The working set limit quota (WSQUOTA) of the process must be sufficient to accommodate the increased size of the process page table required by the increase in virtual address space when the section is mapped.

Related Services

$CREATE_GPFILE, $CREATE_REGION_64, $CRMPSC, $CRMPSC_FILE_64, $CRMPSC_GFILE_64, $CRMPSC_GPFN_64, $CRMPSC_PFN_64, $DELETE_REGION_64, $DELTVA_64, $DGBLSC, $LCKPAG_64, $LKWSET_64, $MGBLSC_64, $PURGE_WS, $SETPRT_64, $ULKPAG_64, $ULWSET_64, $UPDSEC_64, $UPDSEC_64W

Condition Values Returned

SS$_NORMAL	The service completed successfully. The specified global section already exists and has been mapped.
SS$_CREATED	The service completed successfully. The specified global section did not previously exist and has been created.
SS$_ACCVIO	The **gs_name_64** argument cannot be read by the caller, or the **return_va_64** or **return_length_64** argument cannot be written by the caller.
SS$_EXBYTLM	The process has exceeded the byte count quota.
SS$_EXGBLPAGFIL	The process has exceeded the system-wide limit on global page file pages; no part of the section was mapped.
SS$_EXPGFLQUOTA	The process exceeded its paging file quota, creating copy-on-reference pages.
SS$_GBLSEC_MISMATCH	Global section type mismatch. The specified global section was found; however, it is not a global disk or page file section.
SS$_GPTFULL	There is no more room in the system global page table to set up page table entries for the section.
SS$_GSDFULL	There is no more room in the system space allocated to maintain control information for global sections.
SS$_INSFWSL	The process's working set limit is not large enough to accommodate the increased virtual address space.
SS$_IVACMODE	The caller's mode is less privileged than the create mode associated with the region.
SS$_IVLOGNAM	The specified global section name has a length of 0 or has more than 43 characters.
SS$_IVREGID	Invalid region ID specified.
SS$_IVSECFLG	An invalid flag, a reserved flag, or an invalid combination of flags and arguments was specified.
SS$_IVSECIDCTL	The match control field of the global section identification is invalid.
SS$_LEN_NOTPAGMULT	The **length_64** argument is not a multiple of CPU-specified pages or was specified as 0.

SS$_NOPRMGBL	The process does not have the privileges to create or delete a permanent group global section (PRMGBL).
SS$_NOSYSGBL	The process does not have the privileges to create or delete a system global section (SYSGBL).
SS$_NOWRTACC	The specified global section is not copy-on-reference and does not allow write access.
SS$_OFF_NOTPAGALGN	The **section_offset_64** argument is not CPU-specific page aligned if a map to a global page file section was requested (SEC$M_PAGFIL is set in the flags argument).
SS$_OFFSET_TOO_BIG	The **section_offset_64** argument specified is beyond the logical end-of-file.
SS$_PAGNOTINREG	A page in the specified range is not within the specified region.
SS$_PAGOWNVIO	A page in the specified input address range is owned by a more privileged access mode.
SS$_REGISFULL	The specified virtual region is full; no space is available in the region for the pages created to contain the mapped section.
SS$_SECTBLFUL	There are no entries available in the system global section table.
SS$_TOOMANYLNAM	The logical name translation of the **gs_name_64** argument exceeded the allowed depth of 10.
SS$_VA_IN_USE	A page in the specified input address range is already mapped, and the flag SEC$M_NO_OVERMAP is set.
SS$_VA_NOTPAGALGN	The **start_va_64** argument is not CPU-specific page-aligned.

$CRMPSC_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]]

Arguments

gs_name_64

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

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

ident_64

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

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

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

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

When a section is mapped at creation time, the match control field is ignored. If you specify the **ident_64** argument as 0, the version number and match control fields default to 0.

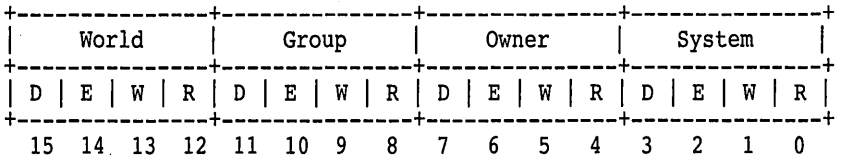
The version number is in the second longword. The version number contains two fields: a minor identification in the low-order 24 bits and a major identification in the high-order 8 bits. You can assign values for these fields by installation convention to differentiate versions of global sections. If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

prot

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

Protection to be applied to the global page file section.

The mask contains four 4-bit fields. Bits are read from right to left in each field. The following diagram depicts the mask:



Cleared bits indicate that read, write, execute, and delete access, in that order, are granted to the particular category of user. Only read, write, and execute access are meaningful for section protection. Delete access bits are ignored. Read access also grants execute access for those situations where execute access applies. If zero is specified, read access and write access are granted to all users.

start_pfn

OpenVMS usage:	page frame number
type:	longword (unsigned)
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.

region_id_64

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

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

The following region IDs are defined:

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

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

relative_page

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

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

acmode

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

Access mode that is to be the owner of the pages created during the mapping. The **acmode** argument is a longword containing the access mode.

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

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

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

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

flags

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

Flag mask specifying the characteristics of the global section to be created. The **flags** argument is a longword bit vector in which each bit corresponds to a flag. The $SECDEF macro and the SECDEF.H file define a symbolic name for each

flag. You construct the **flags** argument by performing a logical OR operation on the symbol names for all desired flags.

The following table describes each flag that is valid for the $CRMPSC_GPFN_64 service:

Flag	Description
SEC$M_EXPREG	Pages are mapped into the first available space at the current end of the specified region.
SEC$M_GBL	Pages form a global section. By default, this flag is always present in this service and cannot be disabled.
SEC$M_PERM	Pages are permanent. By default, this flag is always present in this service and cannot be disabled.
SEC$M_PFNMAP	Pages form a page frame section. By default, this flag is always present in this service and cannot be disabled.
SEC$M_NO_OVERMAP	Pages cannot overmap existing address space. By default, pages can overmap existing address space.
SEC$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 Digital and should be specified as 0. The condition value SS$_IVSECFLG is returned if any undefined bits are set or if an illegal combination of flags is set.

return_va_64

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

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

return_length_64

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

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

start_va_64

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

The starting virtual address to map the global page frame section. The specified virtual address must be a CPU-specific page-aligned address. If the flag SEC$M_EXPREG is specified, the **start_va_64** argument must not be specified or

must be specified as 0. If SEC$M_EXPREG is set and the **start_va_64** argument is non-zero, the condition value SS$_IVSECFLG is returned.

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

map_page_count

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

Length of the global page frame section to be mapped in CPU-specific pages.

Description

The Create and Map Global Page Frame Section service allows a process to create and map to a global page frame section. Creating a global page frame section involves defining certain physical page frame numbers (PFNs) as a section.

All global page frame sections are permanent. Pages mapped to a global page frame section are not included in or charged against the process's working set; they are always valid. Do not lock these pages in the working set by using $LKWSET; this can result in a machine check if they are in I/O space.

If the condition value SS$_ACCVIO is returned by this service, a value *cannot* be returned in the memory locations pointed to by the **return_va_64** and **return_length_64** arguments.

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

Required Privileges

In order to create a global page frame section, the process must have the following privileges:

- SYSGBL privilege to create a system global section (if flag SEC$M_SYSGBL is set)
- PRMGBL privilege to create a permanent global section (if flag SEC$M_PERM is set)
- PFNMAP privilege to create a page frame section

Required Quota

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

Related Services

$CREATE_GPFN, $CREATE_REGION_64, $CRMPSC, $CRMPSC_FILE_64, $CRMPSC_GFILE_64, $CRMPSC_GPFILE_64, $CRMPSC_PFN_64, $DELETE_REGION_64, $DELTVA_64, $DGBLSC, $MGBLSC_GPFN_64

Condition Values Returned

SS$_NORMAL	The service completed successfully. The specified global section already exists and has been mapped.
SS$_CREATED	The service completed successfully. The specified global section did not previously exist and has been created.
SS$_ACCVIO	The **gs_name_64** argument cannot be read by the caller, or the **return_va_64** or **return_length_64** argument cannot be written by the caller.
SS$_EXBYTLM	The process has exceeded the byte count quota.
SS$_GBLSEC_MISMATCH	Global section type mismatch. The specified global section was found; however, it was not a global disk file section.
SS$_GPTFULL	There is no more room in the system global page table to set up page table entries for the section.
SS$_GSDFULL	There is no more room in the system space allocated to maintain control information for global sections.
SS$_ILLRELPAG	The specified relative page argument is either larger than the highest page number within the section or is not a valid 32-bit physical page frame number.
SS$_IVACMODE	The caller's mode is less privileged than the create mode associated with the region.
SS$_IVLOGNAM	The specified global section name has a length of 0 or has more than 43 characters.
SS$_IVREGID	Invalid region ID specified.
SS$_IVSECFLG	An invalid flag, a reserved flag, or an invalid combination of flags and arguments was specified.
SS$_IVSECIDCTL	The match control field of the global section identification is invalid.
SS$_NOPFNMAP	The process does not have the privilege to create or delete a section starting at a specific physical page frame number (PFNMAP).
SS$_NOPRMGBL	The process does not have the privileges to create or delete a permanent group global section (PRMGBL).
SS$_NOWRTACC	The specified global section is not copy-on-reference and does not allow write access.
SS$_NOSYSGBL	The process does not have the privileges to create or delete a system global section (SYSGBL).
SS$_PAGNOTINREG	A page in the specified range is not within the specified region.

SS$_REGISFULL	The specified virtual region is full; no space is available in the region for the pages created to contain the mapped section.
SS$_TOOMANYLNAM	The logical name translation of the **gs_name_64** argument exceeded the allowed depth of 10.
SS$_VA_IN_USE	A page in the specified input address range is already mapped, and the flag SEC$M_NO_OVERMAP is set.
SS$_VA_NOTPAGALGN	The **start_va_64** argument is not CPU-specific page-aligned.

$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]

Arguments

region_id_64

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

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

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

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

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.

acmode

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

Access mode that is to be the owner of the pages created during the mapping. The **acmode** argument is a longword containing the access mode.

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

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

The most privileged access mode used is the access mode of the caller. The calling process can delete pages only if those pages are owned by an access mode equal to or less privileged than the access mode of the calling process.

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

flags

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

Flag mask specifying the characteristics of the private section to be created. The **flags** argument is a longword bit vector in which each bit corresponds to a flag. The $SECDEF macro and the SECDEF.H file define a symbolic name for each flag. You construct the **flags** argument by performing a logical OR operation on the symbol names for all desired flags.

The following table describes each flag that is valid for the $CRMPSC_PFN_64 service:

Flag	Description
SEC$M_EXPREG	Pages are mapped into the first available space at the current end of the specified region.
SEC$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 Digital 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-bit 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-bit or 64-bit virtual address of a naturally aligned quadword into which the service returns the virtual address.

return_length_64

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

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

start_va_64

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

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

Description

The Create and Map Private Page Frame Section service allows a process to create a map to a private page frame section. Creating a private page frame section involves defining certain physical page numbers (PFNs) as a section. The section is mapped from a low address to a high address whether the section is mapped in a region that grows from low to high addresses or from high to low addresses.

All global page frame sections are permanent. Pages mapped by SEC$M_PFNMAP are not included in or charged against the process's working set; they are always valid. Do not lock these pages in the working set by using $LKWSET_64; this can result in a machine check if they are in I/O space.

If the condition value SS$_ACCVIO be returned by this service, a value *cannot* be returned in the memory locations pointed to by the **return_va_64** and **return_length_64** arguments.

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

Required Privileges

PFNMAP privilege is required to create a page frame section.

Required Quota

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

Related Services

$CREATE_REGION_64, $CRMPSC, $CRMPSC_FILE_64, $CRMPSC_GFILE_64, $CRMPSC_GPFILE_64, $CRMPSC_GPFN_64, $DELETE_REGION_64, $DELTVA_64

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The **return_va_64** argument or the **return_length_64** argument cannot be written by the caller.
SS$_INSFWSL	The process's working set limit is not large enough to accommodate the increased virtual address space.
SS$_IVACMODE	The caller's mode is less privileged than the create mode associated with the region.
SS$_IVREGID	Invalid region ID specified.
SS$_IVSECFLG	An invalid flag, a reserved flag, or an invalid combination of flags and arguments was specified.
SS$_NOPFNMAP	The process does not have the privilege to create a section starting at a specific physical page frame number (PFNMAP).
SS$_PAGNOTINREG	A page in the specified range is not within the specified region.
SS$_PAGOWNVIO	A page in the specified input address range is owned by a more privileged access mode.
SS$_REGISFULL	The specified virtual region is full; no space is available in the region for the pages created to contain the mapped section.
SS$_VA_IN_USE	A page in the specified input address range is already mapped, and the flag SEC$M_NO_OVERMAP is set.
SS$_VA_NOTPAGALGN	The **start_va_64** argument is not CPU-specific page-aligned.

$DACEFC
Disassociate Common Event Flag Cluster

Releases the calling process's association with a common event flag cluster.

Format

SYS$DACEFC 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]

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

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

Required Quota

None

Related Services

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

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_IVCHAN	You specified an invalid channel number, that is, a channel number of 0 or a number larger than the number of channels available.
SS$_NOPRIV	The specified channel is not assigned or was assigned from a more privileged access mode.

$DCLAST
Declare AST

Queues an 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]

Arguments

astadr

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

AST service routine to be executed. On Alpha systems, the **astadr** argument is the 32-bit 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) by 32-bit value (VAX)

AST parameter to be passed to the AST routine specified by the **astadr** argument. On Alpha sytems, 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

Alpha

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.♦

VAX

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

Format

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

Arguments

addres

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

Routine to receive control when a change mode trap or a compatibility mode fault occurs. The **addres** argument is the exception handling code in the address space of the calling process.

If you specify the **addres** argument as 0, $DCLCMH clears the previously declared handler.

prvhnd

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

Address of a previously declared handler. The **prvhnd** argument is the address of a longword containing the address of the previously declared handler.

type

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

Handler type indicator. The **type** argument is a longword value. The value 0 (the default) indicates that a change mode handler is to be declared for the access mode at which the request is issued; the value 1 specifies that a compatibility mode handler is to be declared.

Description

Alpha

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.♦

VAX

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, $SETSFM, $UNWIND

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The longword to receive the address of the previous change mode handler cannot be written by the caller.
‡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

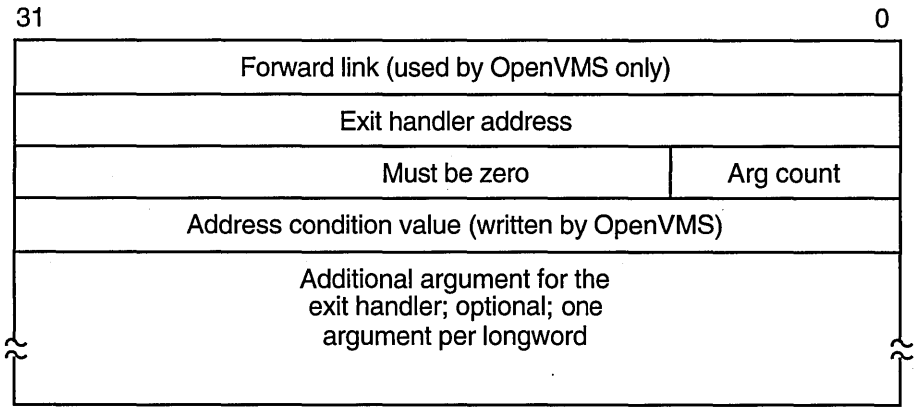
SYS$DCLEXH desblk

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.



ZK-5184A-GE

Description

The Declare Exit Handler service declares an exit handling routine that receives control when an image exits. Image exit normally occurs when the image currently executing in a process returns control to the operating system. Image exit might also occur when you call the Exit ($EXIT) or Force Exit ($FORCEX) service.

Exit handlers are described by exit control blocks. The operating system maintains a separate list of these control blocks for user, supervisor, and executive modes. The $DCLEXH service adds the description of an exit handler to the front of one of these lists. The actual list to which the exit control block is added is determined by the access mode of the caller.

At image exit, the exit handlers declared from user mode are called first; they are called in the reverse order from which they were declared.

Each exit handler is executed only once; it must be redeclared before it can be executed again. The exit handling routine is called as a normal procedure with the argument list specified in the third through *n*th longwords of the exit control block. The first argument is the address of a longword to receive a system status code indicating the reason for exit; the system always fills in this longword before calling the exit handler.

You can call this service only from user, supervisor, and executive modes.

Required Access or Privileges

None

Required Quota

None

Related Services

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

The Cancel Exit Handler ($CANEXH) service removes an exit control block from the list.

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The first longword of the exit control block cannot be written by the caller.
SS$_IVSSRQ	The call to the service is invalid because it was made from kernel mode.
SS$_NOHANDLER	The exit handler control block address was not specified or was specified as 0.

$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

Arguments

buffer_handle_64

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

The buffer object to be deleted. The **buffer_handle_64** argument is the 32-bit or 64-bit address of a 2-longword array previously returned by a $CREATE_BUFOBJ_64 call.

Description

The Delete Buffer Object system service deletes the buffer object identified by the **buffer_handle_64** argument. The associated memory is made free to be paged, swapped, or deleted.

Buffer objects are also automatically deleted at image rundown.

Required Privileges

None

Required Quota

None

Related Services

$CREATE_BUFOBJ_64

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The **buffer_handle_64** argument cannot be read by the caller.
SS$_BADPARAM	The **buffer_handle_64** argument is not a valid buffer handle.
SS$_NOPRIV	The buffer object was created by a more privileged access mode than the caller's access mode.

$DELETE_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]

Arguments

user_criteria

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

Description of intruder or suspect. The **user_criteria** argument is the address of a character-string descriptor pointing to a buffer containing the user criteria to match an intrusion record's user specification in the intrusion database.

The **user_criteria** argument is a character string of 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 for OpenVMS address, local failed user name, or local terminal. The user specification for an intrusion record is based on the input to the $SCAN_INTRUSION service and the settings of the LGI system parameter. For more information, see the *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.

flags

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

Functional specification for the service. The **flags** argument is a longword bit mask wherein each bit corresponds to an option.

Each flag option has a symbolic name. The $CIADEF macro defines the following valid name for the $DELETE_INTRUSION service.

Symbolic Name	Description
CIA$M_IGNORE_RETURN	The service should not wait for the return status from the security server. No return status from the server's function will be returned to the caller.

Description

The Delete Intrusion service deletes from the intrusion database a set of records matching the criteria you specify in the **user_criteria** argument. All records matching the criteria you specify are deleted. You do not have to call the service more than once to delete a set of records.

For example, if you specify an asterisk (*) for the **user_criteria** argument, the service deletes all records in the intrusion database with one call.

Required Access or Privileges

$DELETE_INTRUSION requires access to the intrusion database. You must have SECURITY privilege to access the database.

Required Quota

None

Related Services

$SCAN_INTRUSION, $SHOW_INTRUSION

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The **user_criteria** argument cannot be read.
SS$_BADBUFLEN	The length of the **user_criteria** argument is out of range.
SS$_BADPARAM	An invalid flag was specified in the **flags** argument.
SS$_NOSECURITY	The caller does not have SECURITY privilege.

This service can also return any of the following messages passed from the security server:

SECSRV$_CIADBEMPTY	No records in the intrusion database.
SECSRV$_ NOSUCHINTRUDER	No records matching the specified criteria were found in the intrusion database.
SECSRV$_ SERVERNOTACTIVE	The security server is not currently active. Try the request again later.

$DELETE_PROXY
Delete or Modify Proxy

Deletes an existing proxy or removes the default user or a local user from an existing proxy in the proxy database.

Format

SYS$DELETE_PROXY rem_node ,rem_user ,[local_user] ,[flags]

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 flat is set with the **flags** argument.

Asterisk (*) and percent sign (%) wildcards are allowed for the remote node specification. If you specify wildcards for the **rem_node** argument, the security server searches for an exact match to the specified remote node first. If it does not find an exact match, the server performs the requested operations on all of the matching proxies in the proxy database.

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

Remote user name of the proxy to be deleted from or modified in the proxy database. The **rem_user** argument is the address of a character-string descriptor pointing to the user name string.

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

The **rem_user** argument can be specified in user identification code (UIC) format ([*group, member*]). Brackets are allowed only if the remote user name string specifies a UIC. Group and member are character-string representations of octal numbers with no leading zeros.

Asterisk (*) and percent sign (%) wildcards are allowed for the remote user specification. If you specify wildcards for the **rem_user** argument, the server searches for an exact match to the specified remote user first. If it does not find an exact match, the server performs the requested operations on all of the matching proxies in the proxy database.

local_user

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

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

A local user name consists of 0 to 32 alphanumeric characters, including dollar signs ($) and underscores (_). If the **local_user** argument is not specified or has a length of 0, the server will delete the entire record or records specified by the **rem_node** and **rem_user** arguments from the proxy database.

If the **local_user** argument is specified, the server will delete only the user name specified by the **local_user** argument from the record specified by the **rem_node** and **rem_user** arguments. The **local_user** argument can specify either the proxy's default user or a user name in the proxy's local users list.

flags

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

Functional specification for the service and type of user the **local_user** argument represents. The **flags** argument is a longword bit mask wherein each bit corresponds to an option.

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

Symbolic Name	Description
PRX$M_BYPASS_EXPAND	The service should not convert the node name specified in the **rem_node** argument to its corresponding DECnet for OpenVMS full name. If this flag is set, it is the caller's responsibility to ensure that the fully expanded node name is passed into the service.
PRX$M_IGNORE_RETURN	The service should not wait for a return status from the security server. No return status from the server's function will be returned to the caller.
PRX$M_EXACT	The service should match exactly the remote node and remote user and ignore wildcards.

Description

The Delete Proxy service deletes a proxy from, or modifies an existing proxy in, the proxy database.

Required Access or Privileges

$DELETE_PROXY requires access to the proxy database. To achieve access, the caller must have either SYSPRV privilege or a UIC group less than or equal to the MAXSYSGRP system parameter.

Required Quota

None

Related Services

$ADD_PROXY, $DISPLAY_PROXY, $VERIFY_PROXY

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The **rem_node**, **rem_user**, **local_user**, or **flags** argument cannot be read by the service.
SS$_BADBUFLEN	The length of the **rem_node**, **rem_user**, or **local_user** argument was out of range.
SS$_BADPARAM	An invalid flag was specified in the **flags** argument.
SS$_NOSYSPRV	The caller does not have access to the proxy database.

This service can also return any of the following messages passed from the security server:

SECSRV$_ BADNODENAMELEN	The node name length is out of range.
SECSRV$_ BADREMUSERLEN	The remote user name length is out of range.
SECSRV$_INVALIDDELETE	You attempted to remove the last local user with no default user remaining, or you tried to remove the last default user with no local user remaining. You must have at least one local user or one default user.
SECSRV$_NOSUCHPROXY	The proxy specified by the **rem_node** and **rem_user** arguments does not exist in the proxy database.
SECSRV$_NOSUCHUSER	The specified local user does not exist in the proxy's local user list, or is not the proxy's default user.
SECSRV$_ PROXYNOTACTIVE	Proxy processing is currently stopped. Try the request again later.
SECSRV$_ SERVERNOTACTIVE	The security server is not currently active. Try the request again later.

$DELETE_REGION_64 (Alpha 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

Arguments

region_id_64

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

The region ID associated with the region to be deleted. The region ID specified must be one returned by the $CREATE_REGION_64 service. You cannot specify VAC_P0, VAC_P1, or VA$C_P2.

acmode

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

Access mode associated with the call to $DELETE_REGION_64. The **acmode** argument is a longword containing the access mode.

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

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

The most privileged access mode used is the access mode of the caller. The caller can delete pages only if those pages are owned by an access mode equal to or less privileged than the access mode of the caller.

Once all pages are deleted within the region, the region can be deleted only if the region is owned by an access mode equal to or less privileged than the access mode of the caller.

return_va_64

OpenVMS usage:	address
type:	quadword address
access:	write only

mechanism: by 32-bit 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-bit 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-bit 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-bit 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 Virtual Region service is a kernel mode service that can be called from any mode. The Delete Region 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 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 will be 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 since 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 will be 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 deleted.

If an error other than SS$_ACCVIO occurs, the returned address and returned length indicate the pages that were successfully deleted before the error occurred. If no pages were deleted, the **return_va_64** argument 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, $DELTVA_64, $EXPREG_64, $GET_REGION_INFO, $MGBLSC_64, $MGBLSC_GPFN_64

Condition Values Returned

SS$_NORMAL	Successful completion. All pages within the region have been deleted as well as the region itself.
SS$_ACCVIO	The **return_va_64** argument or the **return_length_64** argument cannot be written by the caller.
SS$_IVREGID	Invalid region ID specified. This condition value is returned if P0, P1, or P2 space is specified since these regions cannot be deleted, or if no region exists for the specified ID.
SS$_REGOWNVIO	The region is owned by a more privileged access mode than the access mode of the caller. All pages within the region have been deleted; however, the region has not been deleted.
SS$_PAGOWNVIO	A page within the specified region is owned by a more privileged access mode than the access mode of the caller.

$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.

Format

SYS$DELLNM tabnam ,[lognam] ,[acmode]

Arguments

tabnam

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

Name of a logical name table or a list of tables to be searched for the logical name to be deleted. The **tabnam** argument is the address of a descriptor that points to the table name. This argument is required.

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

If **tabnam** translates to the name of a list of tables, $DELLNM does the following:

- If you specify the **lognam** argument, $DELLNM searches (in order) each table in the list until it finds the first table that contains the specified logical name. If the logical name is at the specified access mode, $DELLNM then deletes occurrences of the logical name at the specified access mode and at outer access modes within the table.
- If you do not specify the **lognam** argument, $DELLNM deletes all of the logical names at the specified access mode or at outer access modes from the first table in the list whose access mode is equal to or less privileged than the caller's access mode.

lognam

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

Logical name to be deleted. The **lognam** argument is the address of a descriptor that points to the logical name string.

acmode

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

Access mode to be used in the delete operation. The **acmode** argument is the address of a byte containing this access mode. The $PSLDEF macro defines symbolic names for the four access modes.

You determine the access mode actually used in the delete operation by *maximizing* the access mode of the caller with the access mode specified by the **acmode** argument; that is, the less privileged of the two is used.

However, if you have SYSNAM privilege, the delete operation is executed at the specified access mode regardless of the caller's access mode.

If you omit this argument or 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 access to the logical name table to delete a name
- Either delete access to the logical name table or write access to the directory table that contains the table name to delete a shareable logical name table
- SYSNAM privilege to delete a logical name or table at an inner access mode
- GRPNAM or SYSPRV privilege to delete a logical name from a group table
- SYSNAM or SYSPRV privilege to delete a logical name from a system table

Required Quota

None

Related Services

$CRELNM, $CRELNT, $TRNLNM

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The service cannot access the locations specified by one or more arguments.
SS$_BADPARAM	One or more arguments have an invalid value, or a logical name table name was not specified.
SS$_IVLOGNAM	The **lognam** argument specifies a string whose length is not in the required range of 1 through 255 characters.
SS$_IVLOGTAB	The **tabnam** argument does not specify a logical name table.

SS$_NOLOGNAM	The specified logical name table does not exist, or a logical name with an access mode equal to or less privileged than the caller's access mode does not exist in the logical name table.
SS$_NOLOGTAB	The specified logical name table does not exist.
SS$_NOPRIV	The caller lacks the necessary privilege to delete the logical name.
SS$_TOOMANYLNAM	The logical name translation of the table name exceeded the allowable depth (10 translations).

$DELMBX
Delete Mailbox

Marks a permanent mailbox for deletion.

Format

SYS$DELMBX chan

Argument

chan

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

Number of the channel assigned to the mailbox that is to be deleted. The **chan** argument is a word containing this number.

Description

The Delete Mailbox service marks a permanent mailbox for deletion. The actual deletion of the mailbox and of its associated logical name assignment occur when no more I/O channels are assigned to the mailbox.

You can delete a mailbox only from an access mode equal to or more privileged than the access mode from which the mailbox channel was assigned. Temporary mailboxes are automatically deleted when their reference count goes to 0.

The $DELMBX service does not deassign the channel assigned by the caller, if any. The caller must deassign the channel with the Deassign I/O Channel ($DASSGN) service.

Required Access or Privileges

You need PRMMBX privilege to delete a permanent mailbox.

Required Quota

None

Related Services

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

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_DEVNOTMBX	The specified channel is not assigned to a mailbox.

SS$_IVCHAN	You specified an invalid channel number, that is, a channel number of 0 or a number larger than the number of channels available.
SS$_NOPRIV	The specified channel is not assigned to a device; the process does not have the privilege to delete a permanent mailbox or a mailbox in memory shared by multiple processors; or the access mode of the caller is less privileged than the access mode from which the channel was assigned.

$DELPRC
Delete Process

Allows a process to delete itself or another process.

Format

SYS$DELPRC [pidadr] ,[prcnam]

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 VMScluster system.

You must specify the **pidadr** argument to delete processes in other UIC groups.

prcnam

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

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.

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.

Required Access or Privileges

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

- GROUP privilege to delete processes in the same group that do not have the same UIC
- WORLD privilege to delete any process in the system

Required Quota

None. Deductible resource quotas granted to subprocesses are returned to the creating process when the subprocesses are deleted.

Related Services

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

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The process name string or string descriptor cannot be read by the caller, or the process identification cannot be written by the caller.
SS$_INCOMPAT	The remote node is running an incompatible version of the operating system.
SS$_INSFMEM	The system dynamic memory is insufficient for completing the operation.
SS$_NONEXPR	The specified process does not exist, or an invalid process identification was specified.
SS$_NOPRIV	The caller does not have the privilege to delete the specified process.
SS$_NOSUCHNODE	The process name refers to a node that is not currently recognized as part of the cluster.
SS$_REMRSRC	The remote node has insufficient resources to respond to the request. (Bring this error to the attention of your system manager.)
SS$_UNREACHABLE	The remote node is a member of the cluster but is not accepting requests. (This is normal for a brief period early in the system boot process.)

$DELTVA
Delete Virtual Address Space

Deletes a range of addresses from a process's virtual address space. Upon successful completion of the service, the deleted pages are inaccessible, and references to them cause access violations.

Format

SYS$DELTVA inadr ,[retadr] ,[acmode]

Arguments

inadr

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

Starting and ending virtual addresses of the pages to be deleted. The **inadr** argument is the address of a 2-longword array containing, in order, the starting and the ending process virtual addresses. If the starting and ending virtual addresses are the same, a single page is deleted. The addresses are adjusted up or down to fall on CPU-specific page boundaries. Only the virtual page number portion of each virtual address is used; the low-order byte-within-page bits are ignored.

