VSI OpenVMS

TCP/IP Administrator's Guide: Volume II

Document Number: DO-DVTIA2-01B

Publication Date: January 2020

This document describes how to administrate VSI TCP/IP for OpenVMS.

Revision Update Information: This guide supercedes the VSI TCP/IP Administrator's Guide: Volume II, Version 10.5.

Operating System and Version: VSI OpenVMS Version 8.4-2L1 or higher

Software Version: VSI TCP/IP for OpenVMS Version 10.6

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

TCP/IP Administrator's Guide: Volume II:

Preface

1. About VSI

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

VSI seeks to continue the legendary development prowess and customer-first priorities that are so closely associated with the OpenVMS operating system and its original author, Digital Equipment Corporation.

2. Intended Audience

This manual is intended for anyone who will be administering VSI TCP/IP. It provides an overview of VSI TCP/IP Version 10.5 and contains information about:

- RMT Server and client configuration
- FTP configuration and management
- Font Server configuration
- Remote Systems configuration with RARP, BOOTP, and DHCP Server
- XDM Server management and X11-Gateway configuration
- VSI TCP/IP SNMP services configuration
- NFS Client
- SecureShell (SSH) servers versions 1 & 2 configuration
- IPSEC and SETKEY configuration
- Intrusion Prevention System (IPS)
- DECnet-over-IP circuits configuration

The appendices in this document contain DNSSEC parameters and language support parameters for TN3270 and TN5250.

3. Typographical Conventions

The following conventions are used in this manual:

Convention	Meaning
Ctrl/*X*	A sequence such as **Ctrl/*x*** indicates that you must hold down the key labeled Ctrl while you press another key or a pointing device button.
PF1 *X*	A sequence such as **PF1 *X*** indicates that you must first press and release the key labeled PF1 and then press and release another key (***x***) or a pointing device button.
Enter	In examples, a key name in bold indicates that you press that key.

Convention	Meaning
`...`	A horizontal ellipsis in examples indicates one of the following possibilities:- Additional optional arguments in a statement have been omitted.- The preceding item or items can be repeated one or more times.- Additional parameters, values, or other information can be entered.
. . .	A vertical ellipsis indicates the omission of items from a code example or command format; the items are omitted because they are not important to the topic being discussed.
( )	In command format descriptions, parentheses indicate that you must enclose choices in parentheses if you specify more than one. In installation or upgrade examples, parentheses indicate the possible answers to a prompt, such as: `Is this correct? (Y/N) [Y]`
[ ]	In command format descriptions, brackets indicate optional choices. You can choose one or more items or no items. Do not type the brackets on the command line. However, you must include the brackets in the syntax for directory specifications and for a substring specification in an assignment statement. In installation or upgrade examples, brackets indicate the default answer to a prompt if you press **Enter** without entering a value, as in: `Is this correct? (Y/N) [Y]`
|	In command format descriptions, vertical bars separate choices within brackets or braces. Within brackets, the choices are optional; within braces, at least one choice is required. Do not type the vertical bars on the command line.
{ }	In command format descriptions, braces indicate required choices; you must choose at least one of the items listed. Do not type the braces on the command line.
bold type	Bold type represents the name of an argument, an attribute, or a reason. In command and script examples, bold indicates user input. Bold type also represents the introduction of a new term.
italic type	Italic type indicates important information, complete titles of manuals, or variables. Variables include information that varies in system output (Internal error *number*), in command lines (/PRODUCER=*name*), and in command parameters in text (where *dd* represents the predefined code for the device type).
UPPERCASE TYPE	Uppercase type indicates a command, the name of a routine, the name of a file, or the abbreviation for a system privilege.
`Example`	This typeface indicates code examples, command examples, and interactive screen displays. In text, this type also identifies website addresses, UNIX command and pathnames, PC-based commands and folders, and certain elements of the C programming language.
-	A hyphen at the end of a command format description, command line, or code line indicates that the command or statement continues on the following line.
numbers	All numbers in text are assumed to be decimal unless otherwise noted. Nondecimal radixes-binary, octal, or hexadecimal-are explicitly indicated.

4. VSI TCP/IP Support

VSI supports VSI TCP/IP running on VSI OpenVMS Integrity Version 8.4-2L1 (or higher) only. Please contact your support channel for help with this product. Users who have OpenVMS support contracts through VSI can contact support@vmssoftware.com [mailto:support@vmssoftware.com] for help with this product. Users who have OpenVMS support contracts through HPE should contact their HPE Support channel for assistance.

5. VSI Encourages Your Comments

You may send comments or suggestions regarding this manual or any VSI document by sending electronic mail to the following Internet address: `<docinfo@vmssoftware.com>`.

6. How to Order Additional Documentation

For information about how to order additional documentation, email the VSI OpenVMS information account: `<info@vmssoftware.com>`. We will be posting links to documentation on our corporate website soon.

Chapter 6. RMT Server and Client Configuration

This chapter explains how to configure the RMT (Remote Magnetic Tape) server, and how to use RMTALLOC (the RMT client) with tape drives and CD-ROM drives.

6.1. Configuring the Remote Magnetic Tape Server

The VSI TCP/IP remote magnetic tape server (RMT) uses the BSD RMT protocol to allow UNIX and PC users to access tape drives on OpenVMS systems. Most systems derived from BSD 4.2 and 4.3 support the **rdump** and **rrestore** commands for accessing tape drives served by RMT.

To enable and configure the OpenVMS RMT server:

1. Make sure RSHELL works from the UNIX Operating system user "root" to the OpenVMS user ROOT. If no OpenVMS user ROOT exists, the RMT server uses the OpenVMS user SYSTEM.

Note

For ROOT and SYSTEM, the system-wide `IP$:HOSTS.EQUIV` file is ignored and an explicit entry in `SYS$LOGIN:.RHOSTS` is required to grant access.

2. Make sure the **ROOT /SYSTEM LOGIN.COM** and the system-wide **SYLOGIN.COM** do not print anything when you issue remote RSHELL (under OpenVMS) command. Anything written to `SYS$OUTPUT` from these command procedures interferes with the RMT protocol.

The following example shows commands that prevent output from being displayed by **SYSTEM / ROOT LOGIN.COM** and **SYLOGIN.COM**.

```
$ VERIFY = 'F$VERIFY(0)                 ! Turn off verify without echoing
$ IF F$MODE() .EQS. "OTHER" THEN EXIT   ! If a DETACHED process (RSHELL)
$ IF VERIFY THEN SET VERIFY             ! If a batch job, may want to turn
                                          verify back on.
```

You can specify OpenVMS-style magtape device names or an OpenVMS file name for writing to a disk file.

When you specify UNIX-style names, options are encoded in the unit number (minor device number). The correspondence between the options and their associated unit numbers is as follows:

Device	Options				
mt0 through mt3	/NOMOUNT	/STREAM	/DENS=800	/REWIND	/NOUNLOAD
mt4 through mt7	/NOMOUNT	/STREAM	/DENS=800	/NOREWIND	/NOUNLOAD
mt8 through mt11	/NOMOUNT	/STREAM	/DENS=160	/REWIND	/NOUNLOAD

Device	Options				
mt12 through mt15	/NOMOUNT	/STREAM	/DENS=160	/NOREWIND	/NOUNLOAD
mt16 through mt19	/NOMOUNT	/STREAM	/DENS=625	/REWIND	/NOUNLOAD
mt20 through mt23	/NOMOUNT	/STREAM	/DENS=625	/NOREWIND	/NOUNLOAD
rmt0 through rmt3	/NOMOUNT	/NOSTREAM	/DENS=800	/REWIND	/NOUNLOAD
rmt4 through rmt7	/NOMOUNT	/NOSTREAM	/DENS=800	/NOREWIND	/NOUNLOAD
rmt8 through rmt11	/NOMOUNT	/NOSTREAM	/DENS=16	/REWIND	/NOUNLOAD
rmt12 through rmt15	/NOMOUNT	/NOSTREAM	/DENS=16	/NOREWIND	/NOUNLOAD
rmt16 through rmt19	/NOMOUNT	/NOSTREAM	/DENS=62	/REWIND	/NOUNLOAD
rmt20 through rmt23	/NOMOUNT	/NOSTREAM	/DENS=62	/NOREWIND	/NOUNLOAD

The OpenVMS tape drive name is chosen automatically as the first tape drive, or you can set it using the NET-CONFIG **SET DEFAULT-RMT-TAPE-DEVICE** command.