The $DELTVA service deletes pages starting at the address contained in the second longword of the **inadr** argument and ending at the address in the first longword. Thus, if you use the same address array for both the Create Virtual Address Space ($CRETVA) and the $DELTVA services, the pages are deleted in the reverse order from which they were created.

retadr

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

Starting and ending process virtual addresses of the pages that $DELTVA has deleted. The **retadr** argument is the address of a 2-longword array containing, in order, the starting and ending process virtual addresses.

acmode

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

Access mode on behalf of which the service is to be performed. The **acmode** argument is a longword containing the access mode.

The most privileged access mode used is the access mode of the caller. The calling process can delete pages only if those pages are owned by an access mode equal to or less privileged than the access mode of the calling process.

Description

The Delete Virtual Address Space service deletes a range of addresses from a process's virtual address space. Upon successful completion of the service, the deleted pages are inaccessible, and references to them cause access violations. If any of the pages in the specified range have already been deleted or do not exist, the service continues as if the pages were successfully deleted.

If an error occurs while pages are being deleted, the **retadr** argument specifies the pages that were successfully deleted before the error occurred. If no pages are deleted, both longwords in the return address array contain the value –1.

Required Access or Privileges

None

Required Quota

None

Related Services

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

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The input address array cannot be read by the caller, or the return address array cannot be written by the caller.
SS$_NOPRIV	A page in the specified range is in the system address space.
SS$_PAGOWNVIO	A page in the specified range is owned by an access mode more privileged than the access mode of the caller.

$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

Arguments

region_id_64

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

The region ID associated with the region from which to address the VA space.

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

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

Other region IDs, as returned by the $CREATE_REGION_64 service, can be specified. Also, the region ID that a virtual address is in can be obtained by calling the $GET_REGION_INFO service, specifying the VA$_REGSUM_BY_VA function.

start_va_64

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

The starting virtual address of the pages to be deleted. The specified virtual address must be a CPU-specific page-aligned address.

length_64

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

Length of the virtual address space to be deleted. 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 $DELTVA_64. The **acmode** argument is a longword containing the access mode.

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

Value	Symbolic Name	Access Mode
0	PSL$C_KERNEL	Kernel
1	PSL$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-bit or 64-bit reference

The lowest process virtual address of the deleted virtual address range. The **return_va_64** argument is the 32-bit or 64-bit virtual address of a naturally aligned quadword into which the $DELTVA_64 service returns the virtual address.

return_length_64

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

The 32-bit or 64-bit virtual address of a naturally aligned quadword into which the $DELTVA_64 service returns the length of the virtual address range deleted in bytes.

Description

The Delete Virtual Address Space service is a kernel mode service that can be called from any mode. This service deletes a range of addresses from a process's virtual address space. Upon successful completion of the service, the deleted pages are inaccessible, and references to them cause access violations. If any of the pages in the specified range have already been deleted or do not exist, the service continues as if the pages were successfully deleted.

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

Required Privileges

None

Required Quota

None.

Related Services

$CREATE_REGION_64, $CRETVA_64, $CRMPSC_FILE_64, $CRMPSC_GFILE_64, $CRMPSC_GPFILE_64, $CRMPSC_GPFN_64, $CRMPSC_PFN_64, $DELETE_REGION_64, $EXPREG_64, $MGBLSC_64, $MGBLSC_GPFN_64

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The **return_va_64** argument or the **return_length_64** argument cannot be written by the caller.
SS$_IVREGID	Invalid region ID specified. This condition value is returned if P0, P1, or P2 space is specified since these regions cannot be deleted, or if no region exists for the specified ID.
SS$_LEN_NOTPAGMULT	The **length_64** argument is not a multiple of CPU-specific pages.
SS$_PAGNOTINREG	A page in the specified range is not within the specified region.
SS$_PAGOWNVIO	A page in the specified range is owned by an access mode more privileged than the access mode of the caller.
SS$_VA_NOTPAGALGN	The **start_va_64** argument is not a CPU-specific page-aligned address.

$DEQ
Dequeue Lock Request

Dequeues (unlocks) granted locks; dequeues the sublocks of a lock; or cancels an ungranted lock request. The calling process must have previously acquired the lock or queued the lock request by calling the Enqueue Lock Request ($ENQ) service.

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

Format

SYS$DEQ [lkid] ,[valblk] ,[acmode] ,[flags]

Arguments

lkid

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

Lock identification of the lock to be dequeued. The **lkid** argument specifies this lock identification.

Note that if you do not specify the **lkid** argument, you must specify the LCK$M_DEQALL flag in the **flags** argument.

When you specify the LCK$M_DEQALL flag in the **flags** argument, different values (or no value) for the **lkid** argument produce varying behavior:

- When you do not specify the **lkid** argument (or specify it as 0) and you do specify the LCK$M_DEQALL flag, $DEQ dequeues all locks held by the process, at access modes equal to or less privileged than the effective access mode, on all resources. The effective access mode is the least privileged of the caller's access mode and the access mode specified in the **acmode** argument.
- When you specify the **lkid** argument as a nonzero value together with the LCK$M_DEQALL flag, $DEQ dequeues all sublocks of the lock identified by **lkid**; it does not dequeue the lock identified by **lkid**. For this operation, $DEQ ignores the LCK$M_CANCEL flag if it is set. A sublock of a lock is a lock that was created when the **parid** argument in the call to $ENQ was specified, where **parid** is the lock ID of the parent lock.

If you omit the **lkid** argument (or specify it as 0) and the LCK$M_DEQALL flag is not set, the $DEQ service returns the invalid lock ID condition value (SS$_IVLOCKID).

valblk

OpenVMS usage:	lock_value_block
type:	longword (unsigned)
access:	modify
mechanism:	by 32-bit or 64-bit reference (Alpha) by 32-bit reference (VAX)

Lock value block for the resource associated with the lock to be dequeued. The **valblk** argument is the 32-bit 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 LCK$M_DEQALL flag, you cannot use this argument.

When a protected write (PW) or exclusive (EX) mode lock is being dequeued and you specify a lock value block in the **valblk** argument, the contents of that lock value block are written to the lock value block in the lock database. Further, if the lock value block in the lock database was marked as invalid, that condition is cleared; the block becomes valid.

acmode

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

Access mode of the lock to be dequeued. The **acmode** argument is a longword containing the access mode.

The **acmode** argument is valid only if the LCK$M_DEQALL flag of the **flags** argument is set. The $PSLDEF macro defines the following symbols for the four access modes.

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

When dequeuing locks, $DEQ maximizes the access mode of the caller and the specified **acmode** argument. The maximized access mode is the less privileged of the caller's access mode and the **acmode** argument. If you do not specify the **acmode** argument, $DEQ uses the caller's access mode. Only those locks with an access mode that is equal to or less than the maximized access mode are dequeued.

flags

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

Flags specifying options for the $DEQ operation. The **flags** argument is a longword bit mask that is the logical OR of each bit set, where each bit corresponds to an option.

Note that if you do not specify the **lkid** argument, you must specify the LCK$M_DEQALL flag in the **flags** argument.

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

Flag	Description
LCK$M_DEQALL	When you specify this flag, $DEQ dequeues multiple locks, depending on the value of the **lkid** argument. Refer to the description of the **lkid** argument for details. The **acmode** argument is ignored if the LCK$M_DQALL flag is not set. If you specify LCK$M_DEQALL, the LCK$M_CANCEL flag, if set, is ignored.
LCK$M_CANCEL	When you specify this flag, $DEQ attempts to cancel a lock request that was queued by $ENQ. You can cancel only a waiting request. When the request is canceled, $DEQ returns the condition value SS$_NORMAL. If you attempt to cancel a granted lock, the request fails and $DEQ returns the condition value SS$_CANCELGRANT. There are two types of waiting requests that can be canceled: • A request for a new lock • A request to convert an existing lock When canceling a new lock request, the following action is taken: • If a completion AST was requested, the AST is queued for delivery and SS$_ABORT is stored in the lock status block. When canceling a request to convert an existing lock, the conversion request is canceled. The existing granted lock remains unchanged. The following specific actions are taken: • The blocking AST address specified for the existing granted lock is queued for delivery if the granted mode of the existing lock is blocking other waiting requests. • If a completion AST was specified by the conversion request, the completion AST is queued for delivery with SS$_CANCEL status stored in the lock status block that was specified by the conversion request. If you specify the LCK$M_DEQALL flag, the LCK$M_CANCEL flag is ignored.
LCK$M_INVVALBLK	When you specify this flag, $DEQ marks the lock value block, which is maintained for the resource in the lock database, as invalid. The lock value block remains marked as invalid until it is again written to. The Description section of the $ENQ service provides additional information about lock value block invalidation. This flag is ignored if (1) the lock mode of the lock being dequeued is not protected write or exclusive, or (2) you specify the LCK$M_CANCEL flag.

Description

The Dequeue Lock Request system service dequeues (unlocks) granted locks and waiting lock requests. The calling process must have previously acquired the lock or queued the lock request by calling the Enqueue Lock Request ($ENQ) service.

Action taken by the $DEQ service depends on the current state (granted or waiting) and the type of lock request (new lock or conversion request) to be dequeued.

When dequeuing a granted lock, the $DEQ service returns the condition value SS$_NORMAL and the following specific action is taken:

- Any queued blocking ASTs that have not been delivered are removed from the process's AST queues.

There are two types of waiting requests that can be dequeued:

- A request for a new lock
- A request to convert an existing lock

When dequeuing a new lock request, the $DEQ service returns the condition value SS$_NORMAL and the following specific action is taken:

- If a completion AST was requested, the completion AST is queued for delivery with SS$_ABORT stored in the lock status block.

When dequeuing a lock for which there is a conversion request waiting, the existing lock and its conversion request are dequeued. The $DEQ service returns the condition value SS$_NORMAL and the following specific actions are taken:

- If a blocking AST was queued to the process, it is removed from the process's AST queue.
- If a completion AST was specified by the conversion request, the completion AST is queued for delivery with SS$_ABORT status stored in the lock status block that was specified by the conversion request.

When a protected write (PW) or exclusive (EX) mode lock is being dequeued and you specify a lock value block in the **valblk** argument, the contents of that lock value block are written to the lock value block in the lock database.

If you specify the LCK$M_INVVALBLK flag in the **flags** argument and the lock mode of the lock being dequeued is PW or EX, the lock value block in the lock database is marked as invalid whether or not a lock value block was specified in the **valblk** argument.

The $DEQ, $ENQ, $ENQW, and $GETLKI services together provide the user interface to the lock management facility. For additional information about lock management, refer to the descriptions of these other services and to the *OpenVMS Programming Concepts Manual*.

Required Access or Privileges

None

Required Quota

None

Related Services

$ENQ, $ENQW, $GETLKI, $GETLKIW

Condition Values Returned

SS$_NORMAL	The lock was dequeued successfully.
SS$_ACCVIO	The value block specified by the **valblk** argument cannot be accessed by the caller.
SS$_CANCELGRANT	The LCK$M_CANCEL flag in the **flags** argument was specified, but the lock request that $DEQ was to cancel had already been granted.
SS$_ILLRSDM	An illegal attempt to modify a value block was made.
SS$_IVLOCKID	An invalid or nonexistent lock identification was specified or the process does not have the privilege to dequeue a lock at the specified access mode.
SS$_SUBLOCKS	The lock has sublocks and cannot be dequeued.

$DEVICE_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]

Arguments

return_devnam

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

Buffer to receive the device name. The **return_devnam** argument is the address of a character string descriptor pointing to a buffer into which $DEVICE_SCAN writes the name of the first or next device that matches the specified search criteria. The maximum size of any device name is 64 bytes.

retlen

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

Length of the device name string returned by $DEVICE_SCAN. The **retlen** argument is the address of a word into which $DEVICE_SCAN writes the length of the device name string.

search_devnam

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

Name of the device for which $DEVICE_SCAN is to search. The **search_devnam** argument accepts the standard wildcard characters, the asterisk (*), which matches any sequence of characters, and the percent sign (%), which matches any one character. If the **search_devnam** argument does not include a wildcard character, an exact match is used for comparison. For example, to match all unit 0 *DU* devices on any controller, specify **DU%0*. This string is compared to the most complete device name (DVI$_ALLDEVNAM). Only uppercase characters are accepted.

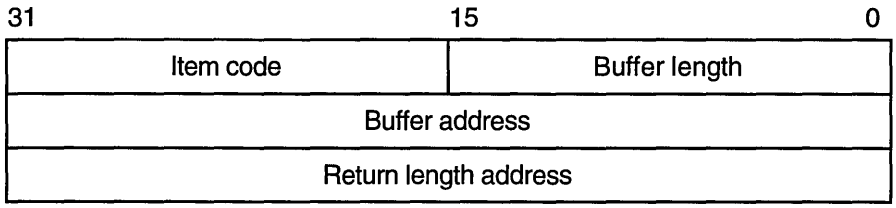
itmlst

OpenVMS usage:	item_list_3
type:	longword_unsigned
access:	read only
mechanism:	by reference

Item list specifying search criteria used to identify the device names for return by $DEVICE_SCAN. The **itmlst** argument is the address of a list of item descriptors,

each of which describes one search criterion. The list of item descriptors is terminated by a longword of 0.

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



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 from which $DEVICE_SCAN is to read the information. The length of the buffer needed depends upon the item code specified in the item code field of the item descriptor.
Item code	A word containing a user-specified symbolic code specifying the item of information that $DEVICE_SCAN is to return. The $DVSDEF macro defines these codes. Each item code is described in the Item Codes section.
Buffer address	A longword containing the address of the buffer from which $DEVICE_SCAN is to read the information.
Return length address	A longword containing the user-supplied address of the buffer from which $DEVICE_SCAN is to read the information.

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 Device Scan system service returns the names of all devices that match a specified set of search criteria. The names returned by $DEVICE_SCAN can then be passed to another service; for example, $GETDVI or $MOUNT.

The device names are returned for one process per call. A context value is used to continue multiple calls to $DEVICE_SCAN.

$DEVICE_SCAN allows wildcard searches based on device names, device classes, and device types. It also provides the ability to perform a wildcard search on other device-related services.

$DEVICE_SCAN makes it possible to combine search criteria. For example, to find only RA82 devices, use the following selection criteria:

```
DVS$_DEVCLASS = DC$_DISK and DVS$_DEVTYPE = DT$_RA82
```

To find all mailboxes with *MB* as part of the device name (excluding mailboxes such as NLA0), use the following selection criteria:

```
DVS$_DEVCLASS = DC$_MAILBOX and DEVNAM = *MB*
```

Required Access or Privileges

None

Required Quota

None

Related Services

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

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The **search_devnam**, **itmlst**, or **contxt** argument cannot be read by the caller, or the **retlen**, **return_devnam**, or **contxt** argument cannot be written by the caller.
SS$_BADPARAM	The **contxt** argument contains an invalid value, or the item list contains an invalid item code.

SS$_NOMOREDEV	No more devices match the specified search criteria.
SS$_NOSUCHDEV	The specified device does not exist on the host system.

$DGBLSC
Delete Global Section

Marks an existing permanent global section for deletion. The actual deletion of the global section takes place when all processes that have mapped the global section have deleted the mapped pages.

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

Format

SYS$DGBLSC [flags] ,gsdnam ,[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.

gsdnam

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

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.

ident

OpenVMS usage:	section_id
type:	quadword (unsigned)
access:	read only
mechanism:	by 32-bit or 64-bit reference (Alpha) 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-bit 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 eight 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 three bits, the matching criteria. The valid values, the symbolic names by which they can be specified, and their meanings are listed in the following table.

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

If you specify no address or specify it as 0 (the default), the version number and match control fields default to 0.

Description

The Delete Global Section service marks an existing permanent global section for deletion. The actual deletion of the global section takes place when all processes that have mapped the global section have deleted the mapped pages.

After a global section has been marked for deletion, any process that attempts to map it receives the warning return status code SS$_NOSUCHSEC.

Temporary global sections are automatically deleted when the count of processes using the section goes to 0.

VAX

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, $EXPREG, $LCKPAG, $LKWSET, $MGBLSC, $PURGWS, $SETPRT, $SETSTK, $SETSWM, $ULKPAG, $ULWSET, $UPDSEC, $UPDSECW

The $DGBLSC service does not unmap a global section from a process's virtual address space. To do this, the process should call the Delete Virtual Address Space ($DELTVA or $DELTVA_64) service, which deletes the pages to which the section is mapped.

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The global section name or name descriptor or the section identification field cannot be read by the caller.
SS$_INTERLOCK	The bit map lock for allocating global sections from the specified shared memory is locked by another process.
SS$_IVLOGNAM	The global section name has a length of 0 or has more than 15 characters.
SS$_IVSECFLG	You set an invalid flag, reserved flag, or flag requiring a user privilege.
SS$_IVSECIDCTL	The section identification match control field is invalid.
SS$_NOPRIV	The caller does not have the privilege to delete a system global section, does not have read/write access to a group global section, or does not have the privilege to delete a global section located in memory that is shared by multiple processors.
SS$_NOSUCHSEC	The specified global section does not exist, or the identifications do not match.
SS$_NOTCREATOR	The section is in memory shared by multiple processors and was created by a process on another processor.
†SS$_SHMNOTCNCT	The shared memory named in the **name** argument is not known to the system. This error can be caused by a spelling error in the string, an improperly assigned logical name, or the failure to identify the multiport memory as shared at system generation time.
SS$_TOOMANYLNAM	The logical name translation of the **gsdnam** string exceeded the allowed depth of 10.

†VAX specific

$DISMOU
Dismount Volume

Dismounts a mounted volume or volume sets.

Format

SYS$DISMOU devnam ,[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.

Flag	Meaning
DMT$M_CLUSTER	The volume is to be dismounted clusterwide, that is, from all nodes in the VMScluster system. $DISMOU dismounts the volume from the caller's node first and then from every other node in the existing cluster. DMT$M_CLUSTER dismounts only system or group volumes. To dismount a group volume clusterwide, the caller must have GRPNAM privilege. To dismount a system volume clusterwide, the caller must have SYSNAM privilege. DMT$M_CLUSTER has no effect if the system is not a member of a cluster. DMT$M_ CLUSTER applies only to disks.
DMT$M_NOUNLOAD	Specifies that the volume is not to be physically unloaded after the dismount. If both the DMT$M_UNLOAD and DMT$M_NOUNLOAD flags are specified, the DMT$M_NOUNLOAD flag is ignored. If neither flag is specified, the volume is physically unloaded, unless the DMT$M_NOUNLOAD flag was specified on the $MOUNT system service or the /NOUNLOAD qualifier was specified on the MOUNT command when the volume was mounted.
DMT$M_OVR_CHECKS	Specifies that the volume should be dismounted without checking for open files, spooled devices, installed images, or installed swap and page files.
DMT$M_UNIT	The specified device, rather than the entire volume set, is dismounted.
DMT$M_UNLOAD	Specifies that the volume is to be physically unloaded after the dismount. If both the DMT$M_UNLOAD and DMT$M_NOUNLOAD flags are specified, the DMT$M_NOUNLOAD flag is ignored. If neither flag is specified, the volume is physically unloaded, unless the DMT$M_NOUNLOAD flag was specified on the $MOUNT system service or the /NOUNLOAD qualifier was specified on the MOUNT command when the volume was mounted.

Description

The Dismount Volume service dismounts a mounted volume or volume sets. To dismount a private volume, the caller must own the volume.

When you issue the $DISMOU service, $DISMOU removes the volume from your list of mounted volumes, deletes the logical name (if any) associated with the volume, and decrements the mount count.

If the mount count does not equal 0 after being decremented, $DISMOU does not mark the volume for dismounting (because the volume must have been mounted shared). In this case, the total effect for the issuing process is that the process is denied access to the volume and a logical name entry is deleted.

If the mount count equals 0 after being decremented, $DISMOU marks the volume for dismounting. After marking the volume for dismounting, $DISMOU waits until the volume is idle before dismounting it. A native volume is idle when no user has an open file to the volume, and a foreign volume is idle when no channels are assigned to the volume.

Native volumes are Files–11 structured disks or ANSI-structured tapes. Foreign volumes are not Files–11 or ANSI structured media.

After a volume is dismounted, nonpaged pool is returned to the system. Paged pool is also returned if you mounted the volume using the /GROUP or /SYSTEM qualifier.

If a volume is part of a Files–11 volume set and the flag bit DMT$V_UNIT is not set, the entire volume set is dismounted.

When a Files–11 volume has been marked for dismount, new channels can be assigned to the volume, but no new files can be opened.

Note that the SS$_NORMAL status code indicates only that $DISMOU has successfully performed one or more of the actions just described: decremented the mount count, marked the volume for dismount, or dismounted the volume. The only way to determine that the dismount has actually occurred is to check the device characteristics using the Get Device/Volume Information ($GETDVI) service.

By specifying the DVI$_DEVCHAR item code in a call to $GETDVI, you can learn whether a volume is mounted (it is if the DEV$V_MNT bit is set) or whether it is marked for dismounting (it is if the DEV$M_DMT bit is set). If DEV$V_MNT is clear or if DEV$M_DMT is set, the mount count is 0.

Required Access or Privileges

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

- GRPNAM privilege to dismount a volume mounted with the /GROUP qualifier
- SYSNAM privilege to dismount a volume mounted with the /SYSTEM qualifier

Required Quota

None

Related Services

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

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The device name descriptor cannot be read or does not describe a readable device name.
SS$_DEVALLOC	The device is allocated to another process and cannot be dismounted by the caller.
SS$_DEVOFFLINE	The specified device is not available.
SS$_DEVNOTMOUNT	The specified device is not mounted.
SS$_IVDEVNAM	The device name string is not valid.
SS$_IVLOGNAM	The device logical name has a length of 0 or is longer than the allowable logical name length.
SS$_NOGRPNAM	GRPNAM privilege is required to dismount a volume mounted for groupwide access.
SS$_NOIOCHAN	No I/O channel is available. To use $DISMOU, a channel must be assigned to the volume.
SS$_NONLOCAL	The device is on a remote node.
SS$_NOSUCHDEV	The specified device does not exist.
SS$_NOSYSNAM	SYSNAM privilege is required to dismount a volume mounted for systemwide access.
SS$_NOTFILEDEV	The specified device is not file structured.

$DISPLAY_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]

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 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 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.

rem_user

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

Remote user name of the proxy about which information is being requested. The **rem_user** argument is the address of a character-string descriptor pointing to the user name string.

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

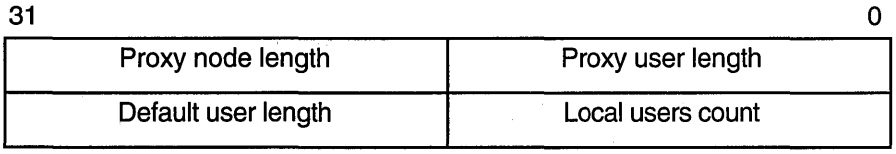
The **rem_user** argument can be specified in user identification code (UIC) format ([*group*, *member*]). Brackets are allowed only if the remote user name string specifies a UIC. Group and member are character-string representations of octal numbers with no leading zeros.

Asterisk (*) and percent sign (%) wildcards are allowed for the remote user specification. If you specify wildcards for the **rem_user** argument, the server searches the entire proxy database for matches to the remote node and remote user you specified. If a match is found, information about the matched proxy is

returned. See the Description section for information about retrieving information about multiple proxies.

buffer_sizes
OpenVMS usage: return length block
type: array of 4 words (unsigned)
access: write only
mechanism: by reference

Array of return lengths for various input buffers. The **buffer_sizes** argument is the address of an array of four words with the following format.



ZK-6169A-GE

The following table defines the **buffer_sizes** fields.

Descriptor Field	Definition
Proxy user length	Return length (in bytes) of the **rem_user** argument. The proxy user length field contains a value in the range of 0 to 32. A value of 0 in this field indicates that the service has failed or that there was no match for the user specified by the **rem_user** argument.
Proxy node length	Return length (in bytes) of the **rem_node** argument. A value of 0 in this field indicates that the service has failed or that there was no match for the node specified by the **rem_node** argument. The proxy node length field contains values in the range of 0 to 1024.
Local users count	Number of local users associated with the matched proxy. The local users count field contains a value in the range of 0 to 16. A value of 0 indicates that the matched proxy had no local users.
Default user length	Return length (in bytes) of the **default_user** argument. The default user length field contains a value in the range of 0 to 32. A value of 0 in this field indicates that the matched proxy did not have a default user.

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

Node name of a proxy matching the remote node name specified by the **rem_node** argument and the remote user name specified by the **rem_user** argument. The **proxy_node** argument is the address of a character-string descriptor pointing to a buffer to receive the proxy node name.

The descriptor's buffer must be 1024 bytes long to receive a node name. The length of the returned node name is specified by the proxy node length field returned in the buffer specified by the **buffer_sizes** argument.

proxy_user

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

User name of a proxy matching the remote node name specified by the **rem_node** argument and the remote user name specified by the **rem_user** argument. The **proxy_user** argument is a character-string descriptor pointing to a buffer to receive the remote user name of a proxy.

The descriptor's buffer must be 32 bytes long to receive a user name. The length of the returned user name is specified by the proxy user length field returned in the buffer specified by the **buffer_sizes** argument.

default_user

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

Default user of a proxy matching the node name specified by the **rem_node** argument and the remote user name specified by the **rem_user** argument. The **default_user** argument is the address of a character-string descriptor pointing to a buffer to receive the default user name.

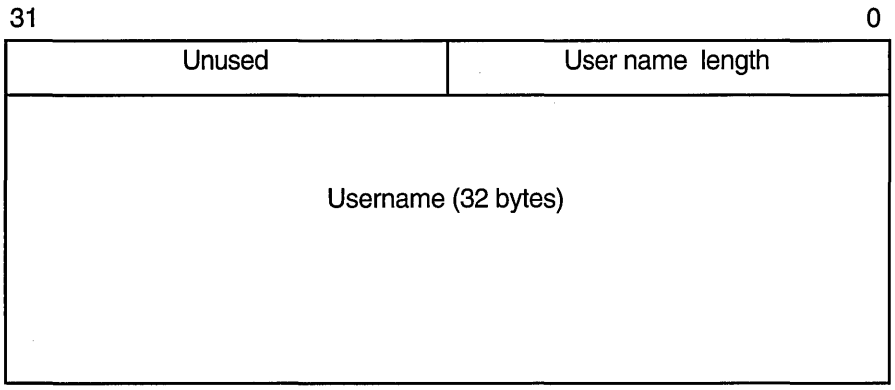
The descriptor's buffer must be 32 bytes long to receive a user name. The length of the returned user name is specified in the default user length field in the buffer specified by the **buffer_sizes** argument.

local_users

OpenVMS usage:	buffer
type:	array of 0 to 16 user name buffers
access:	write only
mechanism:	by reference

Array of local user names associated with a proxy matching the remote node name specified by the **rem_node** argument and the remote user name specified by the **rem_user** argument. The **local_users** argument is the address of a buffer to receive an array of local user names.

Each element in the array is a 36-byte block with the following format.



ZK-6170A-GE

The following table defines the **local_users** fields.

Descriptor Field	Definition
User name length	Length (in bytes) of the associated username string. The length can be in the range of 1 to 32 bytes.
Username	A fixed 32-byte blank padded character string containing a local user name associated with the matched proxy.

The buffer specified by the **local_users** argument must be able to contain up to 16 user name buffers. Therefore, the buffer length must be 576 bytes.

The number of elements returned in the buffer is specified in the local users count field returned in the buffer specified by the **buffer_sizes** argument.

flags

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

Functional specification for the service and type of user the **local_user** argument represents. The **flags** argument is a longword bit mask wherein each bit corresponds to an option.

Each flag option has a symbolic name. The $PRXDEF macro defines the following symbolic name.

Symbolic Name	Description
PRX$M_BYPASS_EXPAND	The service should not convert the node name specified in the **rem_node** argument to its corresponding DECnet 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.

Symbolic Name	Description
PRX$M_EXACT	The service should match exactly the remote node and remote user and ignore wildcards.

context

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

Context information to keep between related calls to the $DISPLAY_PROXY service. The **context** argument is the address of a longword to receive a context from the $DISPLAY_PROXY service.

The initial value contained in the longword pointed to by the **context** argument must be 0. The contents of the unsigned longword must not be changed after the service has set its value. If the contents of the buffer pointed to by the **context** argument are changed between calls to the $DISPLAY_PROXY service, the service will return SS$_BADCONTEXT. If the contents of the **context** argument are changed between calls to the $DISPLAY_PROXY service, you can change the value of the **context** argument back to 0 to start the search over again.

Contexts become invalid after one-half hour of non-use. This means that if you call the $DISPLAY_PROXY service with a wildcard **rem_node** or **rem_user**, and do not call the service to get the next matching record within one-half hour, the context becomes invalid. If the context has become invalid, you must start your search of the proxy database over from its beginning by resetting the context to 0.

Description

The Display Proxy service returns to the caller all information about a specified proxy in the proxy database.

Wildcards can be specified for the **rem_node** and **rem_user** arguments. Because $DISPLAY_PROXY can return information about only one matching proxy at a time, you must call this service repeatedly with the **context** argument to retrieve information about all matching proxies. $DISPLAY_PROXY returns SS$_NOMOREITEMS when information about all of the matching proxies has been returned. No proxy information is returned from the call that returns the SS$_NOMOREITEMS status.

Required Access or Privileges

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

Required Quota

None

Related Services

$ADD_PROXY, $DELETE_PROXY, $VERIFY_PROXY

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The **rem_node** or **rem_user** argument cannot be read by the service; or the **buffer_sizes**, **proxy_node**, **proxy_user**, **default_user**, or **local_users** argument cannot be written by the service; or the **context** argument cannot be read or written by the service.
SS$_BADBUFLEN	The length of the **rem_node**, **rem_user**, **proxy_node**, **proxy_user**, **default_user**, or **local_users** argument was out of range.
SS$_BADCONTEXT	The **context** argument did not contain a 0 on the first call to the service, or the **context** argument's value changed between consecutive calls to the service.
SS$_NOMOREITEMS	Information about all proxies matching the specification of the **rem_node** and **rem_user** arguments has been returned by the service.
SS$_NOREADALL	The caller does not have access to the proxy database.

This service can also return any of the following messages passed from the security server, or any OpenVMS RMS error message encountered during operations on the proxy database:

SECSRV$_BADNODENAMELEN	The node name length is out of range.
SECSRV$_BADREMUSERLEN	The remote user name length is out of range.
SECSRV$_NOSUCHPROXY	The proxy specified by the **rem_node** and **rem_user** arguments does not exist in the proxy database.
SECSRV$_NOSUCHUSER	The specified local user does not exist in the proxy's local user list, or is not the proxy's default user.
SECSRV$_PROXYNOTACTIVE	Proxy processing is currently stopped. Try the request again later.
SECSRV$_SERVERNOTACTIVE	The security server is not currently active. Try the request again later.

$DLCEFC
Delete Common Event Flag Cluster

Marks a permanent common event flag cluster for deletion.

Format

SYS$DLCEFC name

Argument

name

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

Name of the common event flag cluster to be deleted. The **name** argument is the address of a character string descriptor pointing to the name of the cluster.