When you specify OpenVMS-style names, the options are encoded in qualifiers; the exact format is: **vms_node_name:volume_name[/qualifiers[...]]**

For example: **# rdump 0f abc.com:/dev/rmt8 /usr** \

or:

```
# rdump 0f abc.com:mua0:/nomount/nostream \
/dens=1600/nounload /USR
```

or:

```
# rdump 0f abc.com:mua0:xxx/nostream
/dens=1600/nounload \
/comment="Please mount volume XXX on drive mua0" /usr
```

Table 6.1 shows the qualifiers available tape drive names.

Table 6.1. RMT Server Tape Drive Name Qualifiers

Qualifiers	Description
`/BLOCKSIZE=`*`size`*	Block size at which to write the tape. Default: 65534 bytes.
`/DENSITY=`*`density`*	Specifies the density at which to write a tape. Default: current density.
`/[NO]REWIND`	Specifies whether to rewind the drive on close; ignored unless /**NOMOUNT** is specified. Default: **/REWIND**.
`/[NO]UNLOAD`	Specifies whether to unload the drive on close. Default: **/UNLOAD**.

Qualifiers	Description
`/COMMENT="string"`	Comment to display in the remote OPCOM message, either appended to or replacing the default text, depending on the resulting string length being less than the 78-character maximum. This message is the only opportunity to send a tape-specific message to the remote operator. (MOUNT **/COMMENT** strings are not passed to a remote system.) Because RMTALLOC will not complete until a tape has been loaded and the drive is online, use COMMENT to make sure the operator is aware of your request.
`/[NO]MOUNT`	Mounts the tape drive using the OpenVMS MOUNT service / **NOMOUNT** accesses the tape drive without mounting it. This qualifier is used for UNIX utilities which expect the tape drive to hold its current position (not rewind) if they close it. By not mounting it, the tape drive does not rewind when dismounted. Default: **/MOUNT**.
`/[NO]STREAMING`	Accesses the tape drive as a sequential device (a UNIX character device). **/NOSTREAMING** accesses the tape drive as a raw device (a UNIX block device). Default: **/STREAMING**.

6.2. About the RMT Client

The RMT client IP RMTALLOC is used for accessing tape or CD-ROM drives on remote hosts over TCP (using RSHELL). If restrictions apply where RSHELL does not work, or if RSHELL outputs spurious login messages or greetings, RMTALLOC does not work. RMTALLOC depends on an RMT server to function properly. RMTALLOC creates a pseudo device that appears as an OpenVMS physical device to the OpenVMS BACKUP, COPY, and other utilities. The pseudo device is named "RMT*x*:", *x* is the unit number. The actual tape or CD-ROM drive can be on another VSI TCP/IP system or on any host running the RMT server, such as those running the BSD or SunOS UNIX operating system.

For CD-ROM, RMTALLOC treats the drive as a file system, which speeds file access.

There are some limits to the types of tape devices you can access on other operating systems and the amount of control available. Because UNIX tapes and tape drivers cannot write variable-length blocks and do not allow skipping forward over records between read operations, they cannot be used with OpenVMS **BACKUP** or **COPY** commands.

6.2.1. Limitations of UNIX Devices and Software

The following list describes known limitations of common UNIX devices and software:

- UNIX QIC tape drives cannot be used.
- SunOS RMT servers require you to use **/UNIX_SERVER=BROKEN** so you can back up one single-volume **BACKUP** saveset. Use **BACKUP /REWIND** to copy to or from tape.
- ULTRIX RMT servers require you to use **/UNIX_SERVER=ULTRIX** to obtain full OpenVMS tape functionality.
- SGI's IRX RMT server (as of 4.0.5) does not interoperate with OpenVMS tape operations.
- IBM RS6000/AIX requires a special translation on the IBM side because IBM uses incompatible RMT commands.

6.3. Using RMTALLOC

To use RMTALLOC:

1. Make sure the media is in the drive and the drive is online. If you verify this, use the /**SEMANTICS=(COMMENT=*comment*)** qualifier to inform the remote OpenVMS operator of a pending device mount.
2. Allocate the drive with RMTALLOC.
3. Mount the tape or CD-ROM drive. Use the same **MOUNT** command you would for a OpenVMS device.
4. Read from or write to the tape, or read the information from the CD-ROM.
5. Dismount the tape or CD-ROM drive.
6. Deallocate the tape or CD-ROM drive.
7. Remove the media from the drive.

In its simplest form, you can specify RMTALLOC as follows:

```
$ IP RMTALLOChostname::devicename
```

For example, for a tape device, enter:

```
$ RMTALLOC CONE.IRIS.COM::MUA0: MYTAPE
```

For a CD-ROM drive, enter:

```
$ RMTALLOC /CD CONTROL::DISK$CD: MYCD /USER=SYSTEM
```

These examples are explained further in the following sections.

6.3.1. RMTALLOC Tape Drive Access Example

The following example shows how to allocate a tape drive and write to tape using the OpenVMS TAR utility:

```
$ RMTALLOC CONE.FLOWERS.COM::MUA0: MYTAPE
%RMT-I-ALLOC, _MYSYS$RMT1: allocated (CONE.FLOWERS.COM::MUA0:)

$ MOUNT /FOREIGN /RECORD_SIZE=512 /BLOCK_SIZE=10240 MYTAPE
%MOUNT-I-MOUNTED, MYTAPE mounted on _MYSYS$RMT1:

$ TAR /ARCHIVE=MYTAPE WRITE AFILE.TXT
%TAR-S-WRITTEN, written USERS:[ME]AFILE.TXT;1 (13495 bytes)
%TAR-S-TOTWRITE, total of 1 file written

$ DISMOUNT _MYSYS$RMT1:
$ DEALLOCATE _MYSYS$RMT1:
```

In this example, the host `CONE.FLOWERS.COM` contains the tape drive. The two colons separating the host name follow the style of DECnet device specifications; RMTALLOC accepts either single or double colon separators. MYTAPE is the tape logical name associated with the pseudo device. In the MOUNT statement, the **/FOREIGN** qualifier specifies that the device is not file structured.

6.3.2. RMTALLOC CD-ROM Access Examples

In the first example, a CD-ROM on an OpenVMS host is accessed from another OpenVMS host:

```
$ RMTALLOC/CD/NOWRITE CONTROL::DISK$CD: MYCD /USERNAME=SYSTEM
%RMT-I-ALLOC, _MYSYS$RCD3: allocated (CONTROL.FLOWERS.COM::DISK$CD:)
$ MOUNT/OVERRIDE=ID MYCD:
%MOUNT-I-WRITELOCK, volume is write locked
%MOUNT-I-MOUNTED, VMS055LST1 mounted on _MYSYS$RCD3:
```

Read from the CD-ROM drive, then dismount and deallocate the drive:

```
$ DISMOUNT MYCD:
$ DEALLOCATE MYCD:
```

In this example:

- The RMTALLOC statement includes the /**CD** qualifier to indicate the device is a CD-ROM drive.
- The /**NOWRITE** qualifier is the default whenever you specify /**CD**; omitting it indicates the device is read-only.
- CONTROL: is the host on which the CD-ROM is located.
- DISK$CD: is the drive name.
- MYCD is the device name.
- The /**USER=SYSTEM** qualifier ensures the SYSTEM account is accessed on the remote host. If the remote host is an OpenVMS system, the account used must have LOG_IO privilege.
- The /**MOUNT** command uses the /**OVERRIDE=ID** qualifier to inhibit **MOUNT** protection checks of the volume identifier in the CD-ROM label.
- The **DISMOUNT** and **DEALLOCATE** commands are used after information is read from the CD-ROM.

In the next example, a CD-ROM drive on a UNIX host is accessed:

```
$ RMTALLOC/CD/NOWRITE UNIXBOX::"/dev/rsr1" MYCD /USERNAME=root
%RMT-I-ALLOC, _MYSYS$RCD3: allocated (UNIXBOX.EXAMPLE.COM::/dev/rsr0)
```

```
$ MOUNT/OVERRIDE=ID MYCD:
%MOUNT-I-WRITELOCK, volume is write locked
%MOUNT-I-MOUNTED, VMS055LST2 mounted on _MYSYS$RCD3:
```

Read from the CD-ROM drive, then dismount and deallocate the drive:

```
$ DISMOUNT MYCD:
$ DEALLOCATE MYCD:
```

In this example, the RMTALLOC statement contains the name of the UNIX host (UNIXBOX) that has the CD-ROM drive. The device name is specified in UNIX style. If the device name is not specified, the default is the /dev/rsr0 device.