The names of event flag clusters are unique to UIC groups, and the UIC group number of the calling process is part of the name.

Description

The Delete Common Event Flag Cluster service marks a permanent common event flag cluster for deletion. The cluster is actually deleted when no more processes are associated with it. The $DLCEFC service does not disassociate a process from a common event flag cluster; the Disassociate Common Event Flag Cluster ($DACEFC) service does this. However, the system disassociates a process from an event flag cluster at image exit.

If the cluster has already been deleted or does not exist, the $DLCEFC service returns the status code SS$_NORMAL.

Required Access or Privileges

Delete access is required.

Required Quota

None

Related Services

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

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_IVLOGNAM	The cluster name string has a length of 0 or has more than 15 characters.

SS$_NOPRIV	The process does not have the privilege to delete a permanent common event flag cluster, or the process does not have the privilege to delete a common event flag cluster in memory shared by multiple processors.

$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.

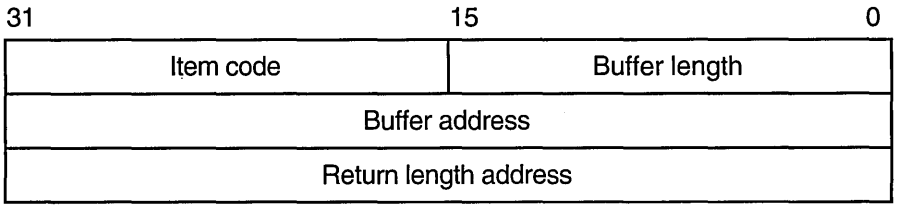
itmlst

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. 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.

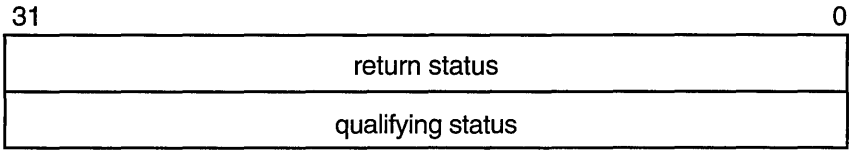
Descriptor Field	Definition
Buffer length	A word specifying the length of the buffer; the buffer either supplies information to be used by $DNS or receives information from $DNS. The required length of the buffer varies, depending on the item code specified. Each item code description specifies the required 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.



ZK-1080A-GE

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, DNS$V_DNSB_INOUTDIRECT and DNS$V_DNSB_OUTLINKED, are defined as follows:

- DNS$V_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.

- DNS$V_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.

Function Codes

DNS$_ADD_REPLICA

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

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 DNSCTS, DNSUTS, DNS$Class, DNS$ClassVersion, and DNS$ACS. The name service creates the DNSCTS, DNSUTS, and DNS$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

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.

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 SS$_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 SS$_NORMAL. Subsequent calls return the child directories, starting with the directory specified in DNS$_CONTEXTVARNAME and continuing in alphabetical order.

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$V_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 SS$_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.

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$V_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. The function can also be used 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.

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 return 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$V_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

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$_RESOLVE_NAME any time they receive the qualifying status DNS$V_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$K_SOFTLINK and request the attribute DNS$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$V_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$V_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$_INOUTDIRECT argument, on input, controls the scope of the search. If you set this item code to 1, the clerk searches only the top-level group. If you set it to 0, the clerk searches all of the subgroups. On output, the clerk returns a 1 in the DNS$V_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.

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$V_DNSB_INOUTDIRECT
DNS$V_DNSB_OUTLINKED

Item Codes

Table SYS1–4 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 SYS1–4 $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$_SET) or a single value (DNS$K_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 DNS$K_HIGH.
DNS$_CONTEXTVARNAME	An opaque simple name.
DNS$_CONTEXTVARTIME	A creation timestamp (CTS).
DNS$_DIRECTORY	An opaque full name of a directory.

(continued on next page)

Table SYS1–4 (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 DNSK_OBJECT, DNSK_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)

Table SYS1–4 (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.

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 a simple or full name in string format for the parse functions DNS$_PARSE_FULLNAME_STRING and DNS$_PARSE_SIMPLENAME_STRING that is to be converted to opaque format.

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.

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 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$K_SECONDARY) or a read-only replica (DNS$K_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 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.

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 SYS1–4 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 particular 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 for OpenVMS I/O on behalf of the DECdns client. It requires system dynamic memory to construct a database to queue the I/O request and may 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, see 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 may 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.

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$_NAMESERVERBUG	A name server encountered an implementation bug. Please submit an SPR.
DNS$_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.

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 may or may 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.

$DNSW (VAX Only)
Distributed Name Service Clerk and Wait

On VAX systems, the DECdns clerk 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]]

Arguments

efn

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

Number of the event flag that is set when the service completes. If this argument is omitted, event flag 0 is set.

flags

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

Flags specifying options for the service. The **flags** argument is a longword bit mask in which each bit corresponds to an option flag. The $DDTMDEF macro defines symbolic names for these option flags. The flag currently defined is shown in the following table. All undefined bits must be 0. If this argument is omitted, no flag is set.

Flag	Description
DDTM$M_SYNC	Set this flag to specify that successful synchronous completion is to be indicated by returning SS$_SYNCH. When SS$_SYNCH is returned, the AST routine is not called, the event flag is not set, and the I/O status block is not filled in.

iosb

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

I/O status block in which the following information is returned:

- The completion status of the service. This is returned as a condition value. See the Condition Values Returned section.
- The outcome of the transaction.

  If the service returns SS$_NORMAL, the outcome of the transaction is commit. If the service returns SS$_ABORT, the outcome of the transaction is abort.

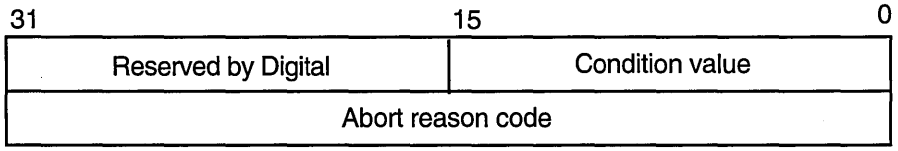
- An abort reason code that gives one reason why the transaction aborted, if the completion status of the service is SS$_ABORT.

The $DDTMMSGDEF macro defines symbolic names for these abort reason codes; those currently defined are shown in Table SYS1–5.

Table SYS1–5 Abort Reason Codes

Symbolic Name	Description
DDTM$_ABORTED	The application aborted the transaction.
DDTM$_COMM_FAIL	A communications link failed.
DDTM$_INTEGRITY	A resource manager integrity constraint check failed.
DDTM$_LOG_FAIL	A write operation to the transaction log failed.
DDTM$_PART_SERIAL	A resource manager serialization check failed.
DDTM$_PART_TIMEOUT	The timeout specified by a resource manager expired.
DDTM$_SEG_FAIL	A process or image terminated.
DDTM$_SERIALIZATION	A DECdtm transaction manager serialization check failed.
DDTM$_SYNC_FAIL	The transaction was not globally synchronized.
DDTM$_TIMEOUT	The timeout specified on $START_TRANS expired.
DDTM$_UNKNOWN	The reason is unknown.
DDTM$_VETOED	A resource manager was unable to commit the transaction.

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



ZK–3667A–GE

astadr

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

AST routine that is executed when the service completes. The **astadr** argument is the address of this routine. 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.

$END_TRANS returns the outcome of the transaction. If it completes successfully, the outcome of the transaction is commit. If it returns the SS$_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.

$END_TRANS does not complete either successfully or with the SS$_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.

- 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$_CURTIDCHANGE	The **tid** argument was omitted and a call to change the default transaction of the calling process was in progress.
SS$_EXASTLM	The process AST limit (ASTLM) was exceeded.
SS$_ILLEFC	The event flag number was invalid.
SS$_INSFARGS	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 **tid** argument was omitted), but the calling process did not have a default transaction.
SS$_NOLOG	The local node did not have a transaction log.
SS$_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.

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 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]]

$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, refer to the Synchronize ($SYNCH) service.

Format

SYS$ENQ [efn] ,lkmode ,lksb ,[flags] ,[resnam] ,[parid] ,[astadr] ,[astprm] ,[blkast] ,[acmode] ,[rsdm_id] ,[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 the request has been granted or canceled. Cancellation occurs if you use $DEQ with the cancel modifier or if the waiting request is chosen to break a deadlock. The **efn** argument is a longword containing this number; however, $ENQ uses only the low-order byte.

Upon request initiation, $ENQ clears the specified event flag (or event flag 0 if **efn** was not specified). Then, when the lock request is granted, the specified event flag (or event flag 0) is set unless you specified the LCK$M_SYNCSTS flag in the **flags** argument.

lkmode

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

Lock mode requested. The **lkmode** argument is a longword specifying this lock mode.

Each lock mode has a symbolic name. The $LCKDEF macro defines these symbolic names. The following table gives the symbolic name and description for each lock mode.

Lock Mode	Description
LCK$K_NLMODE	Null mode. This mode grants no access to the resource but serves rather as a placeholder and indicator of future interest in the resource. The null mode does not inhibit locking at other lock modes; further, it prevents the deletion of the resource and lock value block, which would otherwise occur if the locks held at the other lock modes were dequeued.
LCK$K_CRMODE	Concurrent read. This mode grants the caller read access to the resource while permitting write access to the resource by other users. This mode is used to read data from a resource in an unprotected manner, because other users can modify that data as it is being read. This mode is typically used when additional locking is being performed at a finer granularity with sublocks.
LCK$K_CWMODE	Concurrent write. This mode grants the caller write access to the resource while permitting write access to the resource by other users. This mode is used to write data to a resource in an unprotected fashion, because other users can simultaneously write data to the resource. This mode is typically used when additional locking is being performed at a finer granularity with sublocks.
LCK$K_PRMODE	Protected read. This mode grants the caller read access to the resource while permitting only read access to the resource by other users. Write access is not allowed. This is the traditional *share lock.*
LCK$K_PWMODE	Protected write. This mode grants the caller write access to the resource while permitting only read access to the resource by other users; the other users must have specified concurrent read mode access. No other writers are allowed access to the resource. This is the traditional *update lock.*
LCK$K_EXMODE	Exclusive. The exclusive mode grants the caller write access to the resource and allows no access to the resource by other users. This is the traditional *exclusive lock.*

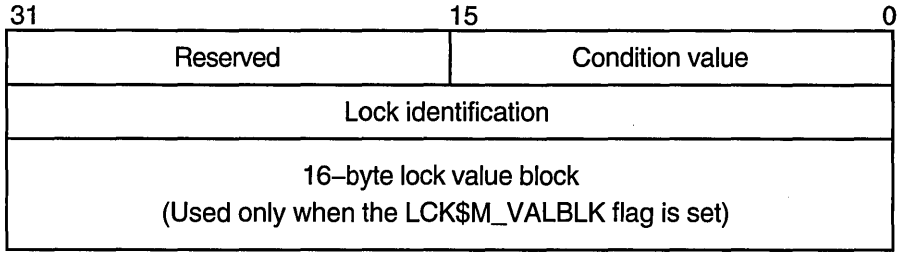
lksb

OpenVMS usage:	lock_status_block
type:	longword (unsigned)
access:	write only
mechanism:	by 32-bit or 64-bit reference (Alpha) 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-bit 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. When you specify the LCK$M_VALBLK flag in the **flags** argument, the lock status block contains a lock value block; in this case, the 16-byte lock value block appears beginning at the first byte following the eighth byte of the lock status block, bringing the total length of the lock status block to 24 bytes.

The following diagram shows the format of the lock status block and the optional lock value block.



ZK–1708–GE

The following table defines the status block fields.

Status Block Field	Definition
Condition value	A word in which $ENQ writes a condition value describing the final disposition of the lock request; for example, whether the lock was granted, converted, and so on. The condition values returned in this field are described in the Condition Values Returned in the Lock Status Block section, which appears following the list of condition values returned in R0.
Reserved	A word reserved to Digital.
Lock identification	A longword containing the identification of the lock. For a new lock, $ENQ writes the lock identification of the requested lock into this longword when the lock request is queued. For a lock conversion on an existing lock, you must supply the lock identification of the existing lock in this field.

Status Block Field	Definition
Lock value block	A user-defined, 16-byte structure containing information about the resource. This information is interpreted only by the user program. When a process acquires a lock on a resource, the lock management facility provides that process with a process-private copy of the lock value block associated with the resource, provided that process has specified the LCK$M_VALBLK flag in the **flags** argument. The copy provided to the process is a copy of the lock value block stored in the lock manager's database. The copy of the lock value block maintained in the lock database is updated in the following way: whenever a process either (1) dequeues a lock at protected write (PW) or exclusive (EX) mode or (2) converts a lock at one of these modes to the same lock mode, the operating system stores the caller's lock value block in the lock database, provided the caller has specified the LCK$M_VALBLK flag.

Callers of $ENQ are provided with copies of the updated lock value block from the lock database in the following way: when $ENQ grants a new lock to the caller or converts the caller's existing lock to 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.

Flag	Description
LCK$M_SYNCSTS	When you specify this flag, $ENQ returns the successful condition value SS$_SYNCH in R0 if the lock request is granted immediately; in this case, no completion AST is delivered and no event flag is set. If the lock request is queued successfully but cannot be granted immediately, $ENQ returns the condition value SS$_NORMAL in R0; then when the request is granted, $ENQ sets the event flag and queues an AST if the **astadr** argument was specified.
LCK$M_SYSTEM	When you specify this flag, the resource name is interpreted as systemwide. By default, resource names are qualified by the UIC group number of the creating process. This flag is ignored in lock conversions.
LCK$M_VALBLK	When you specify this flag, the lock status block contains a lock value block. See the description of the **lksb** argument for more information.
LCK$M_CONVERT	When you specify this flag, $ENQ performs a lock conversion. In this case, the caller must supply (in the second longword of the lock status block) the lock identification of the lock to be converted.
LCK$M_NODLCKWT	By specifying this flag, a process indicates to the lock management services that it is not blocked from execution while waiting for the lock request to complete. For example, a lock request might be left outstanding on the waiting queue as a signaling device between processes. This flag helps to prevent false deadlocks by providing the lock management services with additional information about the process issuing the lock request. When you set this flag, the lock management services do not consider this lock when trying to detect deadlock conditions. A process should specify the LCK$M_NODLCKWT flag only in a call to the $ENQ system service. The $ENQW system service waits for the lock request to be granted before returning to the caller; therefore, specifying the LCK$M_NODLCKWT flag in a call to the $ENQW system service defeats the purpose of the flag and can result in a genuine deadlock being ignored. The lock management services make use of the LCK$M_NODLCKWT flag only when the lock specified by the call to $ENQ is in either the waiting or the conversion queue. Improper use of the LCK$M_NODLCKWT flag can result in the lock management services ignoring genuine deadlocks.

Flag	Description
LCK$M_NODLCKBLK	By specifying this flag, a process indicates to the lock management services that, if this lock is blocking another lock request, the process intends to give up this lock on demand. When you specify this flag, the lock management services do not consider this lock as blocking other locks when trying to detect deadlock conditions. A process typically specifies the LCK$M_NODLCKBLK flag only when it also specifies a blocking AST. Blocking ASTs notify processes with granted locks that another process with an incompatible lock mode has been queued to access the same resource. Use of blocking ASTs can cause false deadlocks, because the lock management services detect a blocking condition, even though a blocking AST has been specified; however, the blocking condition will disappear as soon as the process holding the lock executes, receives the blocking AST, and dequeues the lock. Specifying the LCK$M_NODLCKBLK flag prevents this type of false deadlock. To enable blocking ASTs, the **blkast** argument of the $ENQ system service must contain the address of a blocking AST service routine. If the process specifies the LCK$M_NODLCKBLK flag, the blocking AST service routine should either dequeue the lock or convert it to a lower lock mode without issuing any new lock requests. If the blocking AST routine does otherwise, a genuine deadlock could be ignored. The lock management services make use of the LCK$M_NODLCKBLK flag only when the lock specified by the call to $ENQ has been granted. Improper use of the LCK$M_NODLCKBLK flag can result in the lock management services ignoring genuine deadlocks.

Flag	Description
LCK$M_NOQUOTA	This flag is reserved to Digital. When you set this flag, the calling process is not charged Enqueue Limit (ENQLM) quota for this new lock. The calling process must be running in executive or kernel mode to set this flag. This flag is ignored for lock conversions.
LCK$M_CVTSYS	This flag is reserved to Digital. When you set this flag, the lock is converted from a process-owned lock to a system-owned lock. The calling process must be running in executive or kernel mode to set this flag.
LCK$M_EXPEDITE	This flag is valid only for new lock requests. Specifying this flag allows a request to be granted immediately, provided the requested mode when granted would not block any currently queued requests in the resource conversion and wait queues. Currently, this flag is valid only for NLMODE requests. If this flag is specified for any other lock mode, the request will fail and an error of SS$_UNSUPPORTED will be returned.
LCK$M_QUECVT	This flag is valid only for conversion operations. A conversion request with the LCK$M_QUECVT flag set will be forced to wait behind any already queued conversions. The conversion request is granted immediately, if there are no already queued conversions. The QUECVT behavior is valid only for a subset of all possible conversions. Table SYS1–6 defines the legal set of conversion requests for LCK$M_QUECVT. Illegal conversion requests are failed with SS$_BADPARAM returned.

Table SYS1–6 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-bit or 64-bit descriptor–fixed-length string descriptor (Alpha) 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-bit 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-bit or 64-bit reference (Alpha) 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-bit 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-bit or 64-bit reference (Alpha)
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-bit 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.

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

Placeholding argument reserved to Digital.

Description

The Enqueue Lock Request service queues a new lock or lock conversion on a resource. The $ENQ service completes asynchronously; that is, it returns to the caller after queuing the lock request without waiting for the lock to be either granted or converted. For synchronous completion, use the Enqueue Lock Request and Wait ($ENQW) service. The $ENQW service is identical to the $ENQ service in every way except that $ENQW returns to the caller when the lock is either granted or converted.

The $ENQ service uses system dynamic memory for the creation of the lock and resource blocks.

When $ENQ queues a lock request, it returns the status of the request in R0 and writes the lock identification of the lock in the lock status block. Then, when the lock request is granted, $ENQ writes the final completion status in the lock status block, sets the event flag, and calls the AST routine if this has been requested.

When $ENQW queues a lock request, it returns status in R0 and in the lock status block when the lock has been either granted or converted. Where applicable, it simultaneously sets the event flag and calls the AST routine.

Invalidation of the Lock Value Block In some situations, the lock value block can become invalid. In these situations, $ENQ warns the caller by returning the condition value SS$_VALNOTVALID in the lock status block, provided the caller has specified the flag LCK$M_VALBLK in the **flags** argument.

The SS$_VALNOTVALID condition value is a warning message, not an error message. Therefore, the $ENQ service grants the requested lock and returns this warning on all subsequent calls to $ENQ until either a new lock value block is written to the lock database or the resource is deleted. Resource deletion occurs when no locks are associated with the resource.

The following events can cause the lock value block to become invalid:

- If any process holding a protected write or exclusive mode lock on a resource is terminated abnormally, the lock value block becomes invalid.
- If a node in a VMScluster 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 ($DEQ) service to dequeue this lock and specifies the flag LCK$M_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.

Required Quota

- Enqueue limit (ENQLM) quota
- AST limit (ASTLM) quota in lock conversion requests that you specify either the **astadr** or **blkast** argument

Related Services

$DEQ, $ENQW, $GETLKI, $GETLKIW, $SET_RESOURCE_DOMAIN

Condition Values Returned

SS$_NORMAL	The service completed successfully; the lock request was successfully queued.
SS$_SYNCH	The service completed successfully; the LCK$M_SYNCSTS flag in the **flags** argument was specified, and $ENQ was able to grant the lock request immediately.
SS$_ACCVIO	The lock status block or the resource name cannot be read.
SS$_BADPARAM	You specified an invalid lock mode in the **lkmode** argument.
SS$_CVTUNGRANT	You attempted a lock conversion on a lock that is not currently granted.
SS$_EXDEPTH	The limit of levels of sublocks has been exceeded.
SS$_EXENQLM	The process has exceeded its enqueue limit (ENQLM) quota.
SS$_INSFMEM	The system dynamic memory is insufficient for creating the necessary data structures.
SS$_IVBUFLEN	The length of the resource name was either 0 or greater than 31.
SS$_IVLOCKID	You specified an invalid or nonexistent lock identification, or the lock identified by the lock identification has an associated access mode that is more privileged than the caller's, or the access mode of the parent was less privileged than that of the caller.
SS$_NOLOCKID	No lock identification was available for the lock request.

SS$_NOSYSLCK	The LCK$M_SYSTEM flag in the **flags** argument was specified, but the caller lacks the necessary SYSLCK privilege.
SS$_NOTQUEUED	The lock request was not queued; the LCK$M_NOQUEUE flag in the **flags** argument was specified, and $ENQ was not able to grant the lock request immediately.
SS$_PARNOTGRANT	The parent lock specified in the **parid** argument was not granted.

Condition Values Returned in the Lock Status Block

SS$_NORMAL	The service completed successfully; the lock was successfully granted or converted.
SS$_ABORT	The lock was dequeued (by the $DEQ service) before $ENQ could grant the lock.
SS$_CANCEL	The lock conversion request has been canceled and the lock has been regranted at its previous lock mode. This condition value is returned when $ENQ queues a lock conversion request, the request has not been granted yet (it is in the conversion queue), and, in the interim, the $DEQ service is called (with the LCK$M_CANCEL flag specified) to cancel this lock conversion request. If the lock is granted before $DEQ can cancel the conversion request, the call to $DEQ returns the condition value SS$_CANCELGRANT, and the call to $ENQ returns SS$_NORMAL.
SS$_DEADLOCK	A deadlock was detected.
SS$_VALNOTVALID	The lock value block is marked invalid. This warning message is returned only if the caller has specified the flag LCK$M_VALBLK in the **flags** argument. Note that the lock has been successfully granted despite the return of this warning message. Refer to the Description section for a complete discussion of lock value block invalidation.

$ENQW
Enqueue Lock Request and Wait

The Enqueue Lock Request and Wait service queues a lock on a resource. The $ENQW service completes synchronously; that is, it returns to the caller when the lock has been either granted or converted. For asynchronous completion, use the Enqueue Lock Request ($ENQ) service; $ENQ returns to the caller after queuing the lock request, without waiting for the lock to be either granted or converted. In all other respects, $ENQW is identical to $ENQ. Refer to the $ENQ description for all other information about the $ENQW service.

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

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]

$ERAPAT
Get Security Erase Pattern

Generates a security erase pattern.

Format

SYS$ERAPAT [type] ,[count] ,[patadr]

Arguments

type

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

Type of storage to be written over with the erase pattern. The **type** argument is a longword containing the type of storage. The three storage types, together with their symbolic names, are defined by the $ERADEF macro and are listed in the following table.

Storage Type	Symbolic Name
Main memory	ERA$K_MEMORY
Disk	ERA$K_DISK
Tape	ERA$K_TAPE

count

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

Number of times that $ERAPAT has been called in a single security erase operation. The **count** argument is a longword containing the iteration count.

You should call the $ERAPAT service initially with the **count** argument set to 1, the second time with the **count** argument set to 2, and so on, until the status code SS$_NOTRAN is returned.

patadr

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

Security erase pattern to be written. The **patadr** argument is the address of a longword into which the security erase pattern is to be written.

Description

The Get Security Erase Pattern service generates a security erase pattern that can be written into memory areas containing outdated but sensitive data to make it unreadable. This service is used primarily by the operating system, but it can also be used by users who want to perform security erase operations on foreign disks.

You should call the $ERAPAT service iteratively until the completion status SS$_NOTRAN is returned.

The following example demonstrates how to use the $ERAPAT service to perform a security erase to a disk. Note that, after each call to $ERAPAT, a test for the status SS$_NOTRAN is made. If SS$_NOTRAN has not been returned, $QIO is called to write the pattern returned by $ERAPAT onto the disk. After this write, $ERAPAT is called again and the cycle is repeated until the code SS$_NOTRAN is returned, at which point the security erase procedure is complete.

```
#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 SS$_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$K_MEMORY, count++, &pattern);
    while (status == SS$_NORMAL)
    {
        for (loop = 0; loop < items; loop++)
            ptr[loop] = pattern;
        status = sys$erapat(ERA$K_MEMORY, count++, &pattern);
    }

    if (status == SS$_NOTRAN)   /* Check for expected status */
        status = SS$_NORMAL;    /* Change to SS$_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, $CHANGE_ACL, $CHECK_ACCESS, $CHKPRO, $CREATE_RDB, $FIND_HELD, $FIND_HOLDER, $FINISH_RDB, $FORMAT_ACL, $FORMAT_AUDIT, $GRANTID, $HASH_PASSWORD, $IDTOASC, $MOD_HOLDER, $MOD_IDENT, $MTACCESS, $PARSE_ACL, $REM_HOLDER, $REM_IDENT, $REVOKID

Condition Values Returned

SS$_NORMAL	The service completed successfully; proceed with the next erase step.
SS$_NOTRAN	The service completed successfully; security erase completed.
SS$_ACCVIO	The **patadr** argument cannot be written by the caller.
SS$_BADPARAM	The **type** argument or **count** argument is invalid.

$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]

Argument

code

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

Longword value to be saved in the process header as the completion status of the current image. If you do not specify this argument in a macro call, a value of 1 is passed as the completion code for VAX MACRO and VAX BLISS–32, and a value of 0 is passed for other languages. You can test this value at the command level to provide conditional command execution.

Description

The $EXIT service is unlike all other system services in that it does not return status codes in R0 or anywhere else. The $EXIT service does not return control to the caller; it performs an exit to the command interpreter or causes the process to terminate if no command interpreter is present.

Required Access or Privileges

None

Required Quota

None

Related Services

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

$EXPREG
Expand Program/Control Region

Adds a specified number of new virtual pages to a process's program region or control region for the execution of the current image. Expansion occurs at the current end of that region's virtual address space.

Format

SYS$EXPREG pagcnt ,[retadr] ,[acmode] ,[region]

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.

Alpha

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.

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

Number specifying which program region is to be expanded. The **region** argument is a longword value. A value of 0 (the default) specifies that the program region (P0 region) is to be expanded. A value of 1 specifies that the control region (P1 region) is to be expanded.

Description

The Expand Program/Control Region service adds a specified number of new virtual pages to a process's program region or control region for the execution of the current image. Expansion occurs at the current end of that region's virtual address space.

The new pages, which were previously inaccessible to the process, are created as demand-zero pages.

Because the bottom of the user stack is normally located at the end of the control region, expanding the control region is equivalent to expanding the user stack. The effect is to increase the available stack space by the specified amount.

The starting address returned is always the first available page in the designated region; therefore, the ending address is smaller than the starting address when the control region is expanded and is larger than the starting address when the program region is expanded.

If an error occurs while pages are being added, the **retadr** argument (if specified) indicates the pages that were successfully added before the error occurred. If no pages were added, both longwords of the **retadr** argument contain the value –1.

Required Access or Privileges

None

Required Quota

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

Related Services

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

Typically, the information returned in the location addressed by the **retadr** argument (if specified) can be used as the input range to the Delete Virtual Address Space ($DELTVA) service.

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The return address array cannot be written by the caller.
SS$_EXQUOTA	The process exceeded its paging file quota.
SS$_ILLPAGCNT	The specified page count was less than 1.

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

Arguments

region_id_64

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

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

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

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

length_64

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

Length of the virtual address space to be created. The length specified must be a multiple of CPU-specific pages.

acmode

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

Access mode associated with the call to $EXPREG_64. The access mode determines the owner mode of the pages as well as the read and write protection on the pages. The **acmode** argument is a longword containing the access mode. The $PSLDEF macro defines symbols for the four access modes.

The $EXPREG_64 service uses whichever of the following two access modes is least privileged:

- The access mode specified by the **acmode** argument
- The access mode of the caller. The protection of the pages is read/write for the resultant access mode and those more privileged.

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

flags

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

Flag mask controlling the characteristics of the demand-zero pages created. The **flags** argument is a longword bit vector in which each bit corresponds to a flag. The $VADEF macro and the VADEF.H file define a symbolic name for each flag. You construct the **flags** argument by performing a logical OR operation on the symbol names for all desired flags.

All bits in the **flags** argument are reserved for future use by Digital and should be specified as 0. The condition value SS$_IVVAFLG is returned if any bits are set.

return_va_64

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

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

return_length_64

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

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

Description

The Expand Virtual Address Space service is a kernel mode service that can be called from any mode. This service adds a range of demand-zero allocation pages to a process's virtual address space for the execution of the current image. Expansion occurs at the next free available address within the specified region. The new pages, which were previously inaccessible to the process, are created as demand-zero pages. The returned address is always the lowest virtual address in the range of pages created. The returned length is always an unsigned byte count indicating the length of the range of pages created.

Successful return status from $EXPREG_64 Expand Virtual Address service means that the specified region's virtual address space was expanded by the number of bytes specified in the **length_64** argument.

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

Required Privileges

None

Required Quota

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

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

Related Services

$CREATE_BUFOBJ_64, $CREATE_REGION_64, $CRETVA_64, $DELETE_REGION_64, $DELTVA_64, $LCKPAG_64, $LKWSET_64, $PURGE_WS, $SETPRT_64, $ULKPAG_64, $ULWSET_64

Condition Values Returned

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

$FAO/$FAOL
Formatted ASCII Output Services

The Formatted ASCII Output service (1) converts a binary value into an ASCII character string in decimal, hexadecimal, or octal notation and returns the character string in an output string, and (2) inserts variable character string data into an output string.

The Formatted ASCII Output with List Parameter service provides an alternate method for specifying input parameters when calling the $FAO system service.

The formats for both services are shown in the Format section.

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

Format

SYS$FAO ctrstr ,[outlen] ,outbuf ,[p1]...[pn]

SYS$FAOL ctrstr ,[outlen] ,outbuf ,[prmlst]

Arguments

ctrstr

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

Control string passed to $FAO that contains the text to be output together with one or more $FAO directives. $FAO directives are used to specify repeat counts or the output field length, or both, and they are preceded by an exclamation point (!). The **ctrstr** argument is the 32-bit 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 $FAO directives is described in the Description section.

There is no restriction on the length of the control string or on the number of $FAO directives it can contain. However, if an exclamation point must appear in the output string, it must be represented in the control string by a double exclamation point (!!). A single exclamation point in the control string indicates to $FAO that the next characters are to be interpreted as FAO directives.

When $FAO processes the control string, it writes to the output buffer each character that is not part of an $FAO directive.

If the $FAO directive is valid, $FAO processes it. If the directive requires a parameter, $FAO processes the next consecutive parameter in the specified parameter list. If the $FAO directive is not valid, $FAO terminates and returns a condition value in R0.

Table SYS1–7 lists and describes the $FAO directives. Table SYS1–8 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 **p20**. Each argument specifies one parameter. Because $FAO accepts a maximum of 20 parameters in a single call, you must use $FAOL if the number of parameters exceeds 20. 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-bit or 64-bit reference (Alpha) by 32-bit reference (VAX)

Length in bytes of the fully formatted output string returned by $FAO. The **outlen** argument is the 32-bit 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-bit or 64-bit descriptor–fixed-length string descriptor (Alpha) 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-bit 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 **p20**. If an $FAO directive does not require a parameter, that $FAO directive is processed without reading a parameter from the argument list.

Depending on the directive, a parameter can be a value to be converted, a 32-bit 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-bit or 64-bit reference (Alpha) by 32-bit reference (VAX)

List of $FAO directive parameters to be passed to $FAOL. The **prmlst** argument is the 32-bit 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 service (1) converts a binary value into an ASCII character string in decimal, hexadecimal, or octal notation and returns the character string in an output string, and (2) inserts variable character string data into an output string.

The Formatted ASCII Output with List Parameter ($FAOL) service provides an alternate way to specify input parameters for a call to the $FAO system service. The formats for both $FAO and $FAOL are shown in the Format section.

The $FAO_S macro form uses a PUSHL instruction for all parameters (**p1** through **p20**) 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 20 parameters on the $FAO macro. If more than 20 parameters are required, use the $FAOL macro.

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

$FAO Directives $FAO directives can appear anywhere in the control string. The general format of an $FAO directive is as follows:

```
!DD
```

The exclamation point (!) specifies that the following characters are to be interpreted as an $FAO directive, and the characters *DD* represent a 1- or 2-character $FAO directive.

Note

When the characters of the $FAO directive are alphabetic, they must be uppercase.

An $FAO directive can optionally specify the following:

- A repeat count. The format is as follows:

  ```
  !n(DD)
  ```

  In this case *n* is a decimal value specifying the number of times that $FAO is to repeat the directive. If the directive requires a parameter or parameters, $FAO uses successive parameters from the parameter list for each repetition of the directive; it does not use the same parameters for each repetition. The parentheses are required syntax.

- An output field length. The format is as follows:

  !mDD

  In this case *m* is a decimal value specifying the length of the field (within the output string) into which $FAO is to write the output resulting from the directive. The length is expressed as a number of characters.

- Both a repeat count and output field length. In this case the format is as follows:

  !n(mDD)

You can specify repeat counts and output field lengths as variables by using a number sign (#) in place of an absolute numeric value.

- If you specify a number sign for a repeat count, the next parameter passed to $FAO must contain the count.
- If you specify a number sign for an output field length, the next parameter must contain the length value.
- If you specify a number sign for both the output field length and for the repeat count, only one length parameter is required; each output string will have the specified length.
- If you specify a number sign for the repeat count, the output field length, or both, the parameters specifying the count, length, or both must precede other parameters required by the directive.

Numeric FAO output directives (B, W, L, Q, I, A, H, J) can include the indirect directive @. This immediately precedes the directive (@DD), and indicates that the next parameter is the address of the value instead of the value itself. This directive must be used with any directive that can produce a quadword output when using $FAOL; otherwise, $FAOL creates a 64-bit sign-extended value. This includes the Q, A, I, H, and J directives.

- The indirect directive can be used with repeat counts and output field lengths. In this case the format is as follows:

  !n(m@DD)

To ensure that addresses and integers are displayed properly on the system, use the following conventions when using the $FAO and $FAOL system services:

- Identify longword data as !*x*L (where *x* is O, X, Z, U, or S).
- Identify quadword data as !*x*Q for $FAO and $FAOL_64 or !@*x*Q for $FAOL (where *x* is O, X, Z, U, or S). Omitting the indirect directive for $FAOL can result in a 64-bit sign-extended value being created.
- If the size of an address is determined by operating system software (32 bits on VAX and 64-bits on Alpha systems), identify the address as !*x*A for $FAO and $FAOL_64 or !@*x*A for $FAOL (where *x* is O, X, Z, U, or S).
- If the size of an address is determined by the hardware architecture (32 bits on VAX, but 64 bits on Alpha, identify the address as !*x*H for $FAO and $FAOL_64 or !@*x*H for $FAOL (where *x* is O, X, Z, U, or S). Omitting the indirect directive for $FAOL can result in a 64-bit sign-extended value being created.

- If the size of an integer is determined by operating system software (32 bits on both VAX and Alpha systems), identify the integer as !*x*I for $FAO and $FAOL_64 or !@*x*I for $FAOL (where *x* is O, X, Z, U, or S).
- If the size of an integer is determined by the hardware architecture (32 bits on VAX, but 64 bits on Alpha), identify the address as !*x*J for $FAO and $FAOL_64 or !@*x*J for $FAOL (where *x* is O, X, Z, U, or S). Omitting the indirect directive for $FAOL can result in a 64-bit sign-extended value being created.

Table SYS1–7 lists $FAO directives.

Table SYS1–7 $FAO Directives

Directive	Description
Directives for Character String Substitution	
!AC	Inserts a counted ASCII string. It requires one parameter: the address of the string to be inserted. The first byte of the string must contain the length (in characters) of the string.
!AD	Inserts an ASCII string. It requires two parameters: the length of the string and the address of the string. Each of these parameters is a separate argument.
!AF	Inserts an ASCII string and replaces all nonprintable ASCII codes with periods (.). It requires two parameters: the length of the string and the address of the string. Each of these parameters is a separate argument.
!AS	Inserts an ASCID string. It requires one parameter: the address of a character string descriptor pointing to the string. $FAO assumes that the descriptor is a CLASS_S (static) string descriptor. Other descriptor types might give incorrect results.
!AZ	Inserts a zero-terminated (ASCIZ) string. It requires one parameter: the address of a zero-terminated string.
Directives for Zero-Filled Numeric Conversion	
!OB	Converts a byte value to the ASCII representation of the value's octal equivalent. It requires one parameter: the value to be converted. $FAO uses only the low-order byte of the longword parameter.
!OW	Converts a word value to the ASCII representation of the value's octal equivalent. It requires one parameter: the value to be converted. $FAO uses only the low-order word of the longword parameter.
!OL	Converts a longword value to the ASCII representation of the value's octal equivalent. It requires one parameter: the value to be converted.

(continued on next page)

Table SYS1–7 (Cont.) $FAO Directives

Directive	Description
Directives for Zero-Filled Numeric Conversion	
!OQ	Converts 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.
!OA	Converts an address to the ASCII representation of its octal equivalent. Must use the indirect directive @ to output the quadword value for $FAOL; otherwise, a 32-bit sign-extended value is written to the output buffer. It receives one parameter: the address of the value to be converted.[1]
!OI	Converts an integer to the ASCII representation of its octal equivalent. Must use the indirect directive @ to output the quadword value for $FAOL; otherwise, a 32-bit sign-extended value is written to the output buffer. It receives one parameter: the address of the value to be converted.[1]
!OH	Converts an address to the ASCII representation of its octal equivalent. Must use the indirect directive @ to output the quadword value for $FAOL; otherwise, a 64-bit sign-extended value is written to the output buffer. It receives one parameter: the address of the value to be converted.[2]
!OJ	Converts an integer to the ASCII representation of its octal equivalent. Must use the indirect directive @ to output the quadword value for $FAOL; otherwise, a 64-bit sign-extended value is written to the output buffer. It receives one parameter: the address of the value to be converted.[2]
!XB	Converts a byte value to the ASCII representation of the value's hexadecimal equivalent. It requires one parameter: the value to be converted. $FAO uses only the low-order byte of the longword parameter.
!XW	Converts a word value to the ASCII representation of the value's hexadecimal equivalent. It requires one parameter: the value to be converted. $FAO uses only the low-order word of the longword parameter.
!XL	Converts a longword value to the ASCII representation of the value's hexadecimal equivalent. It requires one parameter: the value to be converted.
!XQ	Converts a quadword to the ASCII representation of its hexadecimal equivalent. Must use the indirect directive @ to output the quadword value for $FAOL; otherwise, a 64-bit sign-extended value is written to the output buffer.

[1]Determined by the operating system. On VAX and Alpha systems, this is 32 bits.

[2]Determined by the hardware architecture. On VAX systems, this is 32 bits; on Alpha systems, this is 64 bits.

(continued on next page)

Table SYS1–7 (Cont.) $FAO Directives

Directive	Description
Directives for Zero-Filled Numeric Conversion	
!XA	Converts an address to the ASCII representation of its hexadecimal equivalent. Must use the indirect directive @ to output the quadword value for $FAOL; otherwise, a 32-bit sign-extended value is written to the output buffer. It receives one parameter: the address of the value to be converted.[1]
!XI	Converts an integer to the ASCII representation of its hexadecimal equivalent. Must use the indirect directive @ to output the quadword value for $FAOL; otherwise, a 32-bit sign-extended value is written to the output buffer. It receives one parameter: the address of the value to be converted.[1]
!XH	Converts an address to the ASCII representation of its hexadecimal equivalent. Must use the indirect directive @ to output the quadword value for $FAOL; otherwise, a 64-bit sign-extended value is written to the output buffer. It receives one parameter: the address of the value to be converted.[2]
!XJ	Converts an integer to the ASCII representation of its hexadecimal equivalent. Must use the indirect directive @ to output the quadword value for $FAOL; otherwise, a 64-bit sign-extended value is written to the output buffer. It receives one parameter: the address of the value to be converted.[2]
!ZB	Converts an unsigned byte value to the ASCII representation of the value's decimal equivalent. It requires one parameter: the value to be converted. $FAO uses only the low-order byte of the longword parameter.
!ZW	Converts an unsigned word value to the ASCII representation of the value's decimal equivalent. It requires one parameter: the value to be converted. $FAO uses only the low-order word of the longword parameter.
!ZL	Converts an unsigned longword value to the ASCII representation of the value's decimal equivalent. It requires one parameter: the value to be converted.
!ZQ	Converts an unsigned quadword to the ASCII representation of its decimal equivalent. Must use the indirect directive @ to output the quadword value for $FAOL; otherwise, a 64-bit zero-extended value is written to the output buffer. It receives one parameter: the address of the value to be converted.
!ZA	Converts an unsigned address to the ASCII representation of its decimal equivalent. Must use the indirect directive @ to output the quadword value for $FAOL; otherwise, a 32-bit value is written to the output buffer. It receives one parameter: the address of the value to be converted.[1]

[1]Determined by the operating system. On VAX and Alpha systems, this is 32 bits.

[2]Determined by the hardware architecture. On VAX systems, this is 32 bits; on Alpha systems, this is 64 bits.

(continued on next page)

Table SYS1–7 (Cont.) $FAO Directives

Directive	Description
Directives for Zero-Filled Numeric Conversion	
!ZI	Converts an unsigned integer to the ASCII representation its decimal equivalent. Must use the indirect directive @ to output the quadword value for $FAOL; otherwise, a 32-bit value is written to the output buffer. It receives one parameter: the address of the value to be converted.[1]
!ZH	Converts an unsigned address to the ASCII representation of its decimal equivalent. Must use the indirect directive @ to output the quadword value for $FAOL; otherwise, a 64-bit zero-extended value is written to the output buffer. It receives one parameter: the address of the value to be converted.[2]
!ZJ	Converts an unsigned integer to the ASCII representation of its decimal equivalent. Must use the indirect directive @ to output the quadword value for $FAOL; otherwise, a 64-bit zero-extended value is written to the output buffer. It receives one parameter: the address of the value to be converted.[2]
Directives for Blank-Filled Numeric Conversion	
!UB	Converts an unsigned byte value to the ASCII representation of the value's decimal equivalent. It requires one parameter: the value to be converted. $FAO uses only the low-order byte of the longword parameter.
!UW	Converts an unsigned word value to the ASCII representation of the value's decimal equivalent. It requires one parameter: the value to be converted. $FAO uses only the low-order word of the longword parameter.
!UL	Converts an unsigned longword value to the ASCII representation of the value's decimal equivalent. It requires one parameter: the value to be converted.
!UQ	Converts an unsigned quadword to the ASCII representation of its decimal equivalent. Must use the indirect directive @ to output the quadword value for $FAOL; otherwise, a 64-bit value is written to the output buffer. It receives one parameter: the address of the value to be converted.[1]
!UA	Converts an unsigned address to the ASCII representation of its decimal equivalent. Must use the indirect directive @ to output the quadword value for $FAOL; otherwise, a 32-bit value is written to the output buffer. It receives one parameter: the address of the value to be converted.[1]

[1]Determined by the operating system. On VAX and Alpha systems, this is 32 bits.

[2]Determined by the hardware architecture. On VAX systems, this is 32 bits; on Alpha systems, this is 64 bits.

(continued on next page)

Table SYS1–7 (Cont.) $FAO Directives

Directive	Description
Directives for Blank-Filled Numeric Conversion	
!UI	Converts an unsigned integer to the ASCII representation of its decimal equivalent. Must use the indirect directive @ to output the quadword value for $FAOL; otherwise, a 32-bit value is written to the output buffer.It receives one parameter: the address of the value to be converted.[1]
!UH	Converts an unsigned address to the ASCII representation of its decimal equivalent. Must use the indirect directive @ to output the quadword value for $FAOL; otherwise, a 64-bit value is written to the output buffer. It receives one parameter: the address of the value to be converted.[2]
!UJ	Converts an unsigned integer to the ASCII representation of its decimal equivalent. Must use the indirect directive @ to output the quadword value for $FAOL; otherwise, a 64-bit value is written to the output buffer. It receives one parameter: the address of the value to be converted.[2]
!SB	Converts a signed byte value to the ASCII representation of the value's decimal equivalent. It requires one parameter: the value to be converted. $FAO uses only the low-order byte of the longword parameter.
!SW	Converts a signed word value to the ASCII representation of the value's decimal equivalent. It requires one parameter: the value to be converted. $FAO uses only the low-order word of the longword parameter.
!SL	Converts a signed longword value to the ASCII representation of the value's decimal equivalent. It requires one parameter: the value to be converted.
!SQ	Converts 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.
!SA	Converts a signed address to the ASCII representation of its decimal equivalent. Must use the indirect directive @ to output the quadword value for $FAOL; otherwise, a 32-bit value is written to the output buffer. It receives one parameter: the address of the value to be converted.[1]
!SI	Converts a signed integer to the ASCII representation of its equivalent.Must use the indirect directive @ to output the quadword value for $FAOL; otherwise, a 32-bit value is written to the output buffer. It receives one parameter: the address of the value to be converted.[1]

[1]Determined by the operating system. On VAX and Alpha systems, this is 32 bits.

[2]Determined by the hardware architecture. On VAX systems, this is 32 bits; on Alpha systems, this is 64 bits.

(continued on next page)

Table SYS1–7 (Cont.) $FAO Directives

Directive	Description
Directives for Blank-Filled Numeric Conversion	
!SH	Converts a signed address to the ASCII representation of its decimal equivalent. Must use the indirect directive @ to output the quadword value for $FAOL; otherwise, a 32-bit value is written to the output buffer. It receives one parameter: the address of the value to be converted.[2]
!SJ	Converts a signed integer to the ASCII representation of its decimal equivalent. Must use the indirect directive @ to output the quadword value for $FAOL; otherwise, a 32-bit value is written to the output buffer. It receives one parameter: the address of the value to be converted.[2]
Directives for Output String Formatting	
!/	Inserts a new line, that is, a carriage return and line feed. It takes no parameters.
!_	Inserts a tab. It takes no parameters.
!^	Inserts a form feed. It takes no parameters.
!!	Inserts an exclamation point. It takes no parameters.
!%S	Inserts the letter *S* if the most recently converted numeric value is not 1. An uppercase *S* is inserted if the character before the !%S directive is an uppercase character; a lowercase *s* is inserted if the character is lowercase.
!%T	Inserts the system time. It takes one parameter: the address of a quadword time value to be converted to ASCII. If you specify 0, the current system time is inserted.
!%U	Converts a longword integer UIC to a standard UIC specification in the format [xxx,yyy], where *xxx* is the group number and *yyy* is the member number. It takes one parameter: a longword integer. The directive inserts the surrounding brackets ([ ]) and comma (,).
!%I	Converts a longword to the appropriate alphanumeric identifier. If the longword represents a UIC, surrounding brackets ([ ]) and comma (,) are added as necessary. If no identifier exists and the longword represents a UIC, the longword is formatted as in !%U. Otherwise it is formatted as in !XL with a preceding !%X added to the formatted result.
!%D	Inserts the system date and time. It takes one parameter: the address of a quadword time value to be converted to ASCII. If you specify 0, the current system date and time is inserted.
!n%C	Inserts a character string when the most recently evaluated argument has the value *n*. (Recommended for use with multilingual products.)

[2]Determined by the hardware architecture. On VAX systems, this is 32 bits; on Alpha systems, this is 64 bits.

(continued on next page)

Table SYS1–7 (Cont.) $FAO Directives

Directive	Description
Directives for Output String Formatting	
!%E	Inserts a character string when the value of the most recently evaluated argument does not match any preceding !n%C directives. (Recommended for use with multilingual products.)
!%F	Makes the end of a plurals statement.
!n<	See description of next directive (!>).
!>	This directive and the preceding one (!n<) are used together to define an output field width of *n* characters within which all data and directives to the right of !n< and to the left of !> are left-justified and blank-filled. It takes no parameters.
!n*c	Repeats the character *c* in the output string *n* times.
Directives for Parameter Interpretation	
!–	Causes $FAO to reuse the most recently used parameter in the list. It takes no parameters.
!+	Causes $FAO to skip the next parameter in the list. It takes no parameters.

Table SYS1–8 shows the $FAO output field lengths and their fill characters.

Table SYS1–8 $FAO Output Field Lengths and Fill Characters

Conversion/Substitution Type	Default Length of Output Field	Action When Explicit Output Field Length Is Longer Than Default	Action When Explicit Output Field Length Is Shorter Than Default
Hexadecimal Byte Word Longword Quadword	 2 (zero-filled) 4 (zero-filled) 8 (zero-filled) 16 (zero-filled)	ASCII result is right-justified and blank-filled to the specified length.	ASCII result is truncated on the left.
Octal Byte Word Longword Quadword	 3 (zero-filled) 6 (zero-filled) 11 (zero-filled) 22 (zero-filled)	Hexadecimal or octal output is always zero-filled to the default output field length, then blank-filled to specified length.	
Signed or unsigned decimal	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 (*).

(continued on next page)

Table SYS1–8 (Cont.) $FAO Output Field Lengths and Fill Characters

Conversion/Substitution Type	Default Length of Output Field	Action When Explicit Output Field Length Is Longer Than Default	Action When Explicit Output Field Length Is Shorter Than Default
Unsigned zero-filled decimal	As many characters as necessary	ASCII result is right-justified and zero-filled to the specified length.	
ASCII string substitution	Length of input character string	ASCII string is left-justified and blank-filled to the specified length.	ASCII string is truncated on the right.

Required Access or Privileges

None

Required Quota

None

Related Services

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

Condition Values Returned

SS$_BUFFEROVF	The service completed successfully. The formatted output string overflowed the output buffer and has been truncated.
SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The **ctrstr**, **p1** through **pn**, or **prmlst** arguments cannot be read, or the **outlen** argument cannot be written (it can specify 0).
SS$_BADPARAM	You specified an invalid directive in the $FAO control string.
SS$_OVERMAXARG	Maximum parameter count exceeded.

$FAO Control String Examples

Each of the following examples shows an $FAO control string with several directives, parameters defined as input for the directives, and the calls to $FAO to format the output strings.

Each example is accompanied by notes. These notes show the output string created by the call to $FAO and describe in more detail some considerations for using directives. The sample output strings show the underscore character (_) for each space in all places where $FAO output contains multiple spaces.

Each of the first 10 examples (numbered 1 through 10) refers to the following output fields but does not include these fields within the examples.

```
int     status,                         /* Status of system calls */
        outlen;                         /* Length of output string from $FAO */
char    out_buffer[80];                 /* Buffer for $FAO output */

$DESCRIPTOR(out_desc, out_buffer);      /* Descriptor for out_buffer */
```

Each of the 10 examples also assumes the caller of each example will check the returned status, and write the output string produced by $FAO if no error occurred. The following code fragment shows how the example call may be made, and the resultant string output to the user's terminal.

```
#include <stdio.h>
#include <stsdef.h>
#include <lib$routines.h>
        .
        .
        .
    status = example();

    /* Immediately signal (and quit) if error occurred */
    if ((status & STS$M_SUCCESS) == 0)  lib$signal(status);

    /* FAO directive succeeded, output resultant string */
    out_buffer[outlen] = '\0';          /* add string terminator to buffer */
    puts(out_buffer);                   /* output the result */
```

The final example (numbered 11) shows a segment of a DEC Fortran for OpenVMS program used to output an ASCII string.

1.
```
/* SYS$FAO example - illustrating !AC, !AS, !AD, and !/ 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}

int example()
{
    char    *nod = "Nod";               /* Normal "C" string */
    const int nodlen = sizeof(nod) - 1; /* Length of "Nod" without '\0' */
    static  ASCIC_STRING(winken, "Winken");
    static  $DESCRIPTOR(blinken, "Blinken");
    static  $DESCRIPTOR(fao_desc, "!/Sailors: !AC !AS !AD");

    return (sys$fao(&fao_desc,  /* Control string for $FAO */
                    &outlen,    /* Pointer for length of output string */
                    &out_desc,  /* Descriptor for output buffer */
                    &winken,    /* P1 - Counted ASCII string */
                    &blinken,   /* P2 - ASCII string descriptor */
                    nodlen,     /* P3 - Length of ASCII string */
                    nod));      /* P4 - ASCII string */
}
```

$FAO writes the following string into the output buffer:

```
<CR><KEY>(LF\TEXT)Sailors: Winken Blinken Nod
```

The !/ directive provides a carriage-return/line-feed character (shown as <CR><KEY>(LF\TEXT)) for terminal output.

The !AC directive requires the address of a counted ASCII string (**p1** argument).

The !AS directive requires the address of a character string descriptor (**p2** argument).

The !AD directive requires two parameters: the length of the string to be substituted (**p3** argument) and its address (**p4** argument).

2.
```
/*
** SYS$FAO example - illustrating !! and !AS directives,
** repeat count, and output field length
*/
#include <descrip.h>
#include <starlet.h>

int example()
{
    static $DESCRIPTOR(jones, "Jones");
    static $DESCRIPTOR(harris, "Harris");
    static $DESCRIPTOR(wilson, "Wilson");
    static $DESCRIPTOR(fao_desc, "Unable to locate !3(8AS)!!");

    return(sys$fao(&fao_desc,   /* Control string for $FAO */
                   &outlen,     /* Pointer for length of output string */
                   &out_desc,   /* Descriptor for output buffer */
                   &jones,      /* P1 - ASCII string descriptor */
                   &harris,     /* P2 - ASCII string descriptor */
                   &wilson));   /* P3 - ASCII string descriptor */
}
```

$FAO writes the following string into the output buffer:

```
Unable to locate Jones___Harris__Wilson__!
```

The !3(8AS) directive contains a repeat count: three parameters (addresses of character string descriptors) are required. $FAO left-justifies each string into a field of eight characters (the output field length specified).

The double exclamation point directive (!!) supplies a literal exclamation point (!) in the output.

If the directive were specified without an output field length, that is, if the directive were specified as !3(AS), the three output fields would be concatenated, as follows:

```
Unable to locate JonesHarrisWilson!
```

3.
```
/* SYS$FAO example - illustrating !UL, !XL, and !SL directives */
#include <descrip.h>
#include <starlet.h>

int example()
{
    int val1 = 200,             /* Values */
        val2 = 300,             /*  for   */
        val3 = -400;            /*   $FAO */

    static $DESCRIPTOR(fao_desc,
                       "Values !UL (Decimal) !XL (Hex) !SL (Signed)");

    return(sys$fao(&fao_desc,   /* Control string for $FAO */
                   &outlen,     /* Pointer for length of output string */
                   &out_desc,   /* Descriptor for output buffer */
                   val1,        /* P1 - longword value */
                   val2,        /* P2 - longword value */
                   val3));      /* P3 - longword value */
}
```

$FAO writes the following string to the output buffer:

```
Values 200 (Decimal) 0000012C (Hex) -400 (Signed)
```

The longword value 200 is converted to decimal, the value 300 is converted to hexadecimal, and the value –400 is converted to signed decimal. The ASCII results of each conversion are placed in the appropriate position in the output string.

Note that the hexadecimal output string has eight characters and is zero-filled to the left. This is the default output length for hexadecimal longwords.

4.
```
/* SYS$FAOL example - illustrating !UL, !XL, and !SL directives */
#include <descrip.h>
#include <starlet.h>

int example()
{
    static int values[3] = {200, 300, -400};   /* Parameters for $FAOL */
    static $DESCRIPTOR(fao_desc,
                       "Values !UL (Decimal) !XL (Hex) !SL (Signed)");

    return(sys$faol(&fao_desc,  /* Control string for $FAO */
                    &outlen,    /* Pointer for length of output string */
                    &out_desc,  /* Descriptor for output buffer */
                    values));   /* Parameter list - longwords */
}
```

$FAOL writes the following string to the output buffer:

```
Values 200 (Decimal) 0000012C (Hex) -400 (Signed)
```

The results are the same as the results of Example 3. However, unlike the $FAO directive, which requires each parameter on the call to be specified, the $FAOL directive points to a list of consecutive longwords, which $FAO reads as parameters.

5.
```
/* SYS$FAOL example - illustrating !UB, !XB, and !SB directives */
#include <descrip.h>
#include <starlet.h>

int example()
{
    static int values[3] = {200, 300, -400};   /* Parameters for $FAOL */
    static $DESCRIPTOR(fao_desc,
                       "Values !UB (Decimal) !XB (Hex) !SB (Signed)");

    return(sys$faol(&fao_desc,  /* Control string for $FAO */
                    &outlen,    /* Pointer for length of output string */
                    &out_desc,  /* Descriptor for output buffer */
                    values));   /* Parameter list - longwords */
}
```

$FAO writes the following output string:

```
Values 200 (Decimal) 2C (Hex) 112 (Signed)
```

The input parameters are the same as those for Example 4. However, the control string (fao_desc) specifies that byte values are to be converted. $FAO uses the low-order byte of each longword parameter passed to it. The high-order three bytes are not evaluated. Compare these results with the results of Example 4.

6.
```
/*
** SYS$FAO example - illustrating !XW, !ZW, and !- directives,
** repeat count, and output field length
*/
#include <descrip.h>
#include <starlet.h>

int example()
{
    static $DESCRIPTOR(fao_desc,
                       "Hex: !2(6XW) Zero-filled Decimal: !2(-)!2(7ZW)");

    return(sys$fao(&fao_desc,   /* Control string for $FAO */
                   &outlen,     /* Pointer for length of output string */
                   &out_desc,   /* Descriptor for output buffer */
                   10000,       /* P1 - longword value */
                   9999));      /* P2 - longword value */
}
```

$FAO writes the following string to the output buffer:

```
Hex:___2710__270F Zero-filled Decimal: 00100000009999
```

Each of the directives !2(6XW) and !2(7ZW) contains repeat counts and output lengths. First, $FAO performs the !XW directive twice, using the low-order word of the numeric parameters passed. The output length specified is two characters longer than the default output field width of hexadecimal word conversion, so two spaces are placed between the resulting ASCII strings.

The !– directive causes $FAO to back up over the parameter list. A repeat count is specified with the directive so that $FAO skips back over two parameters; then, it uses the same two parameters for the !ZW directive. The !ZW directive causes the output string to be zero-filled to the specified length (in this example, seven characters). Thus, there are no spaces between the output fields.

7.
```
/*
** SYS$FAOL example - illustrating !AS, !UB, !%S, and !- directives,
** and variable repeat count
*/
#include <descrip.h>
#include <starlet.h>

/* Layout of argument list for examples */
typedef struct {void    *desc;          /* ASCII string descriptor */
                int     arg[4];         /* Longword arguments */
                } LIST;

$DESCRIPTOR(fao_desc, "!AS received !UB argument!%S: !-!#(4UB)");

int example_a()
{
    static $DESCRIPTOR(orion, "ORION");
    static LIST
        list_a = {&orion,       /* Address of descriptor */
                  3,            /* Number of arguments */
                  10,           /* Argument 1 */
                  123,          /* Argument 2 */
                  210};         /* Argument 3 */

    return(sys$faol(&fao_desc,  /* Control string for $FAO */
                    &outlen,    /* Pointer for length of output string */
                    &out_desc,  /* Descriptor for output buffer */
                    &list_a));  /* Parameter list */
}
```

```
int example_b()
{
    static $DESCRIPTOR(lyra, "LYRA");
    static LIST
        list_b = {&lyra,        /* ASCII descriptor cast as an (int) */
                  1,            /* Number of arguments */
                  255};         /* Argument 1 */

    return(sys$faol(&fao_desc,  /* Control string for $FAO */
                    &outlen,    /* Pointer for length of output string */
                    &out_desc,  /* Descriptor for output buffer */
                    &list_b));  /* Parameter list */
}
```

In example A, $FAO writes the following string to the output buffer:

```
ORION received 3 arguments:___10 123 210
```

In example B, $FAO writes the following string to the output buffer:

```
LYRA received 1 argument:__255
```

In each of the examples, the parameter list argument points to a different parameter list; each list contains, in the first longword, the address of a character string descriptor. The second longword begins an argument list, with the number of arguments remaining in the list. The control string uses this second longword twice: first to output the value contained in the longword, and then to provide the repeat count to output the number of arguments in the list (the !- directive indicates that $FAO should reuse the parameter).

The !%S directive provides a conditional plural. When the last value converted has a value not equal to 1, $FAO outputs the character *s*; if the value is a 1 (as in Example B), $FAO does not output the character *s*. $FAO outputs the plural character in lowercase since the preceding character was in lowercase.

The output field length defines a width of four characters for each byte value converted, to provide spacing between the output fields.

8.
```
/*
** SYS$FAO example - illustrating !n*c (repeat character)
** and !%D (date/time) directives
*/
#include <descrip.h>
#include <starlet.h>

int example()
{
    static $DESCRIPTOR(fao_desc, "!5*> The time is now: !%D");

    return(sys$fao(&fao_desc,   /* Control string for $FAO */
                   &outlen,     /* Pointer for length of output string */
                   &out_desc,   /* Descriptor for output buffer */
                   0));         /* P1 - time value, 0 = current time */
}
```

$FAO writes the following string to the output buffer:

```
>>>>> The time is now: dd-mmm-yyyy hh:mm:ss.cc
```

where:

dd	is the day of the month
mmm	is the month
yyyy	is the year
hh:mm:ss.cc	is the time in hours, minutes, seconds, and hundredths of a second

The !5*> directive requests $FAO to write five greater-than (>) characters into the output string. Because there is a space after the directive, $FAO also writes a space after the greater-than characters on output.

The !%D directive requires the address of a quadword time value, which must be in the system time format. However, when the address of the time value is specified as 0, $FAO uses the current date and time. For a detailed description of the ASCII date and time string returned, see the discussion of the Convert Binary Time to ASCII String ($ASCTIM) system service.

9.
```
/*
** SYS$FAO example - illustrating !%D and !%T (with output field lengths),
** and !n directive with variable repeat count
*/
#include <descrip.h>
#include <starlet.h>

int example()
{
    static $DESCRIPTOR(fao_desc, "Date: !11%D!#*_Time: !5%T");

    return(sys$fao(&fao_desc,   /* Control string for $FAO */
                   &outlen,     /* Pointer for length of output string */
                   &out_desc,   /* Descriptor for output buffer */
                   0,           /* P1 - time value, 0 = current time */
                   5,           /* P2 - Number of underscores */
                   0));         /* P3 - time value, 0 = current time */
}
```

$FAO writes the following string to the output buffer:

```
Date: dd-mmm-yyyy____Time: hh:mm
```

An output length of 11 bytes is specified with the !%D directive so that $FAO truncates the time from the date and time string, and outputs only the date.

The !#*_ directive requests that the underscore character (_) be repeated the number of times specified by the next parameter. Because **p2** is specified as 5, five underscores are written into the output string.

The !%T directive normally returns the full system time. The !5%T directive provides an output length for the time; only the hours and minutes fields of the time string are written into the output buffer.

10.
```
/*
** SYS$FAO example - illustrating !< and !> (define field width),
** !AC, and !UL directives
*/
#include <descrip.h>
#include <starlet.h>

/* MACRO and typedef for counted ASCII strings... */
typedef struct {char len, str[25];} ASCIC;
#define ASCIC_STRING(name, string) ASCIC name = {sizeof(string) - 1, string}

$DESCRIPTOR(fao_desc, "!32<Variable: !AC Value: !UL!>Total:!7UL");
```

```
int example_a()
{
    int     val_a = 334,        /* Current value for variable */
            tot_a = 6554;       /* Current total for variable */

    static ASCIC_STRING(var_a, "Inventory");    /* Counted ASCII string */

    return(sys$fao(&fao_desc,   /* Control string for $FAO */
                   &outlen,     /* Pointer for length of output string */
                   &out_desc,   /* Descriptor for output buffer */
                   &var_a,      /* P1 - Variable name */
                   val_a,       /* P2 - Value for variable */
                   tot_a));     /* P3 - Total for variable */
}

int example_b()
{
    int val_b = 280,            /* Current value for variable */
        tot_b = 10750;          /* Current total for variable */

    static ASCIC_STRING(var_b, "Sales");        /* Counted ASCII string */

    return(sys$fao(&fao_desc,   /* Control string for $FAO */
                   &outlen,     /* Pointer for length of output string */
                   &out_desc,   /* Descriptor for output buffer */
                   &var_b,      /* P1 - Variable name */
                   val_b,       /* P2 - Value for variable */
                   tot_b));     /* P3 - Total for variable */
}
```

In example A, $FAO writes the following string to the output buffer:

```
Variable: Inventory Value: 334__Total:___6554
```

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))

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 DEC Fortran for OpenVMS program used to output the following string:

```
FILE [BOELITZ]TESTING.DAT ABORTED AT ERROR 25
```

$FAOL_64 (Alpha Only)
Formatted ASCI Output with List Parameter for 64-Bit Virtual Addresses

On Alpha systems, (1) converts a binary value into an ASCII character string in decimal, hexadecimal, or octal notation and returns the character string in an output string, and (2) inserts variable character string data into an output string.

$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]]]

Arguments

ctrstr_64
OpenVMS usage: char_string
type: character-coded text string
access: read only
mechanism: by 32-bit or 64-bit descriptor–fixed-length string descriptor

The 32-bit 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-bit or 64-bit reference

The 32-bit 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-bit or 64-bit descriptor–fixed-length string descriptor

The 32-bit or 64-bit address of a character string descriptor that points to the output buffer into which $FAOL_64 writes the fully formatted output string.

quad_prmlst_64
OpenVMS usage: vectpr_quadword_unsigned
type: quadword (unsigned)
access: read only
mechanism: by 32-bit or 64-bit reference

The 32-bit or 64-bit address of a quadword-aligned array of quadword FAO arguments.

$FILESCAN
Scan String for File Specification

Searches a string for a file specification and parses the components of that file specification.

Format

SYS$FILESCAN srcstr ,valuelst ,[fldflags] ,[auxout] ,[retlen]

Arguments

srcstr

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

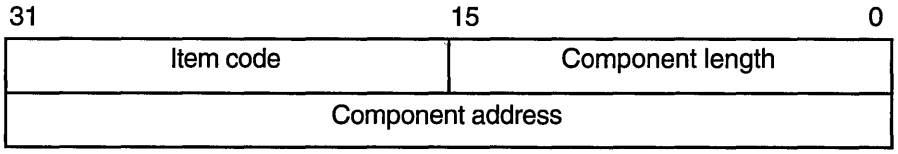
String to be searched for the file specification. The **srcstr** argument is the address of a descriptor pointing to this string.

valuelst

OpenVMS usage:	item_list_2
type:	longword (unsigned)
access:	modify
mechanism:	by reference

Item list specifying which components of the file specification are to be returned by $FILESCAN. The components are the full node specification, primary node name, primary node's access control, secondary node information, device, directory, file name, file type, and version number. The **itmlst** argument is the address of a list of item descriptors wherein each item descriptor specifies one component. The list of item descriptors is terminated by a longword of 0.

The following diagram depicts a single item descriptor.



ZK–5185A–GE

The following table defines the item descriptor fields.

Descriptor Field	Definition
Component length	A word in which $FILESCAN writes the length (in characters) of the requested component. If $FILESCAN does not locate the component, it returns the value 0 in this field and in the component address field and returns the SS$_ NORMAL condition value.
Item code	A user-supplied, word-length symbolic code that specifies the component desired. The $FSCNDEF macro defines the item codes.
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 $FILESCAN does not locate the component, it returns the value 0 in this field and in the component length field, and returns the SS$_ NORMAL condition value. If an auxiliary output buffer was provided, this address points to a location in the auxiliary output buffer, rather than to a location in the input string.

fldflags

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

Longword flag mask in which $FILESCAN sets a bit for each file specification component found in the input string. The **fldflags** argument is the address of this longword flag mask.

The $FSCNDEF macro defines a symbolic name for each significant flag bit. The following table shows the file specification component that corresponds to the symbolic name of each flag bit.

Symbolic Name	Corresponding Component
FSCN$V_DEVICE	Device name
FSCN$V_DIRECTORY	Directory name
FSCN$V_NAME	File name
FSCN$V_NODE	Node name
FSCN$V_NODE_ACS	Access control string of primary node
FSCN$V_NODE_ PRIMARY	Primary (first) node name
FSCN$V_NODE_ SECONDARY	Secondary (additional) node information

Symbolic Name	Corresponding Component
FSCN$V_ROOT	Root directory name string
FSCN$V_TYPE	File type
FSCN$V_VERSION	Version number

The **fldflags** argument is optional. When you want to know which components of a file specification are present in a string but do not need to know the contents or length of these components, specify **fldflags** instead of **valuelst**.

auxout

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

Auxiliary output buffer. The **auxout** argument is the address of a character-string descriptor pointing to the auxiliary buffer.

When you specify an auxiliary output buffer, $FILESCAN copies the entire source string, with quotation information reduced and simplified for only the primary node, into the auxiliary output buffer.

When the auxiliary output buffer is provided, all addresses returned in the item list point to locations in the auxiliary output buffer.

retlen

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

Length of the auxiliary buffer. The **retlen** argument is the address of a word into which $FILESCAN writes the length of the auxiliary buffer name string.

Item Codes

FSCN$_DEVICE
When you specify FSCN$_DEVICE, $FILESCAN returns the length and starting address of the device name. The device name includes the single colon (:).

FSCN$_DIRECTORY
When you specify FSCN$_DIRECTORY, $FILESCAN returns the length and starting address of the directory name. The directory name includes the brackets ([ ]) or angle brackets (< >).

FSCN$_FILESPEC
When you specify FSCN$_FILESPEC, $FILESCAN returns the length and starting address of the full file specification. The full file specification contains the node, device, directory, name, type, and version.

FSCN$_NAME
When you specify FSCN$_NAME, $FILESCAN returns the length and starting address of the file name. The file name includes no syntactical elements.

$FILESCAN also returns the length and starting address of a quoted file specification following a node specification (as in the specification NODE::"FILE-SPEC"). The beginning and ending quotation marks are included.

FSCN$_NODE
When you specify FSCN$_NODE, $FILESCAN returns the length and starting address of the full node specification. The full node specification includes the primary node name, the primary node's access control string, any secondary node information, and the final double colon (::).

FSCN$_NODE_ACS
When you specify FSCN$_NODE_ACS, $FILESCAN returns the length and starting address of the primary access control string. If multiple nodes are specified, the primary access control string represents the control information (if present) for the first node specified. The primary access control string does not contain the double colon (::), but does contain the double quotes.

FSCN$_NODE_PRIMARY
When you specify FSCN$_NODE_PRIMARY, $FILESCAN returns the length and starting address of the primary node name. If multiple nodes are specified, the primary node name represents the first node specification. The node name does not include the double colon (::) or any access control information. If an auxiliary output buffer is specified, quotation information is reduced and simplified for only the primary node.

FSCN$_NODE_SECONDARY
When you specify FSCN$_NODE_SECONDARY, $FILESCAN returns the length and starting address of any secondary node information. The secondary node string contains any node information referring to additional nodes, including the final double colon (::), as well as any access control strings (if present) for the additional nodes.

FSCN$_ROOT
When you specify FSCN$_ROOT, $FILESCAN returns the length and starting address of the root directory string. The root directory name string includes the brackets ([ ]) or angle brackets (< >).

FSCN$_TYPE
When you specify FSCN$_TYPE, $FILESCAN returns the length and starting address of the file type. The file type includes the preceding period (.).

FSCN$_VERSION
When you specify FSCN$_VERSION, $FILESCAN returns the length and starting address of the file version number. The file version number includes the preceding period (.) or semicolon (;) delimiter.

Description

The Scan String for File Specification service searches a string for a file specification and parses the components of that file specification. When $FILESCAN locates a partial file specification (for example, DISK:[FOO]), it returns the length and starting address of those components that were requested in the item list and were found in the string. If a component was requested in the item list but not found in the string, $FILESCAN returns a length of 0 and starting address of 0 to the component length and component address fields of the item descriptor for that component.

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

Related Services

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

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The service could not read the string pointed to by the **srcstr** argument; or it could not write to an item descriptor in the item list specified by the **valuelst** argument; or it could not write to the specified auxiliary output buffer; or the **retlen** argument could not be written
SS$_BADPARAM	The item list contains an invalid item code.

$FIND_HELD
Find Identifiers Held by User

Returns the identifiers held by a specified holder.

Format

SYS$FIND_HELD holder ,[id] ,[attrib] ,[contxt]

Arguments

holder

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

Holder whose identifiers are to be found when $FIND_HELD completes execution. The **holder** argument is the address of a quadword data structure containing the holder identifier. This quadword data structure consists of a longword containing the holder UIC, followed by a longword containing the value 0.

id

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

Identifier value found when $FIND_HELD completes execution. The **id** argument is the address of a longword containing the identifier value with which the holder is associated.

attrib

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

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

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

Bit Position	Meaning When Set
KGB$V_DYNAMIC	Allows holders of the identifier to remove it from or add it to the process rights list by using the DCL command SET RIGHTS_LIST.

Bit Position	Meaning When Set
KGB$V_NOACCESS	Makes any access rights of the identifier null and void. This attribute is intended as a modifier for a resource identifier or the Subsystem attribute.
KGB$V_RESOURCE	Allows the holder to charge resources, such as disk blocks, to the identifier.
KGB$V_SUBSYSTEM	Allows holders of the identifier to create and maintain protected subsystems by assigning the Subsystem ACE to the application images in the subsystem.

contxt

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

Context value used when repeatedly calling $FIND_HELD. The **contxt** argument is the address of a longword used while searching for all identifiers. The context value must be initialized to 0, and the resulting context of each call to $FIND_HELD must be presented to each subsequent call. After **contxt** is passed to $FIND_HELD, you must not modify its value.

Description

The Find Identifiers Held by User service returns a list of the identifiers that another identifier holds. Use the $FIND_HELD service to construct the process rights when a user logs in (unless that process has read access to the rights database). To determine all the identifiers held by the specified holder, call $FIND_HELD repeatedly until it returns the status code SS$_NOSUCHID. When SS$_NOSUCHID is returned, $FIND_HELD has returned all the identifiers, cleared the context value, and deallocated the record stream.

If you complete your calls to $FIND_HELD before SS$_NOSUCHID is returned, use $FINISH_RDB to clear the context value and deallocate the record stream.

Note that, when you use wildcards with this service, the records are returned in the order that they were originally written because the first record is located on the basis of the holder ID. Thus, all the target records have the same holder ID or, in other words, they have duplicate keys, which leads to retrieval in the order in which they were written.

Required Access or Privileges

Read access to the rights database is required to obtain information about identifiers held by other users.

Required Quota

None

Related Services

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

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The **id** argument cannot be written by the service, or the **holder**, **attrib**, or **contxt** argument cannot be read by the service.
SS$_IVCHAN	The contents of the **contxt** longword are not valid.
SS$_INSFMEM	The process dynamic memory is insufficient for opening the rights database.
SS$_IVIDENT	The format of the specified holder identifier is invalid.
SS$_NOIOCHAN	No more rights database context streams are available.
SS$_NOSUCHID	The specified holder identifier does not exist, or no further identifiers are held by the specified holder.
RMS$_PRV	You do not have read access to the rights database.

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

Arguments

id

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

Binary identifier value whose holders are found by $FIND_HOLDER. The **id** argument is a longword containing the binary identifier value.

holder

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

Holder identifier returned when $FIND_HOLDER completes execution. The **holder** argument is the address of a quadword containing the holder identifier. The first longword contains the UIC of the holder with the high-order word containing the group number and the low-order word containing the member number. The second longword contains the value 0.

attrib

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

Mask of attributes associated with the holder record specified by **holder**. The **attrib** argument is the address of a longword containing the attribute mask.

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

Bit Position	Meaning When Set
KGB$V_DYNAMIC	Allows holders of the identifier to remove it from or add it to the process rights list by using the DCL command SET RIGHTS_LIST. For more information on SET RIGHTS_LIST, see the *OpenVMS DCL Dictionary*.

Bit Position	Meaning When Set
KGB$V_NOACCESS	Makes any rights of the identifier null and void. This attribute is intended as a modifier for a resource identifier or the Subsystem attribute.
KGB$V_RESOURCE	Allows the holder of an identifier to charge disk space to the identifier. It is used only for file objects.
KGB$V_SUBSYSTEM	Allows holders of an identifier to create and maintain protected subsystems by assigning the Subsystem ACE to the application images in the subsystem.

contxt

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

Context value used while searching for all the holders of the specified identifier when executing $FIND_HOLDER. The **contxt** argument is the address of a longword containing the context value. When calling $FIND_HOLDER repeatedly, **contxt** must be set initially to 0 and the resulting context of each call to $FIND_HOLDER must be presented to each subsequent call. After the argument is passed to $FIND_HOLDER, you must not modify its value.

Description

The Find Holder of Identifier service returns the holder of the specified identifier. To determine all the holders of the specified identifier, you call SYS$FIND_HOLDER repeatedly until it returns the status code SS$_NOSUCHID, which indicates that $FIND_HOLDER has returned all identifiers, cleared the context longword, and deallocated the record stream. If you complete your calls to $FIND_HOLDER before SS$_NOSUCHID is returned, you use the $FINISH_RDB service to clear the context value and deallocate the record stream.

Note that when you use wildcards with this service, the records are returned in the order in which they were originally written. (This action results from the fact that the first record is located on the basis of the identifier. Thus, all the target records have the same identifier or, in other words, they have duplicate keys, which leads to retrieval in the order in which they were written.)

Required Access or Privileges

Read access to the rights database is required to obtain information about identifiers marked HOLDER_HIDDEN.

Required Quota

None

Related Services

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

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The **id** argument cannot be read by the caller, or the **holder**, **attrib**, or **contxt** argument cannot be written by the caller.
SS$_IVCHAN	The contents of the **contxt** longword are not valid.
SS$_INSFMEM	The process dynamic memory is insufficient for opening the rights database.
SS$_IVIDENT	The specified identifier or holder identifier is of invalid format.
SS$_NOIOCHAN	No more rights database context streams are available.
SS$_NOSUCHID	The specified identifier does not exist in the rights database, or no further holders exist for the specified identifier.
RMS$_PRV	The user does not have read access to the rights database.

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

Format

SYS$FINISH_RDB contxt

Argument

contxt

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

Context value to be cleared when $FINISH_RDB completes execution. The **contxt** argument is a longword containing the address of the context value.

Description

The Terminate Rights Database Context service clears the context longword and deallocates the record stream associated with a sequence of rights database lookups performed by the $IDTOASC, $FIND_HOLDER, and $FIND_HELD services.

If you repeatedly call $IDTOASC, $FIND_HOLDER, or $FIND_HELD until SS$_NOSUCHID is returned, you do not need to call $FINISH_RDB because the record stream has already been deallocated and the context longword has already been cleared.

Required Access or Privileges

None

Required Quota

None

Related Services

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

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The **contxt** argument cannot be written by the caller.
SS$_IVCHAN	The contents of the **contxt** longword are not valid.

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

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 VMScluster system.

The **pidadr** argument is optional but must be specified if the process that is to be forced to exit is not in the same UIC group as the calling process.

prcnam

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

Process name of the process that is to be forced to exit. The **prcnam** argument is the address of a character string descriptor pointing to the process name string. A process running on the local node can be identified with a 1- to 15-character string. To identify a process on a particular node in a cluster, specify the full process name, which includes the node name as well as the process name. The full process name can contain up to 23 characters.

The **prcnam** argument can be used only on behalf of processes in the same UIC group as the calling process. To force processes in other groups to exit, you must specify the **pidadr** argument. This restriction exists because the operating system interprets the UIC group number of the calling process as part of the specified process name; the names of processes are unique to UIC groups.

code

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

Completion code value to be used as the exit parameter. The **code** argument is a longword containing this value. If you do not specify the **code** argument, the value 0 is passed as the completion code.

Description

The Force Exit service causes an Exit service call to be issued on behalf of a specified process.

If you specify neither the **pidadr** nor the **prcnam** argument, the caller is forced to exit and control is not returned.

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

The Force Exit system service requires system dynamic memory.

The image executing in the target process follows normal exit procedures. For example, if any exit handlers have been specified, they gain control before the actual exit occurs. Use the Delete Process ($DELPRC) service if you do not want a normal exit.

When a forced exit is requested for a process, a user-mode 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 may need a certain privilege to use $FORCEX:

- You need GROUP privilege to force an exit for a process in the same group that does not have the same UIC as the calling process.
- You need WORLD privilege to force an exit for any process in the system.

Required Quota

None

Related Services

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

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The process name string or string descriptor cannot be read by the caller, or the process identification cannot be written by the caller.
SS$_INCOMPAT	The remote node is running an incompatible version of the operating system.

SS$_INSFMEM	The system dynamic memory is insufficient for the operation.
SS$_IVLOGNAM	The process name string has a length equal to 0 or greater than 15.
SS$_NONEXPR	The specified process does not exist, or an invalid process identification was specified.
SS$_NOPRIV	The process does not have the privilege to force an exit for the specified process.
SS$_NOSUCHNODE	The process name refers to a node that is not currently recognized as part of the cluster.
SS$_REMRSRC	The remote node has insufficient resources to respond to the request. (Bring this error to the attention of your system manager.)
SS$_UNREACHABLE	The remote node is a member of the cluster but is not accepting requests. (This is normal for a brief period early in the system boot process.)

$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]

Arguments

aclent

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

Description of the ACE formatted when $FORMAT_ACL completes execution. The **aclent** argument is the address of a descriptor pointing to a buffer containing the description of the input ACE. The first byte of the buffer contains the length of the ACE; the second byte contains a value that identifies the type of ACE, which in turn determines the ACE format.

For more information about the ACE format, see the Description section.

acllen

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

Length of the output string resulting when $FORMAT_ACL completes execution. The **acllen** argument is the address of a word containing the number of characters written to **aclstr**.

aclstr

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

Formatted ACE resulting when $FORMAT_ACL completes its execution. The **aclstr** argument is the address of a string descriptor pointing to a buffer containing the output string.

width

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

Maximum width of the formatted ACE resulting when $FORMAT_ACL completes its execution. The **width** argument is the address of a word containing the maximum width of the formatted ACE. If this argument is omitted or contains

the value 0, an infinite length display line is assumed. When the width is exceeded, the character specified by **trmdsc** is inserted.

trmdsc

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

Line termination characters used in the formatted ACE. The **trmdsc** argument is the address of a descriptor pointing to a character string containing the termination characters that are inserted for each formatted ACE when the width has been exceeded.

indent

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

Number of blank characters beginning each line of the formatted ACE. The **indent** argument is the address of a word containing the number of blank characters that you want inserted at the beginning of each formatted ACE.

accnam

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

Names of the bits in the access mask when executing the $FORMAT_ACL. The **accnam** argument is the address of an array of 32 quadword descriptors that define the names of the bits in the access mask. Each element points to the name of a bit. The first element names bit 0, the second element names bit 1, and so on.

You can call LIB$GET_ACCNAM to retrieve the access name table for the class of object whose ACL is to be formatted. If you omit **accnam**, the following names are used.

Bit	Name
Bit 0	READ
Bit 1	WRITE
Bit 2	EXECUTE
Bit 3	DELETE
Bit 4	CONTROL
Bit 5	BIT_5
Bit 6	BIT_6
.	
.	
.	
Bit 31	BIT_31

nullarg

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

Placeholding argument reserved to Digital.

Description

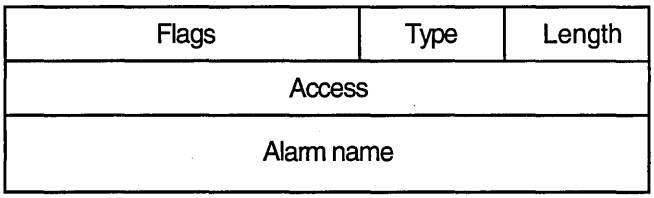
The Format Access Control List Entry service formats the specified access control entry (ACE) into text string representation. There are seven types of ACE:

- Alarm ACE
- Application ACE
- Audit ACE
- Creator ACE
- Default Protection ACE
- Identifier ACE
- Subsystem ACE

The format for each of the ACE types is described in the following sections and the byte offsets and type values for each ACE type are defined in the $ACEDEF system macro library.

Alarm ACE

The access Alarm ACE generates a security alarm. Its format is as follows.



ZK–1710–GE

The following table describes the ACE fields and lists the symbol name for each.

Field	Symbol Name	Description
Length	ACE$B_SIZE	Byte containing the length in bytes of the ACE buffer
Type	ACE$B_TYPE	Byte containing the type value ACE$C_ALARM
Flags	ACE$W_FLAGS	Word containing Alarm ACE information and ACE type-independent information

Field	Symbol Name	Description
Access	ACE$L_ACCESS	Longword containing a mask indicating the access modes to be watched
Alarm name	ACE$T_AUDITNAME	Character string containing the alarm name

The flag field contains information specific to Alarm ACEs and information applicable to all types of ACEs. The following symbols are bit offsets to the Alarm ACE information.

Bit Position	Meaning When Set
ACE$V_SUCCESS	Indicates that the alarm is raised when access is successful
ACE$V_FAILURE	Indicates that the alarm is raised when access fails

The following symbols are bit offsets to ACE information that is independent of ACE type.

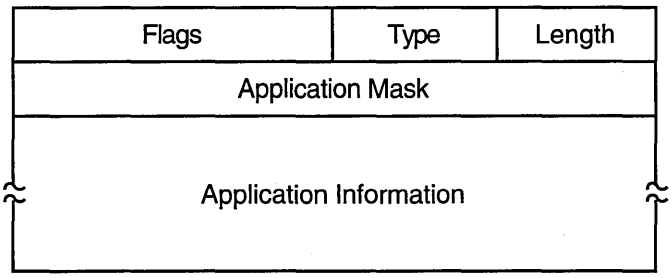
Bit Position	Meaning When Set
ACE$V_DEFAULT	This ACE is added to the ACL of any file created in the directory whose ACL contains this ACE. This bit is applicable only for an ACE in a directory file's ACL.
ACE$V_HIDDEN	This ACE is application dependent. You cannot use the DCL ACL commands and the ACL editor to change the setting; the DCL command DIRECTORY/ACL does not display it.
ACE$V_NOPROPAGATE	This ACE is not propagated among versions of the same file.
ACE$V_PROTECTED	This ACE is not deleted if the entire ACL is deleted; instead you must delete this ACE explicitly.

The following symbol values are offsets to bits within the access mask. You can also obtain the symbol values as masks with the appropriate bit set using the prefix ACE$M rather than ACE$V.

Bit	Meaning When Set
ACE$V_READ	Read access is monitored.
ACE$V_WRITE	Write access is monitored.
ACE$V_EXECUTE	Execute access is monitored.
ACE$V_DELETE	Delete access is monitored.
ACE$V_CONTROL	Modification of the access field is monitored.

Application ACE

The Application ACE contains application-dependent information. Its format is as follows.



ZK–1711–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$C_INFO.
Flags	ACE$W_FLAGS	Word containing Application ACE information and ACE type-independent information.
Application mask	ACE$L_INFO_FLAGS	Longword containing a mask defined and used by the application.
Application information	ACE$T_INFO_START	Variable-length data structure defined and used by the application. The length of this data is implied by the length field.

The flag field contains information specific to Application ACEs and information applicable to all types of ACEs. The following symbol is a bit offset to the Application ACE information.

Bit	Meaning When Set
ACE$V_INFO_TYPE	Four-bit field containing a value indicating whether the application is a CSS application (ACE$C_CSS) or a customer application (ACE$C_CUST).

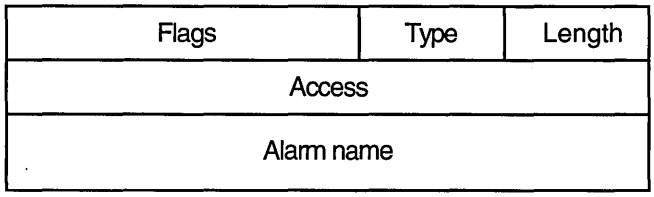
The following symbols are bit offsets to ACE information that is independent of ACE type.

Bit	Meaning When Set
ACE$V_DEFAULT	This ACE is added to the ACL of any file created in the directory whose ACL contains this ACE. This bit is applicable only for an ACE in a directory file's ACL.

Bit	Meaning When Set
ACE$V_HIDDEN	This bit is application dependent. You cannot use the DCL ACL commands and the ACL editor to change the setting; the DCL command DIRECTORY/ACL does not display it.
ACE$V_NOPROPAGATE	This ACE is not propagated among versions of the same file.
ACE$V_PROTECTED	This ACE is not deleted if the entire ACL is deleted; instead you must delete this ACE explicitly.

Audit ACE

The Audit ACE sets a security audit. Its format is as follows.



ZK-1710-GE

The following table describes the ACE fields and lists the symbol name for each.

Field	Symbol Name	Description
Length	ACE$B_SIZE	Byte containing the length in bytes of the ACE buffer
Type	ACE$B_TYPE	Byte containing the type value ACE$C_AUDIT
Flags	ACE$W_FLAGS	Word containing Audit ACE information and ACE type-independent information
Access	ACE$L_ACCESS	Longword containing a mask indicating the access modes to be watched
Alarm name	ACE$T_AUDITNAME	Character string containing the alarm name

The following symbols are bit offsets to ACE information that is independent of ACE type.

Bit Position	Meaning When Set
ACE$V_DEFAULT	This ACE is added to the ACL of any file created in the directory whose ACL contains this ACE. This bit is applicable only for an ACE in a directory file's ACL.

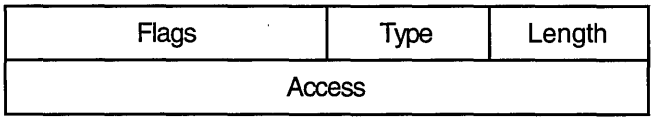
Bit Position	Meaning When Set
ACE$V_HIDDEN	This ACE is application dependent. You cannot use the DCL ACL commands and the ACL editor to change the setting; the DCL command DIRECTORY/ACL does not display it.
ACE$V_NOPROPAGATE	This ACE is not propagated among versions of the same file.
ACE$V_PROTECTED	This ACE is not deleted if the entire ACL is deleted; instead you must delete this ACE explicitly.

The following symbol values are offsets to bits within the access mask. You can also obtain the symbol values as masks with the appropriate bit set using the prefix ACE$M rather than ACE$V.

Bit	Meaning When Set
ACE$V_READ	Read access is monitored.
ACE$V_WRITE	Write access is monitored.
ACE$V_EXECUTE	Execute access is monitored.
ACE$V_DELETE	Delete access is monitored.
ACE$V_CONTROL	Modification of the access field is monitored.

Creator ACE

The Creator ACE controls access to an object based on creators. Its format is as follows.



ZK-5488A-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$C_NEW_OWNER.
Flags	ACE$W_FLAGS	Word containing Creator ACE information and ACE type-independent information.
Access	ACE$L_ACCESS	Longword containing a mask indicating the access modes to be granted to the creator of the file.

The following symbols are bit offsets to ACE information that is independent of ACE type.

Bit	Meaning When Set
ACE$V_NOPROPAGATE	This ACE is not propagated among versions of the same file.
ACE$V_PROTECTED	This ACE is not deleted if the entire ACL is deleted; instead you must delete this ACE explicitly.

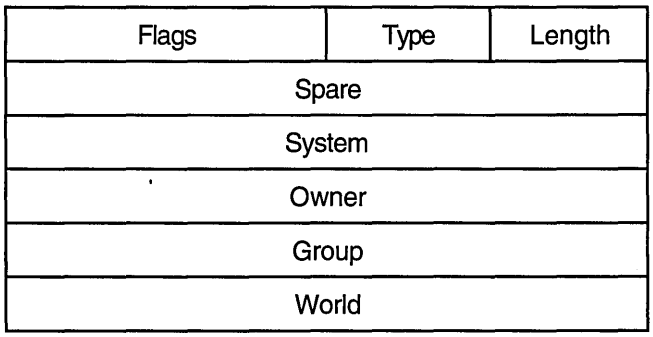
The following symbol values are offsets to bits within the mask indicating the access mode granted in the system, owner, group, and world fields.

Bit Position	Meaning When Set
ACE$V_READ	Read access is granted.
ACE$V_WRITE	Write access is granted.
ACE$V_EXECUTE	Execute access is granted.
ACE$V_DELETE	Delete access is granted.
ACE$V_CONTROL	Modification of the access field is granted.

You can also obtain the symbol values as masks with the appropriate bit set by using the prefix ACE$M rather than ACE$V.

Default Protection ACE

The Default Protection ACE specifies the UIC-based protection for all files created in the directory. You can use this type of ACE only in the ACL of a directory file. Its format is as follows.



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.

Field	Symbol Name	Description
System	ACE$L_SYS_PROT	Longword containing a mask indicating the access mode granted to system users. Each bit represents one type of access.
Owner	ACE$L_OWN_PROT	Longword containing a mask indicating the access mode granted to the owner. Each bit represents one type of access.
Group	ACE$L_GRP_PROT	Longword containing a mask indicating the access mode granted to group users. Each bit represents one type of access.
World	ACE$L_WOR_PROT	Longword containing a mask indicating the access mode granted to the world. Each bit represents one type of access.

The flag field contains information applicable to all types of ACEs. The following symbols are bit offsets to ACE information that is independent of ACE type.

Bit Position	Meaning When Set
ACE$V_HIDDEN	This ACE is application dependent. You cannot use the DCL ACL commands and the ACL editor to change the setting; the DCL command DIRECTORY/ACL does not display it.
ACE$V_NOPROPAGATE	This ACE is not propagated among versions of the same file.
ACE$V_PROTECTED	This ACE is not deleted if the entire ACL is deleted; instead you must delete this ACE explicitly.

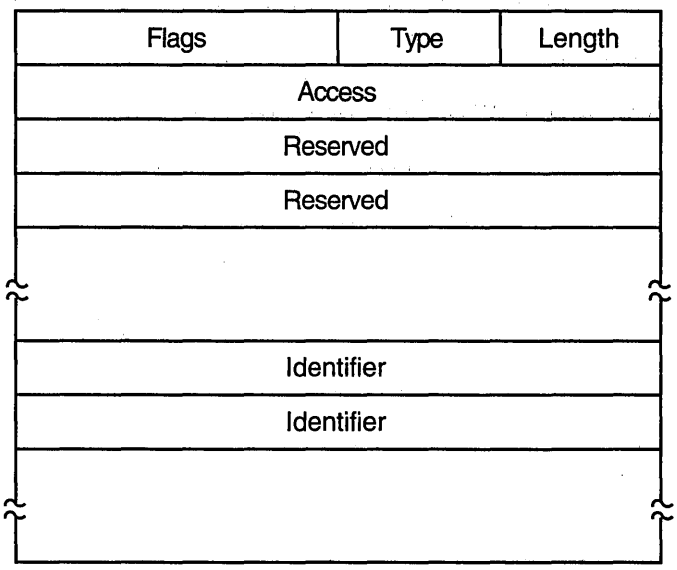
The system interprets the bits within the access mask as shown in the following table. The following symbol values are offsets to bits within the mask indicating the access mode granted in the system, owner, group, and world fields.

Bit Position	Meaning When Set
ACE$V_READ	Read access is granted.
ACE$V_WRITE	Write access is granted.
ACE$V_EXECUTE	Execute access is granted.
ACE$V_DELETE	Delete access is granted.

You can also obtain the symbol values as masks with the appropriate bit set by using the prefix ACE$M rather than ACE$V.

Identifier ACE

The Identifier ACE controls access to an object based on identifiers. Its format is as follows.



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.

Bit	Meaning When Set
ACE$V_RESERVED	Four-bit field containing the number of longwords to reserve for application-dependent data. The number must be between 0 and 15. The reserved longwords, if any, immediately precede the identifiers.

The following symbols are bit offsets to ACE information that is independent of ACE type.

Bit	Meaning When Set
ACE$V_DEFAULT	This ACE is added to the ACL of any file created in the directory whose ACL contains this ACE. This bit is applicable only for an ACE in a directory file's ACL.
ACE$V_HIDDEN	This bit is application dependent. You cannot use the DCL ACL commands and the ACL editor to change the setting; the DCL command DIRECTORY/ACL does not display it.
ACE$V_NOPROPAGATE	This ACE is not propagated among versions of the same file.
ACE$V_PROTECTED	This ACE is not deleted if the entire ACL is deleted; instead you must delete this ACE explicitly.

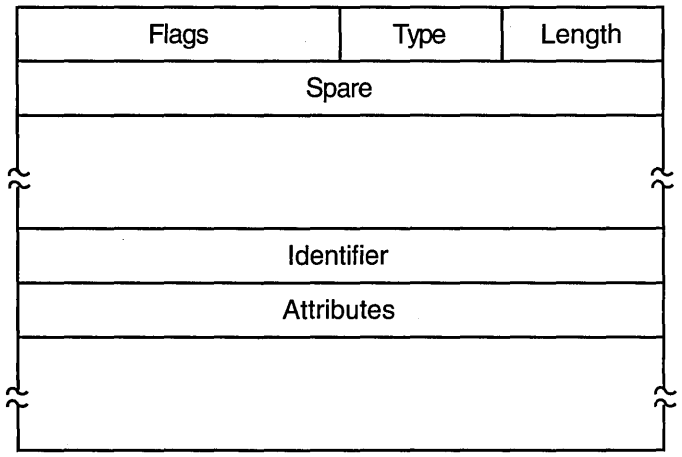
The following symbol values are offsets to bits within the mask indicating the access mode granted in the system, owner, group, and world fields.

Bit Position	Meaning When Set
ACE$V_READ	Read access is granted.
ACE$V_WRITE	Write access is granted.
ACE$V_EXECUTE	Execute access is granted.
ACE$V_DELETE	Delete access is granted.
ACE$V_CONTROL	Modification of the access field is granted.

You can also obtain the symbol values as masks with the appropriate bit set by using the prefix ACE$M rather than ACE$V.

Subsystem ACE

The Subsystem ACE maintains protected subsystems. Its format is as follows.



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.

Bit Position	Meaning When Set
ACE$V_READ	Read access is granted.
ACE$V_WRITE	Write access is granted.
ACE$V_EXECUTE	Execute access is granted.
ACE$V_DELETE	Delete access is granted.
ACE$V_CONTROL	Modification of the access field is granted.

You can also obtain the symbol values as masks with the appropriate bit set by using the prefix ACE$M rather than ACE$V.

Required Access or Privileges

None

Required Quota

None

Related Services

$ADD_HOLDER, $ADD_IDENT, $ASCTOID, $CREATE_RDB, $CREATE_USER_PROFILE, $FIND_HELD, $FIND_HOLDER, $FINISH_RDB, $FORMAT_AUDIT, $GET_SECURITY, $GRANTID, $HASH_PASSWORD, $IDTOASC, $MOD_HOLDER, $MOD_IDENT, $REM_HOLDER, $REM_IDENT, $REVOKID, $SET_RESOURCE_DOMAIN, $SET_SECURITY

Condition Values Returned

SS$_BUFFEROVF	The service completed successfully. The output string has overflowed the buffer and has been truncated.
SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The ACL entry or its descriptor cannot be read by the caller, or the string descriptor cannot be read by the caller, or the length word or the string buffer cannot be written by the caller.

$FORMAT_AUDIT
Format Security Audit Event Message

Converts a security auditing event message from binary format to ASCII text.

Format

SYS$FORMAT_AUDIT [fmttyp] ,audmsg ,[outlen] ,[outbuf] ,[width] ,[trmdsc] ,[routin] ,[fmtflg]

Arguments

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

Format for the message. The **fmttyp** argument is a value indicating whether the security audit message should be in brief format, which is one line of information, or full format. The default is full format. See the *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 character descriptor pointing to a buffer containing the message that requires formatting.

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

Length of the formatted security audit message. The **outlen** argument is the address of the word receiving the final length of the ASCII message.

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

Buffer holding the formatted message. The **outbuf** argument is the address of a descriptor pointing to the buffer receiving the message.

width

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

Maximum width of the formatted message. The **width** argument is the address of a word containing the line width value. The default is 80 columns.

The **width** argument does not work consistently. In most cases, if you specify both the **width** argument and the full format style (NSA$C_FORMAT_STYLE_FULL), $FORMAT_AUDIT ignores the **width** argument. The minimum width is 80 columns; lower values do not limit the width to less than 80. If you specify a width greater than 80 columns, most lines are not joined to use the full width.

In most cases, you should avoid using the **width** argument.

trmdsc

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

Line termination characters used in a full format message. The **trmdsc** argument is the address of a descriptor pointing to the line termination characters to insert within a line segment whenever the **width** is reached.

routin

OpenVMS usage:	procedure
type:	procedure value
access:	read only
mechanism:	by reference

Routine that writes a formatted line to the output buffer. The **routin** argument is the address of a routine called each time a line segment is formatted. The argument passed to the routine is the address of a character string descriptor for the line segment.

When an application wants event messages in the brief format, $FORMAT_AUDIT calls the routine twice to format the first event message. The first time it is called, the routine passes a string containing the column titles for the message. The second and subsequent calls to the routine pass the formatted event message. By using this routine argument, a caller can gain control at various points in the processing of an audit event message.

fmtflg

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

Determines the formatting of certain kinds of audit messages. The **fmtflg** argument is a mask specifying whether sensitive information should be displayed or column titles built for messages in brief format. For example, the operating

system uses bit 0 to suppress plaintext passwords from security alarm messages. The following table describes the significant bits.

Bit	Value	Description
0	1	Do not format sensitive information.
	0	Format sensitive information.
1	1	Build a column title for messages in brief format. (You must specify a **fmttyp** of brief and a **routin** argument.)
	0	Do not build column titles.

Description

The Format Audit service converts a security auditing event message from binary format to ASCII text and can filter sensitive information. $FORMAT_AUDIT allows the caller to format a message in a multiple-line format or a single-line format and tailor the information for a display device of a specific width.

$FORMAT_AUDIT is intended for utilities that need to format the security auditing event messages received from the audit server listener mailbox or the system security audit log file.

Required Access or Privileges

None

Required Quota

$FORMAT_AUDIT can cause a process to exceed its page-file quota (PGFLQUOTA) if it has to format a long auditing event message. The caller of $FORMAT_AUDIT can also receive quota violations from services that $FORMAT_AUDIT uses, such as $IDTOASC, $FAO, and $GETMSG.

Related Services

$AUDIT_EVENT

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_MSGNOTFND	The service completed successfully; however, the message code cannot be found and a default message has been returned.
SS$_ACCVIO	The item list cannot be read by the caller, or the buffer length or buffer cannot be written by the caller.
SS$_BADPARAM	The item list contains an invalid identifier.
SS$_BUFFEROVF	The service completed successfully; however, the formatted output string overflowed the output buffer and has been truncated.
SS$_INSFMEM	The process dynamic memory is insufficient for opening the rights database.

SS$_IVCHAN	The format of the specified identifier is not valid. This condition value returned is not directly returned by $FORMAT_AUDIT. It is indirectly returned when $FORMAT_AUDIT in turn calls another service, such as an identifier translation or binary time translation service.
SS$_IVIDENT	The format of the specified identifier is invalid.
SS$_NOSUCHID	The specified identifier name does not exist in the rights database. This condition value returned is not directly returned by $FORMAT_AUDIT. It is indirectly returned when $FORMAT_AUDIT in turn calls another service, such as an identifier translation or binary time translation service.

$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]

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**.

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 may be either a physical device name or a logical name. If the first character in the string is an underscore (_), the string is considered a physical device name; otherwise, the string is considered a logical name and logical name translation is performed until either a physical device name is found or the system default number of translations has been performed.

If the device name string contains a colon (:), the colon and the characters that follow it are ignored.

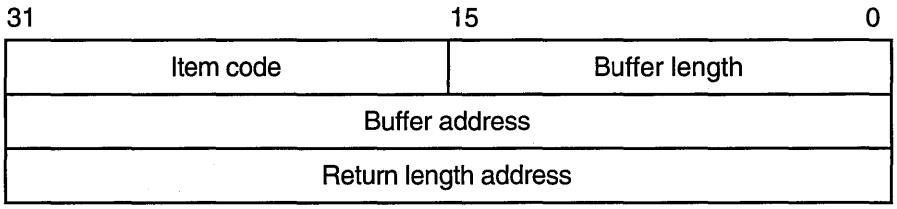
To identify a device to $GETDVI, you can specify either the **chan** or **devnam** argument, but you should not specify both. If both arguments are specified, the **chan** argument is used.

If you specify neither **chan** nor **devnam**, $GETDVI uses a default value of 0 for **chan**.

itmlst

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 upon the item code specified in the item code field of the item descriptor. If the value of buffer length is too long, $GETDVI truncates the data.

Descriptor Field	Definition
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 Digital.

Though this argument is optional, Digital strongly recommends that you specify it, for the following reasons:

- If you are using an event flag to signal the completion of the service, you can test the I/O status block for a condition value to be sure that the event flag was not set by an event other than service completion.
- If you are using the $SYNCH service to synchronize completion of the service, the I/O status block is a required argument for $SYNCH.
- The condition value returned in R0 and the condition value returned in the I/O status block provide information about different aspects of the call to the $GETDVI service. The condition value returned in R0 gives you information about the success or failure of the service call itself; the condition value returned in the I/O status block gives you information about the success or failure of the service operation. Therefore, to accurately assess the success or failure of the call to $GETDVI, you must check the condition values returned in both R0 and the I/O status block.

astadr

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

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

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

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

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

nullarg
OpenVMS usage: null_arg
type: quadword (unsigned)
access: read only
mechanism: by reference

Placeholding argument reserved to Digital.

Item Codes

DVI$_ACPPID
When you specify DVI$_ACPPID, $GETDVI returns the ACP process ID as a 4-byte hexadecimal number.

DVI$_ACPTYPE
When you specify DVI$_ACPTYPE, $GETDVI returns the ACP type code as a 4-byte hexadecimal number. The following symbols define each of the ACP type codes that $GETDVI can return.

Symbol	Description
DVI$C_ACP_F11V1	Files–11 Level 1
DVI$C_ACP_F11V2	Files–11 Level 2
DVI$C_ACP_MTA	Magnetic tape
DVI$C_ACP_NET	Networks
DVI$C_ACP_REM	Remote I/O

DVI$_ALLDEVNAM
When you specify DVI$_ALLDEVNAM, $GETDVI returns the allocation-class device name, which is a 64-byte hexadecimal string. The allocation-class device name uniquely identifies each device that is currently connected to any node in a VMScluster 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 DVI$_DEVLOCKNAM item code should be used to coordinate access to volumes.

DVI$_ALLOCLASS

When you specify DVI$_ALLOCLASS, $GETDVI returns the allocation class of the host as a longword integer between 0 and 255. An allocation class is a unique number between 0 and 255 that the system manager assigns to a pair of hosts and the dual-pathed devices that the hosts make available to other nodes in the 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 *VMScluster Systems for OpenVMS*.

DVI$_ALT_HOST_AVAIL

When you specify DVI$_ALT_HOST_AVAIL, $GETDVI returns a longword that is interpreted as Boolean. A value of 1 indicates that the host serving the alternate path is available; a value of 0 indicates that it is not available.

The host is the node that makes the device available to other nodes in the VMScluster 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.

DVI$_ALT_HOST_NAME

When you specify DVI$_ALT_HOST_NAME, $GETDVI returns the name of the host serving the alternate path as a 64-byte zero-filled string.

For more information about hosts, dual-pathed devices, and primary and alternate paths, refer to the description of the DVI$_ALT_HOST_AVAIL item code.

DVI$_ALT_HOST_TYPE

When you specify DVI$_ALT_HOST_TYPE, $GETDVI returns, as a 4-byte string, the hardware type of the host serving the alternate path. Each hardware type has a symbolic name.

VAX

The following table shows each symbolic name and the host it denotes on VAX systems.

Name	Host
VAX	Any VAX family processor
HS50	HSC50
HS70	HSC70♦

Alpha

The following table shows each symbolic name and the host it denotes on Alpha systems.

Name	Host
Alpha	Any Alpha family processor
HS50	HSC50
HS70	HSC70♦

For more information about hosts, dual-pathed devices, and primary and alternate paths, refer to the description of the DVI$_ALT_HOST_AVAIL item code.

DVI$_CLUSTER
When you specify DVI$_CLUSTER, $GETDVI returns the volume cluster size as a 4-byte decimal number. This item code is applicable only to disks.

DVI$_CYLINDERS
When you specify DVI$_CYLINDERS, $GETDVI returns the number of cylinders on the volume as a 4-byte decimal number. This item code is applicable only to disks.

DVI$_DEVBUFSIZ
When you specify DVI$_DEVBUFSIZ, $GETDVI returns the device buffer size (for example, the width of a terminal or the block size of a tape) as a 4-byte decimal number.

DVI$_DEVCHAR
When you specify DVI$_DEVCHAR, $GETDVI returns device-independent characteristics as a 4-byte bit vector. Each characteristic is represented by a bit. When $GETDVI sets a bit, the device has the corresponding characteristic. Each bit in the vector has a symbolic name. The $DEVDEF macro defines the following symbolic names.

Symbol	Description
DEV$V_REC	Device is record oriented.
DEV$V_CCL	Device is a carriage control device.
DEV$V_TRM	Device is a terminal.
DEV$V_DIR	Device is directory structured.
DEV$V_SDI	Device is single-directory structured.

Symbol	Description
DEV$V_SQD	Device is sequential and block oriented.
DEV$V_SPL	Device is being spooled.
DEV$V_OPR	Device is an operator.
DEV$V_RCT	Disk contains Revector Cache Table (RCT). This bit is set for every DAA disk.
DEV$V_NET	Device is a network device.
DEV$V_FOD	Device is files oriented.
DEV$V_DUA	Device is dual ported.
DEV$V_SHR	Device is shareable.
DEV$V_GEN	Device is a generic device.
DEV$V_AVL	Device is available for use.
DEV$V_MNT	Device is mounted.
DEV$V_MBX	Device is a mailbox.
DEV$V_DMT	Device is marked for dismount.
DEV$V_ELG	Device has error logging enabled.
DEV$V_ALL	Device is allocated.
DEV$V_FOR	Device is mounted foreign.
DEV$V_SWL	Device is software write locked.
DEV$V_IDV	Device can provide input.
DEV$V_ODV	Device can provide output.
DEV$V_RND	Device allows random access.
DEV$V_RTM	Device is a real-time device.
DEV$V_RCK	Device has read-checking enabled.
DEV$V_WCK	Device has write-checking enabled.

Note that each device characteristic has its own individual $GETDVI item code with the format DVI$_xxxx, where *xxxx* are the characters following the underscore character in the symbolic name for that device characteristic.

For example, when you specify the item code DVI$_REC, $GETDVI returns a longword value that is interpreted as Boolean. If the value is 0, the device is not record oriented; if the value is 1, it is record oriented. This information is identical to that returned in the DEV$V_REC bit of the longword vector specified by the DVI$_DEVCHAR item code.

The buffer must specify a longword for all of these device-characteristic item codes.

DVI$_DEVCHAR2

When you specify DVI$_DEVCHAR2, $GETDVI returns additional device-independent characteristics as a 4-byte bit vector. Each bit in the vector, when set, corresponds to a symbolic name. The $DEVDEF macro defines the following symbolic names.

Symbol	Description
DEV$V_CLU	Device is available clusterwide.
DEV$V_DET	Device is detached terminal.
DEV$V_RTT	Device has remote terminal UCB extension.
DEV$V_CDP	Dual-pathed device with two UCBs.
DEV$V_2P	Two paths are known to this device.