The user name is set to the root login, which is a UNIX login similar to the OpenVMS SYSTEM login. The **MOUNT**, **DISMOUNT**, and **DEALLOCATE** commands are the same whether the CD-ROM is on another OpenVMS system or a UNIX host.

6.4. Using RMTALLOC Qualifiers

RMTALLOC qualifiers, apart from those already discussed (**/CD**, **/NOWRITE**, and **/USERNAME**), provide the following additional features:

- VMS-to-VMS negotiation
- Remote operator interaction
- Remote login control
- Message suppression
- Write protection

6.4.1. VMS-to-VMS Negotiation

When accessing a drive on an OpenVMS host, if both systems are running VSI TCP/IP, RMT uses an improved protocol to transfer OpenVMS device attributes and I/O completion status values between your system and the remote host. Because this negotiation is compatible with UNIX Operating System implementations of RMT (including BSD and SunOS), it is enabled by default. You may disable it with the RMTALLOC **/NOVMS_ATTRIBUTES** qualifier if compatibility problems arise.

6.4.2. Suppressing Messages

Use the **/NOLOG** qualifier to suppress system status messages. Use this option in DCL command procedures to prevent the messages from displaying. **/LOG** is the default.

6.4.3. Controlling Remote Login

Use the **/PASSWORD** qualifier to specify a password for the remote host when you do not have a `.RHOSTS` file. This qualifier poses a security risk because the password is transmitted over the network as plain text. When you use **/PASSWORD**, the REXEC server (instead of the RSHELL server) is called on the remote host. The password is in the format used by the system you are contacting.

Similarly, use the **/USERNAME** qualifier to specify the login name to access on the remote system. On a UNIX system, the specified login must exist in the /etc/passwd file.

Use the **/TRUNCATE_USERNAME** qualifier to truncate an OpenVMS user name to a maximum of eight characters for use with some UNIX systems.

6.4.4. Interacting with the Remote Operator

Use the **/SEMANTICS** qualifier only with tape drive access to interact with the operator of the remote system or to specify tape drive information to the remote system.

Use the optional BLOCKSIZE and DENSITY values to specify information used by the remote system to read the tape. All other values send messages to the operator via the OPCOM message facility. Without specifying any values, the following information is displayed when RMTALLOC is called:

```
%%%%%%%%%%% OPCOM date time %%%%%%%%%%%
```

```
FROM NODE nodename AT date time
REQUEST nn, FROM USER username ON nodename
Please mount device _nodename$devicename:
RMT tape service request from nodename.domain
```

The **COMMENT** value is specified as a string enclosed in double-quotes; the information is displayed in the remote OPCOM message, either appended to or replacing the default text depending on whether the resulting length is less than the 78-character maximum. Supplying the **COMMENT** value is the only way you can send a tape-specific message to the remote operator. The OPCOM message from the DCL **MOUNT /COMMENT** command is not passed to the remote RMT server; this message is only sent to OPCOM for a local operation. The default RMTALLOC command mounts the remote tape foreign, causing an OPCOM message to be generated if the tape drive is offline.

Note

The RMTALLLOC **/SEMANTICS=NOMOUNT** command does not work correctly with multivolume **BACKUP** savesets.

6.4.5. Write Protection

Use the **/WRITE** qualifier to make sure that write protection is respected; **/NOWRITE** is the default for CD-ROM drives.

By default, RMTALLOC mounts a remote tape drive for read-write access. If the remote tape drive is physically protected from write access, you must use **/NOWRITE** to indicate you want read-only access to the tape drive. Otherwise, the remote UNIX RMT server usually returns an error indicating "Permission Denied."

Chapter 7. Configuring and Managing FTP

The FTP (File Transfer Protocol) server provides file access between remote systems. FTP is configured automatically during the VSI TCP/IP installation procedure. This chapter explains how to administer the FTP client and server.

7.1. Configuring the FTP Client

Configuring the FTP client consists of creating a `IP$:FTP.INIT` file for site-specific purposes. When the FTP client is started, the commands in the `IP$:FTP.INIT` file are executed before the commands in the `SYS$LOGIN:FTP.INIT` file of the user running FTP. See the *VSI TCP/IP User's Guide* for more information about creating `FTP.INIT` files.