DEV$V_MSCP	Device accessed using MSCP (disk or tape). Before using this bit to differentiate between types of disk and tape devices, be sure that no other more appropriate differentiation mechanism exists.
DEV$V_SSM	Device is a shadow set member.
DEV$V_SRV	Device is served by the MSCP server.
DEV$V_RED	Device is redirected terminal.
DEV$V_NNM	Device has node$ prefix.
DEV$V_WBC	Device supports write-back caching.
DEV$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

When you specify DVI$_DEVCLASS, $GETDVI returns the device class as a 4-byte decimal number. Each class has a corresponding symbol. The $DCDEF macro defines these symbols. The following table describes each device class symbol.

Symbol	Description
DC$_DISK	Disk device
DC$_TAPE	Tape device
DC$_SCOM	Synchronous communications device

Symbol	Description
DC$_CARD	Card reader
DC$_TERM	Terminal
DC$_LP	Line printer
DC$_REALTIME	Real-time
DC$_MAILBOX	Mailbox
DC$_MISC	Miscellaneous device

DVI$_DEVDEPEND
When you specify DVI$_DEVDEPEND, $GETDVI returns device-dependent characteristics as a 4-byte bit vector. To determine what information is returned for a particular device, refer to the *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$_DEVDEPEND2
When you specify DVI$_DEVDEPEND2, $GETDVI 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.

Alpha

DVI$_DEVICE_TYPE_NAME
On Alpha systems, when you specify DVI$_DEVICE_TYPE_NAME, $GETDVI returns a string identifying the type of the device about which information was requested.♦

DVI$_DEVLOCKNAM
When you specify DVI$_DEVLOCKNAM, $GETDVI returns the device lock name, which is a 64-byte hexadecimal string. The device lock name uniquely identifies each volume or volume set in a VMScluster 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 a VMScluster system to access a uniquely identified volume.

One use for the device lock name might be in an application wherein processes need to coordinate their access to files using the lock manager. In this case, the program would make the file a resource to be locked by the lock manager, specifying as the resource name the following concatenated components: (1) a user facility prefix followed by an underscore character, (2) the device lock name of the volume on which the file resides, and (3) the file ID of the file.

DVI$_DEVNAM
When you specify DVI$_DEVNAM, $GETDVI returns the device name as a 64-byte, zero-filled string. The node name is also returned.

DVI$_DEVSTS
When you specify DVI$_DEVSTS, $GETDVI returns device-dependent status information as a 4-byte bit vector. The $UCBDEF macro defines symbols for the status bits. For this device-dependent information, refer to the *OpenVMS I/O User's Reference Manual.*

DVI$_DEVTYPE
When you specify DVI$_DEVTYPE, $GETDVI returns the device type as a 4-byte decimal number. The $DCDEF macro defines symbols for the device types.

DVI$_DFS_ACCESS
When you specify DVI$_DFS_ACCESS, $GETDVI 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$_DISPLAY_DEVNAM
When you specify DVI$_DISPLAY_DEVNAM, $GETDVI returns the preferred device name for user displays as a 256-byte zero-filled string. The DVI$_DISPLAY_DEVNAM item code is not recommended for use with the $ASSIGN service. Use the DVI$_ALLDEVNAM item code to return an allocation class device name that is usable as input to a program.

DVI$_ERRCNT
When you specify DVI$_ERRCNT, $GETDVI returns the device's error count as a 4-byte decimal number.

DVI$_FREEBLOCKS
When you specify DVI$_FREEBLOCKS, $GETDVI returns the number of free blocks on a disk as a 4-byte decimal number. This item code is applicable only to disks.

DVI$_FULLDEVNAM
When you specify DVI$_FULLDEVNAM, $GETDVI returns the node name and device name as a 64-byte, zero-filled string.

The DVI$_FULLDEVNAM item code is useful in a VMScluster environment because, unlike DVI$_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$_HOST_AVAIL
When you specify DVI$_HOST_AVAIL, $GETDVI returns a longword, which is interpreted as Boolean. A value of 1 indicates that the host serving the primary path is available; a value of 0 indicates that it is not 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$_HOST_COUNT
When you specify DVI$_HOST_COUNT, $GETDVI returns, as a longword integer, the number of hosts that make the device available to other nodes in the VMScluster 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$_HOST_NAME
When you specify DVI$_HOST_NAME, $GETDVI 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$_HOST_TYPE
When you specify DVI$_HOST_TYPE, $GETDVI returns, as a 4-byte string, the type of host serving the primary path. Each hardware type has a symbolic name.

VAX

The following table shows each symbolic name and the host it denotes on VAX systems.

Name	Host
VAX	Any VAX family processor
HS50	HSC50
HS70	HSC70♦

Alpha

The following table shows each symbolic name and the host it denotes on Alpha systems.

Name	Host
Alpha	Any Alpha family processor
HS50	HSC50
HS70	HSC70♦

For more information about hosts, dual-pathed devices, and primary and alternate paths, refer to the description of the DVI$_ALT_HOST_AVAIL item code.

DVI$_LOCKID
When you specify DVI$_LOCKID, $GETDVI returns the lock ID of the lock on a disk. The lock manager locks a disk if it is available to all nodes in a VMScluster system and it is either allocated or mounted. A disk is available to all nodes in a VMScluster system if, for example, it is served by an HSC controller or MSCP server or if it is a dual-ported MASSBUS disk.

The buffer must specify a longword into which $GETDVI is to return the 4-byte hexadecimal lock ID.

DVI$_LOGVOLNAM
When you specify DVI$_LOGVOLNAM, $GETDVI returns the logical name of the volume or volume set as a 64-byte string.

DVI$_MAXBLOCK
When you specify DVI$_MAXBLOCK, $GETDVI returns the maximum number of blocks on the volume as a 4-byte decimal number. This item code is applicable only to disks.

DVI$_MAXFILES
When you specify DVI$_MAXFILES, $GETDVI returns the maximum number of files on the volume as a 4-byte decimal number. This item code is applicable only to disks.

DVI$_MEDIA_ID
When you specify DVI$_MEDIA_ID, $GETDVI returns the nondecoded media ID as a longword. This item code is applicable only to disks and tapes.

DVI$_MEDIA_NAME
When you specify DVI$_MEDIA_NAME, $GETDVI returns the name of the volume type (for example, RK07 or TA78) as a 64-byte, zero-filled string. This item code is applicable only to disks and tapes.

DVI$_MEDIA_TYPE
When you specify DVI$_MEDIA_TYPE, $GETDVI returns the device name prefix of the volume (for example, DM for an RK07 device or MU for a TA78 device) as a 64-byte, zero-filled string. This item code is applicable only to disks and tapes.

DVI$_MOUNTCNT
When you specify DVI$_MOUNTCNT, $GETDVI returns the mount count for the volume as a 4-byte decimal number.

DVI$_MSCP_UNIT_NUMBER
When you specify DVI$_MSCP_UNIT_NUMBER, $GETDVI returns the internal coded value for MSCP unit numbers as a longword integer. This item code is reserved to Digital.

DVI$_NEXTDEVNAM
When you specify DVI$_NEXTDEVNAM, $GETDVI returns the device name of the next volume in the volume set as a 64-byte, zero-filled string. The node name is also returned. This item code is applicable only to disks.

DVI$_OPCNT
When you specify DVI$_OPCNT, $GETDVI returns the operation count for the volume as a 4-byte decimal number.

DVI$_OWNUIC
When you specify DVI$_OWNUIC, $GETDVI returns the user identification code (UIC) of the owner of the device as a standard 4-byte UIC.

DVI$_PID
When you specify DVI$_PID, $GETDVI returns the process identification (PID) of the owner of the device as a 4-byte hexadecimal number.

DVI$_RECSIZ
When you specify DVI$_RECSIZ, $GETDVI returns the blocked record size as a 4-byte decimal number.

DVI$_REFCNT
When you specify DVI$_REFCNT, $GETDVI returns the number of channels assigned to the device as a 4-byte decimal number.

DVI$_REMOTE_DEVICE
When you specify DVI$_REMOTE_DEVICE, $GETDVI returns a longword, which is interpreted as Boolean. A value of 1 indicates that the device is a remote device; a value of 0 indicates that it is not a remote device. A remote device is a device that is not directly connected to the local node, but instead is visible through the VMScluster system.

DVI$_ROOTDEVNAM
When you specify DVI$_ROOTDEVNAM, $GETDVI returns the device name of the root volume in the volume set as a 64-byte, zero-filled string. This item code is applicable only to disks.

DVI$_SECTORS
When you specify DVI$_SECTORS, $GETDVI returns the number of sectors per track as a 4-byte decimal number. This item code is applicable only to disks.

DVI$_SERIALNUM
When you specify DVI$_SERIALNUM, $GETDVI returns the serial number of the volume as a 4-byte decimal number. This item code is applicable only to disks.

DVI$_SERVED_DEVICE
When you specify DVI$_SERVED_DEVICE, $GETDVI returns a longword, which is interpreted as Boolean. A value of 1 indicates that the device is a served device; a value of 0 indicates that it is not a served device. A served device is one whose local node makes it available to other nodes in the VMScluster system.

DVI$_SHDW_CATCHUP_COPYING
When you specify DVI$_SHDW_CATCHUP_COPYING, $GETDVI returns a longword, which is interpreted as Boolean. The value 1 indicates that the device is the target of a full copy operation.

DVI$_SHDW_FAILED_MEMBER
When you specify DVI$_SHDW_FAILED_MEMBER, $GETDVI returns a longword, which is interpreted as Boolean. The value 1 indicates that the device is a member that has been removed from the shadow set by the remote server. The DVI$_SHDW_FAILED_MEMBER item code is for use only with VAX Volume Shadowing (phase I).

DVI$_SHDW_MASTER
When you specify DVI$_SHDW_MASTER, $GETDVI returns a longword, which is interpreted as Boolean. The value 1 indicates that the device is a virtual unit.

DVI$_SHDW_MASTER_NAME
When you specify DVI$_SHOW_MASTER_NAME and 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 a VMScluster system. See *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$_SHDW_MEMBER
When you specify DVI$_SHDW_MEMBER, $GETDVI returns a longword, which is interpreted as Boolean. The value 1 indicates that the device is a shadow set member.

DVI$_SHDW_MERGE_COPYING
When you specify DVI$_SHDW_MERGE_COPYING, $GETDVI returns a longword, which is interpreted as Boolean. The value 1 indicates that the device is a merge member of the shadow set.

DVI$_SHDW_NEXT_MBR_NAME
When you specify DVI$_SHDW_NEXT_MBR_NAME, $GETDVI returns the device name of the next member in the shadow set. If you specify a virtual unit, $GETDVI returns the shadow set member device names in random order. If you specify the name of a device that is neither a virtual unit nor a member, $GETDVI returns a null string.

$GETDVI returns the device name of the next member in the shadow set even if the remote server has removed the next member from the shadow set.

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 255DUS10.

Note

Shadow set members must have a nonzero allocation class to operate in a VMScluster system. See *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$_STS
When you specify DVI$_STS, $GETDVI returns the device unit status as a 4-byte bit vector. Each bit in the vector, when set, corresponds to a symbolic name that is defined by the $UCBDEF macro. The following table describes each name.

Symbol	Description
UCB$V_TIM	Timeout is enabled.
UCB$V_INT	Interrupt is expected.
UCB$V_ERLOGIP	Error log is in progress on unit.
UCB$V_CANCEL	I/O on unit is canceled.
UCB$V_ONLINE	Unit is on line.
UCB$V_POWER	Power failed while unit busy.
UCB$V_TIMOUT	Unit timed out.
UCB$V_INTTYPE	Receiver interrupt.
UCB$V_BSY	Unit is busy.
UCB$V_MOUNTING	Device is being mounted.
UCB$V_DEADMO	Deallocate at dismount.
UCB$V_VALID	Volume is software valid.
UCB$V_UNLOAD	Unload volume at dismount.
UCB$V_TEMPLATE	Template UCB from which other UCBs for this device type are made.
UCB$V_MNTVERIP	Mount verification is in progress.
UCB$V_WRONGVOL	Wrong volume detected during mount verification.
UCB$V_DELETEUCB	Delete this UCB when reference count equals 0.

DVI$_TRACKS
When you specify DVI$_TRACKS, $GETDVI returns the number of tracks per cylinder as a 4-byte decimal number. This item code is applicable only to disks.

DVI$_TRANSCNT
When you specify DVI$_TRANSCNT, $GETDVI returns the transaction count for the volume as a 4-byte decimal number.

DVI$_TT_ACCPORNAM
When you specify DVI$_TT_ACCPORNAM, $GETDVI returns the name of the remote access port associated with a channel number or 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.

DVI$_TT_CHARSET
When you specify DVI$TT_CHARSET, $GETDVI 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$_TT_CS_HANGUL
When you specify DVI$_TT_CS_HANGUL, $GETDVI 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$_TT_CS_HANYU
When you specify DVI$_TT_CS_HANYU, $GETDVI 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$_TT_CS_HANZI
When you specify DVI$_TT_CS_HANZI, $GETDVI 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$_TT_CS_KANA
When you specify DVI$_TT_CS_KANA, $GETDVI 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$_TT_CS_KANJI
When you specify DVI$_TT_CS_KANJI, $GETDVI 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$_TT_CS_THAI
When you specify DVI$_TT_CS_THAI, $GETDVI 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$_TT_PHYDEVNAM
When you specify DVI$_TT_PHYDEVNAM, $GETDVI returns a string containing the physical device name of a terminal. If the caller specifies a disconnected virtual terminal or a device that is not a terminal, $GETDVI returns a null string. $GETDVI returns the physical device name as a 64-byte zero-filled string.

DVI$_UNIT
When you specify DVI$_UNIT, $GETDVI returns the unit number as a 4-byte decimal number.

DVI$_VOLCOUNT
When you specify DVI$_VOLCOUNT, $GETDVI returns the number of volumes in the volume set as a 4-byte decimal number. This item code is applicable only to disks.

DVI$_VOLNAM
When you specify DVI$_VOLNAM, $GETDVI returns the volume name as a 12-byte zero-filled string.

DVI$_VOLNUMBER
When you specify DVI$_VOLNUMBER, $GETDVI returns the volume number of this volume in the volume set as a 4-byte decimal number. This item code is applicable only to disks.

DVI$_VOLSETMEM
When you specify DVI$_VOLSETMEM, $GETDVI returns a longword value, which is interpreted as Boolean. A value of 1 indicates that the device is part of a volume set; a value of 0 indicates that it is not. This item code is applicable only to disks.

DVI$_VPROT
When you specify DVI$_VPROT, $GETDVI returns the volume protection mask as a standard 4-byte protection mask.

DVI$_TT_xxxx
DVI$_TT_xxxx is the format for a series of item codes that return information about terminals. This information consists of terminal characteristics. The *xxxx* portion of the item code name specifies a single terminal characteristic.

Each of these item codes requires that the buffer specify a longword into which $GETDVI will write a 0 or 1: 0 if the terminal does not have the specified characteristic, and 1 if the terminal does have it. The one exception is the DVI$_TT_PAGE item code, which when specified causes $GETDVI to return a decimal longword value that is the page size of the terminal.

You can also obtain this terminal-specific information by using the DVI$_DEVDEPEND and DVI$_DEVDEPEND2 item codes. Each of these two item codes specifies a longword bit vector wherein each bit corresponds to a terminal characteristic; $GETDVI sets the corresponding bit for each characteristic possessed by the terminal.

Following is a list of the item codes that return information about terminal characteristics. For information about these characteristics, refer to the description of the F$GETDVI lexical function in the *OpenVMS DCL Dictionary*.

DVI$_TT_NOECHO
DVI$_TT_HOSTSYNC
DVI$_TT_ESCAPE
DVI$_TT_MECHTAB
DVI$_TT_LFFILL
DVI$_TT_CRFILL
DVI$_TT_EIGHTBIT
DVI$_TT_READSYNC
DVI$_TT_NOBRDCST
DVI$_TT_MODEM
DVI$_TT_LOCALECHO
DVI$_TT_PAGE
DVI$_TT_MODHANGUP
DVI$_TT_DMA
DVI$_TT_ANSICRT
DVI$_TT_AVO
DVI$_TT_BLOCK
DVI$_TT_EDITING
DVI$_TT_DIALUP
DVI$_TT_FALLBACK
DVI$_TT_PASTHRU
DVI$_TT_PRINTER
DVI$_TT_DRCS
DVI$_TT_DECCRT2
DVI$_TT_DECCRT3
DVI$_TT_DECCRT4
DVI$_TT_NOTYPEAHD
DVI$_TT_TTSYNC
DVI$_TT_LOWER
DVI$_TT_WRAP
DVI$_TT_SCOPE
DVI$_TT_SETSPEED
DVI$_TT_MBXDSABL
DVI$_TT_MECHFORM
DVI$_TT_HALFDUP
DVI$_TT_OPER
DVI$_TT_AUTOBAUD
DVI$_TT_HANGUP
DVI$_TT_BRDCSTMBX
DVI$_TT_ALTYPEAHD
DVI$_TT_REGIS
DVI$_TT_EDIT
DVI$_TT_DECCRT
DVI$_TT_INSERT
DVI$_TT_SECURE
DVI$_TT_DISCONNECT
DVI$_TT_SIXEL
DVI$_TT_APP_KEYPAD
DVI$_TT_SYSPWD

DVI$_yyyy

DVI$_yyyy is the format for a series of item codes that return device-independent characteristics of a device. There is an item code for each device characteristic returned in the longword bit vector specified by the DVI$_DEVCHAR item code.

In the description of the DVI$_DEVCHAR item code is a list of symbol names in which each symbol represents a device characteristic. To construct the $GETDVI item code for each device characteristic, substitute for *yyyy* that portion of the symbol name that follows the underscore character. For example, the DVI$_REC item code returns the same information as the DEV$V_REC bit in the DVI$_DEVCHAR longword bit vector.

The buffer for each of these item codes must specify a longword value, which is interpreted as Boolean. The $GETDVI service writes the value 1 into the longword if the device has the specified characteristic and the value 0 if it does not.

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 logically OR the item code specifying the information desired with the code DVI$C_SECONDARY.

You can obtain information about primary and secondary devices in a single call to $GETDVI.

In most cases, the two sets of characteristics (primary and secondary) returned by $GETDVI are identical. However, the two sets provide different information in the following cases:

- If the device has an associated mailbox, the primary characteristics are those of the assigned device and the secondary characteristics are those of the associated mailbox.
- If the device is a spooled device, the primary characteristics are those of the intermediate device (such as the disk) and the secondary characteristics are those of the spooled device (such as the printer).
- If the device represents a logical link on the network, the secondary characteristics contain information about the link.

Unless otherwise stated in the description of the item code, $GETDVI returns information about the local node only.

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, $MOUNT, $PUTMSG, $QIO, $QIOW, $SNDERR, $SNDJBC, $SNDJBCW, $SNDOPR

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The device name string descriptor, device name string, or **itmlst** argument cannot be read; or the buffer or return length longword cannot be written by the caller.
SS$_BADPARAM	The item list contains an invalid item code, or the buffer address field in an item descriptor specifies less than four bytes for the return length information.
SS$_EXASTLM	The process has exceeded its AST limit quota.
SS$_IVCHAN	You specified an invalid channel number, that is, a channel number larger than the number of channels.
SS$_IVDEVNAM	The device name string contains invalid characters, or neither the **devnam** nor **chan** argument was specified.
SS$_IVLOGNAM	The device name string has a length of 0 or has more than 63 characters.
SS$_NONLOCAL	The device is on a remote system.
SS$_NOPRIV	The specified channel is not assigned or was assigned from a more privileged access mode.
SS$_NOSUCHDEV	The specified device does not exist on the host system.

Condition Values Returned in the I/O Status Block

Same as those returned in R0.

$GETDVIW
Get Device/Volume Information and Wait

The Get Device/Volume Information and Wait service returns information about an I/O device; this information consists of primary and secondary device characteristics.

The $GETDVIW service completes synchronously; that is, it returns to the caller with the requested information. Digital 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]

$GETJPI
Get Job/Process Information

Returns information about one or more processes on the system or across the VMScluster system.

The $GETJPI service completes asynchronously. For synchronous completion, use the Get Job/Process Information and Wait ($GETJPIW) service.

Format

SYS$GETJPI [efn] ,[pidadr] ,[prcnam] ,itmlst ,[iosb] ,[astadr] ,[astprm]

Arguments

efn

OpenVMS usage:	ef_number
type:	longword (unsigned)
access:	read only
mechanism:	by value

Number of the event flag to be set when $GETJPI returns the requested information. The **efn** argument is a longword containing this number; however, $GETJPI uses only the low-order byte.

Upon request initiation, $GETJPI clears the specified event flag (or event flag 0 if **efn** was not specified). Then, when $GETJPI returns the requested information, it sets the specified event flag (or event flag 0).

pidadr

OpenVMS usage:	process_id
type:	longword (unsigned)
access:	modify
mechanism:	by reference

Process identification (PID) of the process about which $GETJPI is to return information. The **pidadr** argument is the address of a longword containing the PID. The **pidadr** argument can refer to a process running on the local node or a process running on another node in the cluster.

If you give **pidadr** the value –1, $GETJPI assumes a wildcard operation and returns the requested information for each process on the system that it has the privilege to access, one process per call. To perform a wildcard operation, you must call $GETJPI in a loop, testing for the condition value SS$_NOMOREPROC after each call and exiting from the loop when SS$_NOMOREPROC is returned.

If you use $GETJPI with $PROCESS_SCAN you can perform wildcard searches across the cluster. In addition, with $PROCESS_SCAN you can search for specific processes based on many different selection criteria.

You cannot abbreviate a PID. All significant digits of a PID must be specified; only leading zeros can be omitted.

prcnam

OpenVMS usage:	process_name
type:	character-coded text string
access:	read only
mechanism:	by descriptor–fixed-length string descriptor

Name of the process about which $GETJPI is to return information. The **prcnam** argument is the address of a character string descriptor pointing to this name string.

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.

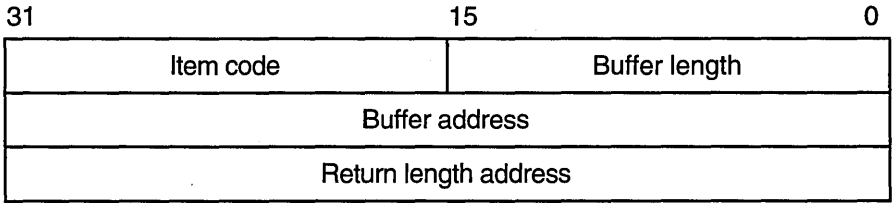
A local process name can look like a remote process name. Therefore, if you specify ATHENS::SMITH, the system checks for a process named ATHENS::SMITH on the local node before checking node ATHENS for a process named SMITH.

You may use the **prcnam** argument only if the process identified by **prcnam** has the same UIC group number as the calling process. If the process has a different group number, $GETJPI returns no information. To obtain information about processes in other groups, you must use the **pidadr** argument.

itmlst

OpenVMS usage:	item_list_3
type:	longword (unsigned)
access:	read only
mechanism:	by reference

Item list specifying which information about the process or processes is to be returned. The **itmlst** argument is the address of a list of item descriptors, each of which describes an item of information. The list of item descriptors is terminated by a longword of 0. 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 $GETJPI is to write the information. The length of the buffer needed depends upon the item code specified in the item code field of the item descriptor. If the value of buffer length is too small, $GETJPI truncates the data.
Item code	A word containing a user-supplied symbolic code specifying the item of information that $GETJPI is to return. The $JPIDEF macro defines these codes. Each item code is described in the Item Codes section.
Buffer address	A longword containing the user-supplied address of the buffer in which $GETJPI is to write the information.
Return length address	A longword containing the user-supplied address of a word in which $GETJPI writes the length (in bytes) of the information it actually returned.

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, $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, Digital strongly recommends that you specify it, for the following reasons:

- If you are using an event flag to signal the completion of the service, you can test the I/O status block for a condition value to be sure that the event flag was not set by an event other than service completion.
- If you are using the $SYNCH service to synchronize completion of the service, the I/O status block is a required argument for $SYNCH.
- The condition value returned in R0 and the condition value returned in the I/O status block provide information about different aspects of the call to the $GETJPI service. The condition value returned in R0 gives you information about the success or failure of the service call itself; the condition value returned in the I/O status block gives you information about the success or failure of the service operation. Therefore, to accurately assess the success or failure of the call to $GETJPI, you must check the condition values returned in both R0 and the I/O status block.

astadr

OpenVMS usage:	ast_procedure
type:	procedure value
access:	call without stack unwinding
mechanism:	by reference

AST service routine to be executed when $GETJPI 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 $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$_ACCOUNT
When you specify JPI$_ACCOUNT, $GETJPI returns the account name of the process, which is an 8-byte string, filled with trailing blanks if necessary.

JPI$_APTCNT
When you specify JPI$_APTCNT, $GETJPI returns, 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$_ASTACT
When you specify JPI$_ASTACT, $GETJPI returns the names of the access modes having active ASTs. This information is returned in a longword bit vector. When bit 0 is set, an active kernel mode AST exists; bit 1, an executive mode AST; bit 2, a supervisor mode AST; and bit 3, a user mode AST.

JPI$_ASTCNT
When you specify JPI$_ASTCNT, $GETJPI returns a count of the remaining AST quota, which is a longword integer value.

JPI$_ASTEN
When you specify JPI$_ASTEN, $GETJPI returns the names of the access modes having ASTs enabled. This information is returned in a longword bit vector. When bit 0 is set, kernel mode has ASTs enabled; bit 1, executive mode; bit 2, supervisor mode; and bit 3, user mode.

JPI$_ASTLM
When you specify JPI$_ASTLM, $GETJPI returns the AST limit quota of the process, which is a longword integer value.

JPI$_AUTHPRI
When you specify JPI$_AUTHPRI, $GETJPI returns the authorized base priority of the process, which is a longword integer value. The authorized base priority is the highest priority a process without ALTPRI privilege can attain by means of the $SETPRI service.

JPI$_AUTHPRIV
When you specify JPI$_AUTHPRIV, $GETJPI returns the privileges that the process is authorized to enable. These privileges are returned in a quadword privilege mask and are defined by the $PRVDEF macro.

JPI$_BIOCNT
When you specify JPI$_BIOCNT, $GETJPI returns a count of the remaining buffered I/O quota, which is a longword integer value.

JPI$_BIOLM
When you specify JPI$_BIOLM, $GETJPI returns the buffered I/O limit quota of the process, which is a longword integer value.

JPI$_BUFIO
When you specify JPI$_BUFIO, $GETJPI returns a count of the buffered I/O operations of the process, which is a longword integer value.

JPI$_BYTCNT
When you specify JPI$_BYTCNT, $GETJPI returns the remaining buffered I/O byte count quota of the process, which is a longword integer value.

JPI$_BYTLM
When you specify JPI$_BYTLM, $GETJPI returns the buffered I/O byte count limit quota of the process, which is a longword integer value.

JPI$_CHAIN
When you specify JPI$_CHAIN, $GETJPI processes another item list immediately after processing the current one. The buffer address field in the item descriptor specifies the address of the next item list to be processed. You must specify the JPI$_CHAIN item code last in the item list.

JPI$_CLINAME
When you specify JPI$_CLINAME, $GETJPI returns the name of the command language interpreter that the process is currently using. Because the CLI name can include up to 39 characters, the buffer length field in the item descriptor should specify 39 bytes.

JPI$_CPU_ID
When you specify JPI$_CPU_ID, $GETJPI 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$_CPULIM
When you specify JPI$_CPULIM, $GETJPI returns the CPU time limit of the process, which is a longword integer value.

JPI$_CPUTIM
When you specify JPI$_CPUTIM, $GETJPI returns the process's accumulated CPU time in 10-millisecond ticks, which is a longword integer value.

JPI$_CREPRC_FLAGS
When you specify JPI$_CREPRC_FLAGS, $GETJPI returns the flags specified by the **stsflg** argument in the $CREPRC call that created the process. The flags are returned as a longword bit vector.

JPI$_CURPRIV
When you specify JPI$_CURPRIV, $GETJPI returns the current privileges of the process. These privileges are returned in a quadword privilege mask and are defined by the $PRVDEF macro.

JPI$_DFMBC
When you specify JPI$_DFMBC, $GETJPI returns the default multibuffer count for a process as a longword integer value.

JPI$_DFPFC
When you specify JPI$_DFPFC, $GETJPI 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$_DFWSCNT
When you specify JPI$_DFWSCNT, $GETJPI 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$_DIOCNT
When you specify JPI$_DIOCNT, $GETJPI returns the remaining direct I/O quota of the process, which is a longword integer value.

JPI$_DIOLM
When you specify JPI$_DIOLM, $GETJPI returns the direct I/O quota limit of the process, which is a longword integer value.

JPI$_DIRIO
When you specify JPI$_DIRIO, $GETJPI returns a count of the direct I/O operations of the process, which is a longword integer value.

JPI$_EFCS
When you specify JPI$_EFCS, $GETJPI returns the state of the process's local event flags 0 through 31 as a longword bit vector.

JPI$_EFCU
When you specify JPI$_EFCU, $GETJPI returns the state of the process's local event flags 32 through 63 as a longword bit vector.

JPI$_EFWM
When you specify JPI$_EFWM, $GETJPI returns the event flag wait mask of the process, which is a longword bit vector.

JPI$_ENQCNT
When you specify JPI$_ENQCNT, $GETJPI returns the remaining lock request quota of the process, which is a longword integer value.

JPI$_ENQLM
When you specify JPI$_ENQLM, $GETJPI returns the lock request quota of the process, which is a longword integer value.

JPI$_EXCVEC
When you specify JPI$_EXCVEC, $GETJPI returns the address of a list of exception vectors for the process. Each exception vector in the list is a longword. There are eight vectors in the list: these are, in order, a primary and a secondary

vector for kernel mode access, for executive mode access, for supervisor mode access, and for user mode access.

The $GETJPI service cannot return this information for any process other than the calling process; if you specify this item code and the process is not the calling process, $GETJPI returns the value 0 in the buffer.

JPI$_FAST_VP_SWITCH
When you specify JPI$_FAST_VP_SWITCH, $GETJPI 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$_FILCNT
When you specify JPI$_FILCNT, $GETJPI returns the remaining open file quota of the process, which is a longword integer value.

JPI$_FILLM
When you specify JPI$_FILLM, $GETJPI returns the open file limit quota of the process, which is a longword value.

JPI$_FINALEXC
When you specify JPI$_FINALEXC, $GETJPI returns the address of a list of final exception vectors for the process. Each exception vector in the list is a longword. There are four vectors in the list, one for each access mode, in this order: kernel, executive, supervisor, and user.

The $GETJPI service cannot return this information for any process other than the calling process; if you specify this item code and the process is not the calling process, $GETJPI returns the value 0 in the buffer.

JPI$_FREP0VA
When you specify JPI$_FREP0VA, $GETJPI returns the address of the first free page at the end of the program region (P0 space) of the process.

JPI$_FREP1VA
When you specify JPI$_FREP1VA, $GETJPI returns the address of the first free page at the end of the control region (P1 space) of the process.

JPI$_FREPTECNT
When you specify JPI$_FREPTECNT, $GETJPI returns the number of pages (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, $GETJPI may be unable to retrieve all the data requested in an item list because JPI$_GETJPI_CONTROL_FLAGS requests that $GETJPI not perform certain

actions that may be necessary to collect the data. For example, a $GETJPI control flag may 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.

Since the JPI$_GETJPI_CONTROL_FLAGS item code tells $GETJPI how to interpret the item list, it must be the first entry in the $GETJPI item list. The error code SS$_BADPARAM is returned if it is not the first item in the list.

The JPI$_GETJPI_CONTROL_FLAGS item code includes the following flags.

Flag	Description
JPI$M_NO_TARGET_ INSWAP	Does not retrieve a process that has been swapped out of the balance set. This control flag is used to avoid adding the load of swapping processes into a system. By using this control flag and requesting information from a process that has been swapped out, the following occurs: • Any data stored in the virtual address space of the process is not accessible. • Any data stored in the process header (PHD) may not be accessible. • Any data stored in resident data structures, such as the process control block (PCB) or the job information block (JIB), is accessible. You must examine the return length of an item to verify that the item was retrieved.

Flag	Description
JPI$M_NO_TARGET_AST	Does not deliver a kernel mode AST to the target process. This control flag is used to avoid executing a target process to retrieve information. By using this control flag and not delivering an AST to a target process, the following occurs: • Any data stored in the virtual address space of the process is not accessible. • Any data stored in system data structures, such as the process header (PHD), the process control block (PCB), or the job information block (JIB), is accessible. You must examine the return length of an item to verify that the item was retrieved. The use of this control flag also implies that $GETJPI does not swap in a process, because $GETJPI would only bring a process into memory to deliver an AST to that process.
JPI$M_IGNORE_TARGET_STATUS	Attempts to retrieve as much information as possible, even though the process might be suspended or is being deleted. This control flag is used to retrieve all possible information from a process.

JPI$_GPGCNT
When you specify JPI$_GPGCNT, $GETJPI 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$_GRP
When you specify JPI$_GRP, $GETJPI returns, as a longword integer value, the group number of the process's UIC.

JPI$_IMAGECOUNT
When you specify JPI$_IMAGECOUNT, $GETJPI returns, as a longword integer value, the number of images that have been run down for the process.

JPI$_IMAGE_RIGHTS
When you specify JPI$_IMAGE_RIGHTS, $GETJPI 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 SYS1–9. The image rights list is a set of identifiers associated with a protected subsystem image. When a process runs a protected subsystem, the subsystem rights are automatically added to the process's image rights list. These identifiers are subsequently removed during image rundown. Allocate a buffer that is sufficient to hold the image rights list, because $GETJPI returns only as much of the list as will fit in the buffer.

Table SYS1–9 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$_IMAGNAME
When you specify JPI$_IMAGNAME, $GETJPI returns, as a character string, the directory specification and the image file name.

JPI$_IMAGPRIV
When you specify JPI$_IMAGPRIV, $GETJPI returns a quadword mask of the privileges with which the current image was installed. If the current image was not installed, $GETJPI returns the value 0 in the buffer.

JPI$_JOBPRCCNT
When you specify JPI$_JOBPRCCNT, $GETJPI returns the total number of subprocesses owned by the job, which is a longword integer value.

JPI$_JOBTYPE
When you specify JPI$_JOBTYPE, $GETJPI returns the execution mode of the process at the root of the job tree, which is a longword integer value. The symbolic name and value for each execution mode are listed in the following table. The $JPIDEF macro defines the symbolic names.

Mode Name	Value
JPI$K_DETACHED	0
JPI$K_NETWORK	1
JPI$K_BATCH	2
JPI$K_LOCAL	3
JPI$K_DIALUP	4
JPI$K_REMOTE	5

JPI$_LAST_LOGIN_I
When you specify JPI$_LAST_LOGIN_I, $GETJPI 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$_LAST_LOGIN_N
When you specify JPI$_LAST_LOGIN_N, $GETJPI 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$_LOGIN_FAILURES
When you specify JPI$_LOGIN_FAILURES, $GETJPI 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$_LOGIN_FLAGS
When you specify JPI$_LOGIN_FLAGS, $GETJPI returns a longword bitmask 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$_LOGINTIM
When you specify JPI$_LOGINTIM, $GETJPI returns the time at which the process was created, which is a standard 64-bit absolute time.

JPI$_MASTER_PID
When you specify JPI$_MASTER_PID, $GETJPI returns the process identification (PID) of the master process in the job. The PID is a longword hexadecimal value.

JPI$_MAXDETACH
When you specify JPI$_MAXDETACH, $GETJPI returns the maximum number of detached processes allowed for the user who owns the process specified in the call to $GETJPI. This limit is set in the UAF record of the user. The number is returned as a word decimal value. A value of 0 means that there is no limit on the number of detached processes for that user name.

JPI$_MAXJOBS
When you specify JPI$_MAXJOBS, $GETJPI returns the maximum number of active processes allowed for the user who owns the process specified in the call to $GETJPI. This limit is set in the UAF record of the user. The number is returned as a word decimal value. A value of 0 means that there is no limit on the number of active processes for that user name.

JPI$_MEM
When you specify JPI$_MEM, $GETJPI returns the member number of the process's UIC, which is a longword integer value.

JPI$_MODE
When you specify JPI$_MODE, $GETJPI returns the mode of the process, which is a longword integer value. The symbolic name and value for each mode are listed in the following table; the $JPIDEF macro defines the symbolic names.

Mode Name	Value
JPI$K_OTHER	0
JPI$K_NETWORK	1
JPI$K_BATCH	2
JPI$K_INTERACTIVE	3

JPI$_MSGMASK
When you specify JPI$_MSGMASK, $GETJPI returns the default message mask of the process, which is a longword bit mask.

JPI$_NODENAME
When you specify JPI$_NODENAME, $GETJPI returns, as a character string, the name of the VMScluster node on which the process is running.

JPI$_NODE_CSID
When you specify JPI$_NODE_CSID, $GETJPI returns, as a longword hexadecimal integer, the cluster ID of the VMScluster node on which the process is running.

JPI$_NODE_VERSION
When you specify JPI$_NODE_VERSION, $GETJPI returns, as a character string, the operating system version number of the VMScluster node on which the process is running.

JPI$_OWNER
When you specify JPI$_OWNER, $GETJPI returns the process identification (PID) of the process that created the specified process. The PID is a longword hexadecimal value.

JPI$_PAGEFLTS
When you specify JPI$_PAGEFLTS, $GETJPI returns the total number of page faults incurred by the process. This is a longword integer value.

JPI$_PAGFILCNT
When you specify JPI$_PAGFILCNT, $GETJPI returns the remaining paging file quota of the process, which is a longword integer value, measured in pages (on VAX systems) or pagelets (on Alpha systems).

JPI$_PAGFILLOC
When you specify JPI$_PAGFILLOC, $GETJPI returns the current paging file assignment of the process. The fourth byte of the returned longword value is the index of the system page file to which the process is currently assigned.

JPI$_PGFLQUOTA
When you specify JPI$_PGFLQUOTA, $GETJPI returns the paging file quota (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$_PHDFLAGS
When you specify JPI$_PHDFLAGS, $GETJPI returns the process header flags as a longword bit vector.

JPI$_PID
When you specify JPI$_PID, $GETJPI returns the process identification (PID) of the process. The PID is a longword hexadecimal value.

Alpha

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).♦

Alpha

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).♦

Alpha

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).♦

JPI$_PPGCNT
When you specify JPI$_PPGCNT, $GETJPI 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$_PRCCNT
When you specify JPI$_PRCCNT, $GETJPI returns, as a longword integer value, the number of subprocesses created by the process. The number returned by JPI$_PRCCNT does not include any subprocesses created by subprocesses of the process named in the **procnam** argument.

JPI$_PRCLM
When you specify JPI$_PRCLM, $GETJPI returns the subprocess quota of the process, which is a longword integer value.

JPI$_PRCNAM
When you specify JPI$_PRCNAM, $GETJPI returns, as a character string, the name of the process. Because the process name can include up to 15 characters, the buffer length field of the item descriptor should specify at least 15 bytes.

JPI$_PRI
When you specify JPI$_PRI, $GETJPI returns the current priority of the process, which is a longword integer value.

JPI$_PRIB
When you specify JPI$_PRIB, $GETJPI returns the base priority of the process, which is a longword integer value.

JPI$_PROCESS_RIGHTS
When you specify JPI$_PROCESS_RIGHTS, $GETJPI 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 SYS1–9. 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$_PROC_INDEX
When you specify JPI$_PROC_INDEX, $GETJPI returns, as a longword integer value, the process index number of the process. The process index number is a number between 1 and the SYSGEN parameter MAlphaROCESSCNT, 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$_PROCPRIV
When you specify JPI$_PROCPRIV, $GETJPI returns the default privileges of the process in a quadword bit mask.

JPI$_RIGHTSLIST
When you specify JPI$_RIGHTSLIST, $GETJPI returns, as an array of quadword identifiers, all identifiers applicable to the process. This includes the process rights list (JPI$_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 SYS1–9. 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$_RIGHTS_SIZE
When you specify JPI$_RIGHTS_SIZE, $GETJPI returns the number of bytes required to buffer the rights list. The rights list includes both the system rights list and the process rights list. Because the space requirements for the rights list can change between the time you request the size of the rights list and the time you fetch the rights list with JPI$_RIGHTSLIST, you might want to allocate a buffer that is 10 percent larger than this item indicates.

Alpha

JPI$_SCHED_POLICY
On Alpha systems, when you specify JPI$_SCHED_POLICY, $GETJPI 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$_SHRFILLM
When you specify JPI$_SHRFILLM, $GETJPI returns the maximum number of open shared files allowed for the job to which the process specified in the call to $GETJPI belongs. This limit is set in the UAF record of the user who owns the process. The number is returned as a word decimal value. A value of 0 means that there is no limit on the number of open shared files for that job.

JPI$_SITESPEC
When you specify JPI$_SITESPEC, $GETJPI returns the per-process, site-specific longword, which is a longword integer value.

JPI$_SLOW_VP_SWITCH
When you specify JPI$_SLOW_VP_SWITCH, $GETJPI 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$_STATE
When you specify JPI$_STATE, $GETJPI returns the state of the process, which is a longword integer value. Each state has a symbolic representation. If the process is currently executing, its state is always SCH$K_CUR. The $STATEDEF macro defines the following symbols, which identify the various possible states.

State	Description
SCH$C_CEF	Common event flag wait
SCH$C_COM	Computable
SCH$C_COMO	Computable, out of balance set
SCH$C_CUR	Current process
SCH$C_COLPG	Collided page wait
SCH$C_FPG	Free page wait
SCH$C_HIB	Hibernate wait
SCH$C_HIBO	Hibernate wait, out of balance set
SCH$C_LEF	Local event flag wait
SCH$C_LEFO	Local event flag wait, out of balance set
SCH$C_MWAIT	Mutex and miscellaneous resource wait
SCH$C_PFW	Page fault wait
SCH$C_SUSP	Suspended
SCH$C_SUSPO	Suspended, out of balance set

JPI$_STS
When you specify JPI$_STS, $GETJPI returns the 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

Symbol	Description
PCB$V_BATCH	Process is a batch job
PCB$V_DELPEN	Delete pending
PCB$V_DISAWS	Disable automatic working set adjustment
PCB$V_FORCPEN	Force exit pending
PCB$V_HARDAFF	Process bound to a particular CPU
PCB$V_HIBER	Hibernate after initial image activate
PCB$V_INQUAN	Initial quantum in progress
PCB$V_INTER	Process is an interactive job
PCB$V_LOGIN	Log in without reading authorization file
PCB$V_NETWRK	Process is a network connect object
PCB$V_NOACNT	No accounting for process
PCB$V_NODELET	No delete
PCB$V_PHDRES	Process header resident
PCB$V_PREEMPTED	Kernel mode suspend has overridden supervisor mode suspend
PCB$V_PSWAPM	Process swap mode (1=noswap)
PCB$V_PWRAST	Power fail AST
PCB$V_RECOVER	Process can recover locks
PCB$V_RES	Resident, in balance set
PCB$V_RESPEN	Resume pending, skip suspend
PCB$V_SECAUDIT	Mandatory security auditing
PCB$V_SOFTSUSP	Process is in supervisor mode suspend
PCB$V_SSFEXC	System service exception enable (kernel)
PCB$V_SSFEXCE	System service exception enable (exec)
PCB$V_SSFEXCS	System service exception enable (super)
PCB$V_SSFEXCU	System service exception enable (user)
PCB$V_SSRWAIT	System service resource wait disable
PCB$V_SUSPEN	Suspend pending
PCB$V_WAKEPEN	Wake pending, skip hibernate
PCB$V_WALL	Wait for all events in mask

JPI$_STS2
When you specify JPI$_STS2, $GETJPI 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
PCB$V_NOUNSHELVE	Process does not automatically unshelve files.

JPI$_SWPFILLOC
When you specify JPI$_SWPFILLOC, $GETJPI returns the location of the process's swapping file, which is a longword hexadecimal value. If the number returned is positive, the fourth byte of this value identifies a specific swapping file, and the lower three bytes contain the VBN within the swapping file. If

the number returned is 0 or negative, the swap file location information is not currently available for the process.

JPI$_SYSTEM_RIGHTS
When you specify JPI$_SYSTEM_RIGHTS, $GETJPI 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 SYS1–9. Allocate a buffer that is sufficient to hold the system rights list because $GETJPI only returns as much of the list as will fit in the buffer.

JPI$_TABLENAME
When you specify JPI$_TABLENAME, $GETJPI returns the file specification of the process's current command language interpreter (CLI) table. Because the file specification can include up to 255 characters, the buffer length field in the item descriptor should specify 255 bytes.

JPI$_TERMINAL
When you specify JPI$_TERMINAL, $GETJPI returns, for interactive users, the process's login terminal name as a character string. Because the terminal name can include up to 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$_TMBU
When you specify JPI$_TMBU, $GETJPI returns the termination mailbox unit number, which is a longword integer value.

JPI$_TQCNT
When you specify JPI$_TQCNT, $GETJPI returns the remaining timer queue entry quota of the process, which is a longword integer value.

JPI$_TQLM
When you specify JPI$_TQLM, $GETJPI returns the process's limit on timer queue entries, which is a longword integer value.

JPI$_TT_ACCPORNAM
When you specify JPI$_TT_ACCPORNAM, $GETJPI returns the access port name for the terminal associated with the process. (The terminal name is returned by JPI$_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$_TT_PHYDEVNAM
When you specify JPI$_TT_PHYDEVNAM, $GETJPI returns the physical device name of the terminal associated with the process. This name is the same as JPI$_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$_UAF_FLAGS
When you specify JPI$_UAF_FLAGS, $GETJPI returns the UAF flags from the UAF record of the user who owns the process. The flags are returned as a longword bit vector. For a list of the symbolic names of these flags, see the UAI$_FLAGS item code under the $GETUAI system service.

JPI$_UIC
When you specify JPI$_UIC, $GETJPI returns the UIC of the process in the standard longword format.

JPI$_USERNAME
When you specify JPI$_USERNAME, $GETJPI returns the user name of the process as a 12-byte string. If the name is less than 12 bytes, $GETJPI fills out the 12 bytes with trailing blanks and always returns 12 as the string length.

JPI$_VIRTPEAK
When you specify JPI$_VIRTPEAK, $GETJPI 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$_VOLUMES
When you specify JPI$_VOLUMES, $GETJPI returns the number of volumes that the process currently has mounted, which is a longword integer value.

JPI$_VP_CONSUMER
When you specify JPI$_VP_CONSUMER, $GETJPI returns a byte, the low-order bit of which, when set, indicates that the process is a vector consumer.

JPI$_VP_CPUTIM
When you specify JPI$_VP_CPUTIM, $GETJPI returns an unsigned longword that contains the total amount of time the process has accumulated as a vector consumer.

JPI$_WSAUTH
When you specify JPI$_WSAUTH, $GETJPI 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$_WSAUTHEXT
When you specify JPI$_WSAUTHEXT, $GETJPI 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$_WSEXTENT
When you specify JPI$_WSEXTENT, $GETJPI 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$_WSPEAK
When you specify JPI$_WSPEAK, $GETJPI 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

When you specify JPI$_WSQUOTA, $GETJPI 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

When you specify JPI$_WSSIZE, $GETJPI 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 other process's virtual address space, and the process might have a lower priority or might be currently swapped out of the balance set. To allow your program to overlap other functions with the time needed to schedule the other process for execution or swap it into the balance set, $GETJPI returns immediately after it has queued its information-gathering request to the other process.

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

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$_BADPARAM	The item list contains an invalid identifier.
SS$_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, $GETJPI found no more processes.
SS$_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