The FTP client censors the output of the **NLST /LIST** commands. A period (.) replaces unprintable characters.

If the logical name `IP$FTP_DELAY_TRANSFER_NEGOTIATION` is defined, then the FTP client does not attempt to negotiate STRU O VMS transfer mode until after you have logged into the remote system successfully. You can define this logical at the user or system level.

```
$ DEFINE IP$FTP_DELAY_TRANSFER_NEGOTIATION anything
```

If the logical `IP$FTP_SIZE_BEFORE_GET` is defined to FALSE, NO, or 0 (zero) the **SIZE** command will not be sent before the GET command for a file. When the logical is not defined, or is defined to a value other than FALSE, NO, or 0, the **SIZE** command is sent. Any returned value is used to preallocate the file size and to report progress of a file transfer. Some FTP servers leave the file open accidentally after the **SIZE** command.

If the logical name `IP$FTP_NONPASV` is defined, then the FTP client will start up in PASSIVE OFF mode. The default client behavior is PASSIVE ON.

7.2. Managing an FTP Server

Managing an FTP server may include the following tasks:

- Creating an anonymous FTP login (see Section 7.2.1).
- Creating an FTP server login command procedure (see Section 7.2.3).
- Using log files (see Section 7.2.4).
- Managing FTP security (see Section 7.2.5).
- Specifying a message at connect time (see Section 7.2.7).
- Specifying UNIX-style listings (see Section 7.2.8).
- Specifying the maximum idle time before a connection times out (see Section 7.2.10).
- Using FTP server site commands (see Section 7.2.11).
- Using Network Service Monitoring (see Section 7.5).

- Using Session Accounting (see Section 7.6).

7.2.1. Configuring Anonymous FTP

To set up anonymous FTP access on your system:

1. Create an account named ANONYMOUS with the password GUEST. Any password works from the remote host but the account is validated with the password GUEST. Use the OpenVMS AUTHORIZE utility to create the account:

   ```
   $ RUN SYS$SYSTEM:AUTHORIZE
   UAF>ADD ANONYMOUS /PASSWORD=GUEST /OWNER="Anonymous FTP" -
   _UAF>/DEVICE=device /UIC=[uic]
   UAF> Ctrl/Z
   ```

   `uic` is the UIC to use for ANONYMOUS.
   `device` is the device on which the directory [ANONYMOUS] is located.

2. Use the NOLOCAL, NOBATCH, NOREMOTE, and NODIALUP access restrictions for the ANONYMOUS FTP user to prevent other forms of access. You set these restrictions by running AUTHORIZE and issuing the command:

   ```
   UAF>MODIFY ANONYMOUS /NOLOCAL /NOBATCH /NOREMOTE /NODIALUP
   ```

3. To prevent access through DECnet, do *not* grant the NETMBX privilege to the ANONYMOUS FTP user. To make sure that ANONYMOUS does not have the NETMBX privilege, issue the following AUTHORIZE command:

   ```
   UAF>MODIFY ANONYMOUS /PRIV=NONETMBX /DEFPRIV=NONETMBX
   ```

4. To restrict anonymous FTP access to the [ANONYMOUS] directory tree, use the NET-CONFIG **SET ANONYMOUS-FTP-DIRECTORY** command. To restrict the ANONYMOUS FTP user permissions to list, read, write, or delete files, use the NET-CONFIG **SET ANONYMOUS-FTP-ACCESS** command. See the *VSI TCP/IP Administrator's Reference* for additional information about NET-CONFIG commands.

Anonymous FTP server processes are created with the following process name:

`*FTP_pwd`

`pwd` is the password the user specifies.

By convention, many people specify GUEST, their personal name, or their local user name for the password, because anything is accepted.