```
#include <stdio.h>
#include <ssdef.h>
#include <stsdef.h>
#include <jpidef.h>
#include <descrip.h>
#include <starlet.h>

typedef struct {short   buflen,         /* Length of output buffer */
                        itmcode;        /* Item code */
                void    *buffer;        /* Buffer address */
                void    *retlen;        /* Return length address */
                } ITMLST;  /* Layout of item-list elements */

int     retpid;                         /* PID returned by $GETJPI */
char    username[16],                   /* Username for retpid */
        procname[16],                   /* Process name for retpid */
        imagename[80];                  /* Image running under retpid */

/* descriptors: */
$DESCRIPTOR(user_desc, username);
$DESCRIPTOR(proc_desc, procname);
$DESCRIPTOR(image_desc, imagename);

/* Initialize $GETJPI item list... */
ITMLST  item_list[5] = {
                {12, JPI$_USERNAME, username,  &user_desc.dsc$w_length},
                {15, JPI$_PRCNAM,   procname,  &proc_desc.dsc$w_length},
                {79, JPI$_IMAGNAME, imagename, &image_desc.dsc$w_length},
                { 4, JPI$_PID,      &retpid,   0},
                { 0, 0,             0,         0} };

main()
{
    int status,                 /* Status of system calls */
        count = 0,              /* Count of matching processes */
        pid = -1;               /* Wildcard PID for $GETJPI */

    /* initial wildcard $GETJPI... */
    status = sys$getjpiw(0, &pid, 0, item_list, 0, 0, 0);
```

```
    /* Loop for all processes on this node.... */
    while (status != SS$_NOMOREPROC)
    {
        if (status & STS$M_SUCCESS)
        {
            /* add string terminator to process name */
            procname[proc_desc.dsc$w_length] = '\0';

            /* print header if this is the first */
            if (count == 0)
                printf("   PID   Process name                    Image\n");

            /* count process and print process data... */
            count++;
            printf("%08X %-15s %s\n", retpid, procname, imagename);
        }
        /* Skip process if suspended or no privilege to see process */
        /* Return any other error */
        else if ((status != SS$_NOPRIV) && (status != SS$_SUSPENDED))
            return (status);

        /* Find next process */
        status = sys$getjpiw(0, &pid, 0, item_list, 0, 0, 0);
    }
    return (SS$_NORMAL);
}
```

This example shows a segment of a program used to obtain the PID, process name, and current image being executed for every process for which the caller has the privilege to obtain information.

$GETJPIW
Get Job/Process Information and Wait

The Get Job/Process Information and Wait service returns information about one or more processes on the system.

The $GETJPIW service completes synchronously; that is, it returns to the caller with the requested information. Digital 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.

Format

```
SYS$GETJPIW  [efn] ,[pidadr] ,[prcnam] ,itmlst ,[iosb] ,[astadr] ,[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]

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.

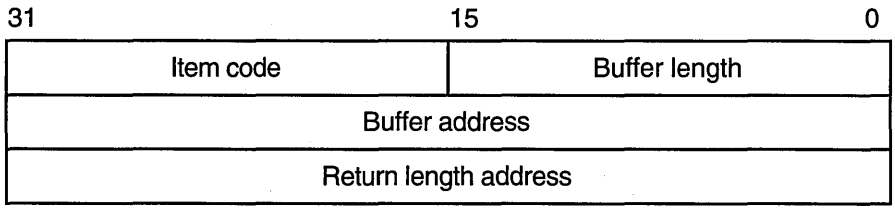
To use the $GETLKI service, you must have read/write access to the lock ID.

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 upon the item code specified in the item code field of the item descriptor. If the value of the buffer length field is too small, $GETLKI truncates the data and returns the success condition value SS$_NORMAL.
Item code	A word containing a user-specified symbolic code the item of information that $GETLKI is to return. The $LKIDEF macro defines these codes. Each item code is described in the list of $GETLKI item codes that follows the argument descriptions.
Buffer address	A longword containing a user-supplied address of the buffer in which $GETLKI is to write the information.
Return length address	A longword containing the user-supplied address of a longword in which $GETLKI writes return length information. This longword contains the following three bit fields. Bits 0 to 15 — In this field $GETLKI writes the length in bytes of the data actually written to the buffer specified by the buffer address field in the item descriptor.

Descriptor Field	Definition	
	Bits 16 to 30	$GETLKI uses this field only when the item code field of the item descriptor specifies LKI$_BLOCKEDBY, LKI$_BLOCKING, or LKI$_LOCKS, each of which requests information about a list of locks. $GETLKI writes in this field the length in bytes of the information returned for a single lock on the list. You can divide this length into the total length returned for all locks (bits 0 to 15) to determine the number of locks located by that item code request.
	Bit 31	$GETLKI sets this bit if the user-supplied buffer length argument specifies too small a buffer to contain the information returned. Note that in such a case $GETLKI will return the SS$_NORMAL condition value in R0. Therefore, to locate any faulty item descriptor, 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, Digital strongly recommends that you specify it, for the following reasons:

- If you are using an event flag to signal the completion of the service, you can test the I/O status block for a condition value to be sure that the event flag was not set by an event other than service completion.
- If you are using the $SYNCH service to synchronize completion of the service, the I/O status block is a required argument for $SYNCH.
- The condition value returned in R0 and the condition value returned in the I/O status block provide information about different aspects of the call to the $GETLKI service. The condition value returned in R0 gives you information about the success or failure of the service call itself; the condition value returned in the I/O status block gives you information about the success or failure of the service operation. Therefore, to accurately assess the success or failure of the call to $GETLKI, you must check the condition values returned in both R0 and the I/O status block.

astadr
OpenVMS usage: ast_procedure
type: procedure value
access: call without stack unwinding
mechanism: by reference

AST service routine to be executed when the service completes. The **astadr** argument is the address of this routine.

If you specify this argument, the AST routine executes at the same access mode as the caller of the $GETLKI service.

astprm
OpenVMS usage: user_arg
type: longword (unsigned)
access: read only
mechanism: by value

AST parameter to be passed to the AST service routine specified by the **astadr** argument. The **astprm** argument is the longword parameter.

nullarg
OpenVMS usage: null_arg
type: longword (unsigned)
access: read only
mechanism: by value

Placeholding argument reserved to Digital.

Item Codes

LKI$_BLOCKEDBY
When you specify LKI$_BLOCKEDBY, $GETLKI returns information about all locks that are currently blocked by the lock specified by **lkidadr**. The $GETLKI service returns eight items of information about each blocked lock.

The $LKIDEF macro defines the following symbolic names that refer to the eight items in the buffer.

Symbolic Name	Description
LKI$L_MSTLKID	Lock ID of the blocked lock on the system maintaining the resource (4 bytes)
LKI$L_PID	Process ID (PID) of the process that took out the blocked lock (4 bytes)
LKI$L_MSTCSID	VMScluster system identifier (CSID) of the node maintaining the resource that is locked by the blocked lock (4 bytes)
LKI$B_RQMODE	Lock mode requested for the blocked lock; this lock mode was specified by the **lkmode** argument in the call to $ENQ (1 byte)
LKI$B_GRMODE	Lock mode granted to the blocked lock; this lock mode is written to the lock value block (1 byte)

Symbolic Name	Description
LKI$B_QUEUE	Name of the queue on which the blocked lock currently resides (1 byte)
LKI$L_LKID	Lock ID of the lock on the system where the lock was requested (4 bytes)
LKI$L_CSID	VMScluster system identifier (CSID) of the system where the lock was requested (4 bytes)

The values that $GETLKI can write into the LKIB_RQMODE, LKIB_GRMODE, and LKI$B_QUEUE items have symbolic names; these symbolic names specify the six lock modes and the three types of queue in which a lock can reside. The Description section describes these names.

Thus, the buffer specified by the buffer address field in the item descriptor will contain the eight items of information, repeated in sequence, for each blocked lock.

The length of the information returned for each blocked lock is returned in bits 16 to 30 of the longword specified by the return length address field in the item descriptor, while the total length of information returned for all blocked locks is returned in bits 0 to 15. Therefore, to determine the number of blocked locks, you divide the value of bits 16 to 30 into the value of bits 0 to 15.

LKI$_BLOCKING

When you specify LKI$_BLOCKING, $GETLKI returns information about all locks that are currently blocking the lock specified by **lkidadr**. The $GETLKI service returns eight items of information about each blocking lock.

The $LKIDEF macro defines the following symbolic names that refer to the eight items in the buffer.

Symbolic Name	Description
LKI$L_MSTLKID	Lock ID of the blocked lock on the system maintaining the resource (4 bytes)
LKI$L_PID	Process ID (PID) of the process that took out the blocking lock (4 bytes)
LKI$L_MSTCSID	VMScluster system identifier (CSID) of the node maintaining the resource that is locked by the blocking lock (4 bytes)
LKI$B_RQMODE	Lock mode requested for the blocking lock; this lock mode was specified by the **lkmode** argument in the call to $ENQ (1 byte)
LKI$B_GRMODE	Lock mode granted to the blocking lock; this lock mode is written to the lock value block (1 byte)
LKI$B_QUEUE	Name of the queue on which the blocking lock currently resides (1 byte)
LKI$L_LKID	Lock ID of the lock on the system where the lock was requested (4 bytes)
LKI$L_CSID	VMScluster system identifier (CSID) of the system where the lock was requested (4 bytes)

The values that $GETLKI can write into the LKIB_RQMODE, LKIB_GRMODE, and LKI$B_QUEUE items have symbolic names; these symbolic names specify the six lock modes and the three types of queue in which a lock can reside. The Description section describes these names.

Thus, the buffer specified by the buffer address field in the item descriptor will contain the eight items of information, repeated in sequence, for each blocking lock.

The length of the information returned for each blocking lock is returned in bits 16 to 30 of the longword specified by the return length address field in the item descriptor, while the total length of information returned for all blocking locks is returned in bits 0 to 15. Therefore, to determine the number of blocking locks, you divide the value of bits 16 to 30 into the value of bits 0 to 15.

LKI$_CSID
When you specify LKI$_CSID, $GETLKI returns the Cluster System ID (CSID) of the system where the process owning the lock resides. LKI$_CSID returns the CSID of the node where the $GETLKI system service is issued when the resource is mastered on that node. When the processor is not part of a cluster, LKI$_CSID returns 0.

The buffer length field in the item descriptor should specify 4 (bytes).

LKI$_CVTCOUNT
When you specify LKI$_CVTCOUNT, $GETLKI returns the total number of locks that are currently on the conversion queue of the resource associated with the lock. These locks are granted at one mode and are waiting to be converted to another.

The buffer length field in the item descriptor should specify 4 (bytes).

LKI$_GRANTCOUNT
When you specify LKI$_GRANTCOUNT, $GETLKI returns the total number of locks that are currently on the grant queue of the resource associated with the lock. Note that the total number of granted locks on the resource is equal to the sum of LKI$_CVTCOUNT and LKI$_GRANTCOUNT.

The buffer length field in the item descriptor should specify 4 (bytes).

LKI$_LCKREFCNT
When you specify LKI$_LCKREFCNT, $GETLKI returns the number of locks that have this lock as a parent lock. When these locks were created, the **parid** argument in the call to $ENQ or $ENQW specified the lock ID of this lock.

The buffer length field in the item descriptor should specify 4 (bytes).

LKI$_LKID
When you specify LKI$_LKID, $GETLKI returns the lock ID of the lock on the system where the process owning the lock resides. The lock ID returned by this item code is meaningful only on the system specified in the value returned by the LKI$_CSID item code.

The buffer length field in the item descriptor should specify 4 (bytes).

LKI$_LOCKID
When you specify LKI$_LOCKID, $GETLKI returns the lock ID of the current lock. The current lock is the one specified by the **lkidadr** argument unless **lkidadr** is specified as –1 or 0, which indicates a wildcard operation. Thus, this

item code is usually specified only in wildcard operations where it is useful to know the lock IDs of the locks that $GETLKI has discovered in the wildcard operation.

The lock ID is a longword value, so the buffer length field in the item descriptor should specify 4 (bytes).

LKI$_LOCKS
When you specify LKI$_LOCKS, $GETLKI returns information about all locks on the resource associated with the lock specified by **lkidadr**.

The $LKIDEF macro defines the following symbolic names that refer to the eight items in the buffer.

Symbolic Name	Description
LKI$L_MSTLKID	Lock ID of the blocked lock on the system maintaining the resource (4 bytes)
LKI$L_PID	Process ID (PID) of the process that took out the lock (4 bytes)
LKI$L_MSTCSID	VMScluster system identifier (CSID) of the node maintaining the resource that is locked by the lock (4 bytes)
LKI$B_RQMODE	Lock mode requested for the lock; this lock mode was specified by the **lkmode** argument in the call to $ENQ (1 byte)
LKI$B_GRMODE	Lock mode granted to the lock; this lock mode is written to the lock value block (1 byte)
LKI$B_QUEUE	Name of the queue on which the lock currently resides (1 byte)
LKI$L_LKID	Lock ID of the lock on the system where the lock was requested (4 bytes)
LKI$L_CSID	VMScluster system identifier (CSID) of the system where the lock was requested (4 bytes)

The values that $GETLKI can write into the LKIB_RQMODE, LKIB_GRMODE, and LKI$B_QUEUE items have symbolic names; these symbolic names specify the six lock modes and the three types of queue in which a lock can reside. The Description section describes these names.

Thus, the buffer specified by the buffer address field in the item descriptor will contain the eight items of information, repeated in sequence, for each lock.

The length of the information returned for each lock is returned in bits 16 to 30 of the longword specified by the return length address field in the item descriptor, while the total length of information returned for all locks is returned in bits 0 to 15. Therefore, to determine the number of locks, you divide the value of bits 16 to 30 into the value of bits 0 to 15.

LKI$_MSTCSID
When you specify LKI$_MSTCSID, $GETLKI returns the Cluster System ID (CSID) of the node currently mastering the resource that is associated with the specified lock. Although the resource can be locked by processes on any node in the cluster, the resource itself is maintained on a single node. You can use the

DCL command SHOW CLUSTER or the $GETSYI service to determine which node in the VMScluster is identified by the CSID that $GETLKI returns.

Because the processor mastering the lock can change at any time, multiple calls to $GETLKI for the same lock can produce different values for this item code. LKI$_MSTCSID returns the CSID of the node where the $GETLKI system service is issued when the resource is mastered on that node. When the processor where the $GETLKI was issued is not part of a VMScluster, this item code returns 0.

The buffer length field in the item descriptor should specify 4 (bytes).

LKI$_MSTLKID

When you specify LKI$_MSTLKID, $GETLKI returns the lock ID for the current master copy of the lock. Although the resource can be locked by processes on any node in the cluster, the resource itself is maintained on a single node. Because lock IDs are unique to each processor on a cluster, the lock ID returned by this item code has meaning only on the processor that is specified in the value returned by the LKI$_MSTCSID item code.

Because the processor mastering the lock can change at any time, multiple calls to $GETLKI for the same lock can produce different values for this item code. When the lock is mastered on the node where the $GETLKI system service is issued, or when the node is not a member of a cluster, this item code returns the same information as LKI$_LKID.

The buffer length field in the item descriptor should specify 4 (bytes).

LKI$_NAMSPACE

When you specify LKI$_NAMSPACE, $GETLKI returns information about the resource name space. This information is contained in a longword consisting of four bit fields; therefore, the buffer length field in the item descriptor should specify 4 (bytes).

Each of the four bit fields can be referred to by its symbolic name; the $LKIDEF macro defines the symbolic names. The following table lists, in order, the symbolic name of each bit field.

Symbolic Name	Description
LKI$W_GROUP	In this field (bits 0 to 15) $GETLKI writes the UIC group number of the process that took out the first lock on the resource, thereby creating the resource name. This process issued a call to $ENQ or $ENQW specifying the name of the resource in the **resnam** argument. However, if this process specified the LCK$_SYSTEM flag in the call to $ENQ or $ENQW, the resource name is systemwide. In this case, the UIC group number of the process is not associated with the resource name. Consequently, this field (bits 0 to 15) is significant only if the resource name is not systemwide. $GETLKI sets bit 31 if the resource name is systemwide.

Symbolic Name	Description
LKI$B_RMOD	In this field (bits 16 to 23) $GETLKI writes the access mode associated with the first lock taken out on the resource.
LKI$B_STATUS	This field (bits 24 to 30) is not used. $GETLKI sets it to 0.
LKI$V_SYSNAM	This field (bit 31) indicates whether the resource name is systemwide. $GETLKI sets this bit if the resource name is systemwide and clears it if the resource name is qualified by the creating process's UIC group number. The state of this bit determines the interpretation of bits 0 to 15.

LKI$_PARENT
When you specify LKI$_PARENT, $GETLKI returns the lock ID of the parent lock for the lock, if a parent lock was specified in the call to $ENQ or $ENQW. If the lock does not have a parent lock, $GETLKI returns the value 0.

Because the parent lock ID is a longword, the buffer length field in the item descriptor should specify 4 (bytes).

LKI$_PID
When you specify LKI$_PID, $GETLKI returns the process identification (process ID) of the process that owns the lock.

The process ID is a longword value, so the buffer length field in the item descriptor should specify 4 (bytes).

LKI$_RESNAM
When you specify LKI$_RESNAM, $GETLKI returns the resource name string and its length, which must be from 1 to 31 bytes. The resource name string was specified in the **resnam** argument in the initial call to $ENQ or $ENQW.

The $GETLKI service returns the length of the string in the return length address field in the item descriptor. However, in the call to $GETLKI, you do not know how long the string is. Therefore, to avoid buffer overflow, you should specify the maximum length (31 bytes) in the buffer length field in the item descriptor.

LKI$_RSBREFCNT
When you specify LKI$_RSBREFCNT, $GETLKI returns the number of subresources of the resource associated with the lock. A subresource has the resource as a parent resource. Note, however, that the number of subresources can differ from the number of sublocks of the lock, because any number of processes can lock the resource. If any of these processes then locks another resource, and in doing so specifies the lock ID of the lock on the first resource as a parent lock, then the second resource becomes a subresource of the first resource.

Thus, the number of sublocks on a lock is limited to the number of sublocks that a single process takes out, whereas the number of subresources on a resource is determined by (potentially) multiple processes.

The subresource reference count is a longword value, so the buffer length field in the item descriptor should specify 4 (bytes).

LKI$_STATE
When you specify LKI$_STATE, $GETLKI returns the current state of the lock. The current state of the lock is described by the following three 1-byte items (in the order specified): (1) the lock mode requested (in the call to $ENQ or $ENQW)

for the lock, (2) the lock mode granted (by $ENQ or $ENQW) for the lock, and (3) the name of the queue on which the lock currently resides.

The **buffer length** field in the item descriptor should specify 3 (bytes). The $LKIDEF macro defines the following symbolic names that refer to the three 1-byte items in the buffer.

Symbolic Name	Description
LKI$B_STATE_RQMODE	Lock mode requested
LKI$B_STATE_GRMODE	Lock mode granted
LKI$B_STATE_QUEUE	Name of queue on which the lock resides

The values that $GETLKI can write into each 1-byte item have symbolic names; these symbolic names specify the six lock modes and the three types of queue in which a lock can reside. The Description section describes these names.

LKI$_VALBLK

When you specify LKI$_VALBLK, $GETLKI returns the lock value block of the locked resource. This lock value block is the master copy that the lock manager maintains for the resource, not the process-private copy.

Because the lock value block is 16 bytes, the buffer length field in the item descriptor should specify 16.

LKI$_WAITCOUNT

When you specify LKI$_WAITCOUNT, $GETLKI returns the total number of locks that are currently on the wait queue of the resource associated with the lock. These locks are waiting to be granted.

The buffer length field in the item descriptor should specify 4 (bytes).

Description

The Get Lock Information service returns information about the lock database on a system.

The access mode of the calling process must be equal to or more privileged than the access mode at which the lock was initially granted.

When locking on a resource is clusterwide, a single master copy of the resource is maintained on the node that owns the process that created the resource by taking out the first lock on it. When a process on another node locks that same resource, a local copy of the resource is copied to the node and the lock is identified by a lock ID that is unique to that node.

In a cluster environment, however, you cannot use $GETLKI to obtain directly information about locks on other nodes in the cluster; that is, you cannot specify in a call to $GETLKI the lock ID of a lock held by a process on another node. The $GETLKI service interprets the **lkidadr** argument as the lock ID of a lock on the caller's node, even though the resource associated with a lock might have its master copy on the caller's node.

However, because a process on another node in the cluster can have a lock on the same resource as the caller of $GETLKI, the caller, in obtaining information about the resource, can indirectly obtain some information about locks on the resource that are held by processes on other nodes. One example of information indirectly obtained about a resource is the contents of lock queues; these queues

contain information about all locks on the resource, and some of these locks can be held by processes on other nodes.

Another example of information more directly obtained is the remote lock ID of a lock held by a process on another node. Specifically, if the caller of $GETLKI on node A specifies a lock (by means of **lkidadr**) and that lock is held by a process on node B, $GETLKI will return the lock ID of the lock from node B's lock database if the LKI$_REMLKID item code is specified in the call.

Item codes LKI$_BLOCKEDBY, LKI$_BLOCKING, LKI$_LOCKS, and LKI$_STATE specify that $GETLKI return various items of information; some of these items are the names of lock modes or the names of lock queues. The $LCKDEF macro defines the following symbolic names.

Symbolic Name	Lock Mode
LCK$K_NLMODE	Null mode
LCK$K_CRMODE	Concurrent read mode
LCK$K_CWMODE	Concurrent write mode
LCK$K_PRMODE	Protected read mode
LCK$K_PWMODE	Protected write mode
LCK$K_EXMODE	Exclusive mode

Symbolic Name	Queue Name
LKI$C_GRANTED	Granted queue, holding locks that have been granted
LKI$C_CONVERT	Converting queue, holding locks that are currently being converted to another lock mode
LKI$C_WAITING	Waiting queue, holding locks that are neither granted nor converting (for example, a blocked lock)

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.

Required Quota

The caller must have sufficient ASTLM or BYTLM quota.

Related Services

$DEQ, $ENQ, $ENQW, $GETLKIW, $SET_RESOURCE_DOMAIN

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The item list cannot be read; the areas specified by the buffer address and return length address fields in the item descriptor cannot be written; or the location specified by the **lkidadr** argument cannot be written.
SS$_BADPARAM	You specified an invalid item code.
SS$_EXQUOTA	The caller has insufficient ASTLM or BYTLM quota.
SS$_INSFMEM	The nonpaged dynamic memory is insufficient for the operation.
SS$_IVLOCKID	The **lkidadr** argument specified an invalid lock ID.
SS$_IVMODE	A more privileged access mode is required.
SS$_NOMORELOCK	The caller requested a wildcard operation by specifying a value of 0 or –1 for the **lkidadr** argument, and $GETLKI has exhausted the locks about which it can return information to the caller; or no **lkidadr** argument is specified. This is an alternate success status.
SS$_NOSYSLCK	The caller attempted to acquire information about a systemwide lock and did not have the required SYSLCK privilege.
SS$_NOWORLD	The caller attempted to acquire information about a lock held by a process in another group and did not have the required WORLD privilege.

Condition Values Returned in the I/O Status Block

Same as those returned in R0.

$GETLKIW
Get Lock Information and Wait

The Get Lock Information and Wait service 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]

$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]

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-bit or 64-bit reference (Alpha) by 32-bit reference (VAX)

Length of the message string returned by $GETMSG. The **msglen** argument is the 32-bit 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-bit or 64-bit descriptor–fixed-length string descriptor (Alpha) by 32-bit descriptor–fixed-length string descriptor (VAX)

Buffer to receive the message string. The **bufadr** argument is the 32-bit or 64-bit address (on Alpha systems) or the 32-bit address (on VAX systems) of a character string descriptor pointing to the buffer into which $GETMSG writes the message string. The maximum size of any message string is 256 bytes.

flags

OpenVMS usage:	mask_longword
type:	longword (unsigned)
access:	read only
mechanism:	by value

Message components to be returned. The **flags** argument is a longword bit vector wherein a bit, when set, specifies that the message component is to be returned. The following table describes the significant bits.

Bit	Value	Description
0	1	Include text of message
	0	Do not include text of message
1	1	Include message identifier
	0	Do not include message identifier
2	1	Include severity indicator
	0	Do not include severity indicator
3	1	Include facility name
	0	Do not include facility name

If you omit this argument in a VAX MACRO or BLISS-32 service call, it defaults to a value of 15; that is, all flags are set and all components of the message are returned. If you omit this argument in a Fortran service call, it defaults to a value of 0; the value 0 causes $GETMSG to use the process default flags.

outadr

OpenVMS usage:	vector_byte_unsigned
type:	byte (unsigned)
access:	write only
mechanism:	by 32-bit or 64-bit reference (Alpha) by 32-bit reference (VAX)

Optional information to be returned by $GETMSG. The **outadr** argument is the 32-bit 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=xxxxxxxx[hex], (facility=n, message=n[dec])
   ```

Required Access or Privileges

None

Required Quota

None

Related Services

$ALLOC, $ASSIGN, $BRKTHRU, $BRKTHRUW, $CANCEL, $CREMBX, $DALLOC, $DASSGN, $DELMBX, $DEVICE_SCAN, $DISMOU, $GETDVI, $GETDVIW, $GETQUI, $GETQUIW, $INIT_VOL, $MOUNT, $PUTMSG, $QIO, $QIOW, $SNDERR, $SNDJBC, $SNDJBCW, $SNDOPR

Condition Values Returned

SS$_NORMAL	The service completed successfully.
SS$_BUFFEROVF	The service completed successfully. The string returned overflowed the buffer provided and has been truncated.
SS$_INSFARG	The call arguments are insufficient.
SS$_MSGNOTFND	The service completed successfully; however, the message code cannot be found, and a default message has been returned.

Example

```
#include <stdio.h>
#include <ssdef.h>
#include <stsdef.h>
#include <descrip.h>
#include <starlet.h>

int     status,                         /* Status of system calls */
        msg_flag = 0x0001,              /* Text only */
        msg_code = SS$_DUPLNAM,         /* Message code to retrieve */
        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
```

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

T

U

V

W

NOTES