If you do not want to create `FTP_SERVER.LOG` files in the anonymous directory, assign a new default directory for the login with a directory restriction to make sure the log files appear in the correct directory. In this example, an alternate FTP directory is created for the log files:

```
$ SET DEFAULT SYS$SYSTEM
$ RUN SYS$SYSTEM:AUTHORIZE
UAF>MODIFY ANONYMOUS/DEVICE=SYS$COMMON:/DIR=[SYSMGR.ANONYMOUS]
UAF>EXIT
$ CREATE/DIRECTORY/OWNER=ANONYMOUS SYS$COMMON:[SYSMGR.ANONYMOUS]
$ IP CONFIGURE
NET-CONFIG>SET ANONYMOUS-FTP-DIRECTORY USERS:[ANONYMOUS]
NET-CONFIG>EXIT
```

You can control the setting of the `IP$ANONYMOUS_FTP_CONTROL` logical name using the NET-CONFIG **SET ANONYMOUS-FTP-ACCESS** command. For example:

```
$ IP CONFIGURE
NET-CONFIG>SET ANONYMOUS-FTP-ACCESS NOSPAWN,NODELETE
NET-CONFIG>EXIT
[Changes take effect after the next VSI TCP/IP for OpenVMS reload]
```

You can make the changes take effect before the system reboot by defining the associated `IP$ANONYMOUS_FTP_CONTROL` logical name as follows:

```
$ DEFINE /SYSTEM /EXECUTIVE_MODE IP$ANONYMOUS_FTP_CONTROL -
_$ "NOSPAWN,NODELETE"
```

By default, the `NOWRITE`, `NOSPAWN` settings are used for anonymous FTP sessions. See Table 7.1 for detailed description of the access options.

If you do not want to use the "anonymous" name, there is a logical that will allow users to use names which are not anonymous, but have the same anonymous account behavior. The `IP$ANONYMOUS_USERNAMES` logical usage is shown in the following example:

```
$ DEFINE /SYSTEM /EXEC IP$ANONYMOUS_USERNAMES -
_$ "anonymous,user1,user2,..."
```

If you define this logical as shown in the preceding example and set the "user1,user2,…" accounts using the same password as the anonymous account, then the FTP server will treat "user1,user2,…" as an anonymous type of user.

7.2.2. Specifying a Range of FTP Server Port Numbers

The logical `IP$FTP_SERVER_DATA_PORT_RANGE` specifies the range of port numbers to use for passive connections.

```
$ DEFINE /SYSTEM /EXEC IP$FTP_SERVER_DATA_PORT_RANGE -
_$ "starting port number end port number"
```

When this logical is defined, the FTP server will use port numbers between the specified values for the data channel when operating in passive mode.

7.2.3. Creating an FTP Server Login Command Procedure

To limit user activities during an FTP session, edit the `FTP_SERVER.COM` file using this command:

```
$ IP FTP/SERVER[qualifier]
```

Table 7.1 lists the FTP server qualifiers.

Table 7.1. FTP Server Qualifiers

Qualifier	Purpose
/ACCESS=([NOLIST], [NOWRITE], [NOSPAWN], [NOREAD], [NODELETE])	Defines file access rights and permission to spawn a subprocess for the current FTP session. NOLIST disables the listing of files.

Qualifier	Purpose
	NOWRITE disables the storing of files. NOSPAWN disables the SPAWN command. NOREAD disables reading of files. NODELETE disables the deletion/renaming of files. By default, /ACCESS=(NOWRITE, NOSPAWN) is used for anonymous FTP sessions. **Note** To cancel a restriction, omit the appropriate option from the parameter list. By default, all access rights are granted if not listed.
/DIRECTORY=*(directory1,...)*	Restricts access to these directory trees.

/GET REMOTE INFO	Gets information about the remote system. This qualifier works by defining the logical names
	
	`IP$FTP_ADDRESS`
	`IP$FTP_HOSTNAME`
	`IP$FTP_LOCAL_ADDRESS`
	`IP$FTP_ANONYMOUS_PASSWORD`
	
	and then exiting without invoking the FTP server.
	
	Note
	
	`IP$FTP_ANONYMOUS_PASSWORD` is only set if the user name is "anonymous".
	
	When `IP$FTP_DONT_REPORT_FILESIZE` is defined, the estimate of the number of bytes to be transmitted is not included in the 150 reply line to a GET operation.

/MESSAGE=*message*	Displays a banner message when the user logs in. This message precedes the "User *xxx* logged in..." line.
/REJECT	Instead of accepting the connection, rejects the login with the error specified in the /MESSAGE qualifier.

7.2.4. Using FTP Log Files

The VSI TCP/IP FTP server keeps a log of all FTP transactions that occur after login between the client and server in an `FTP_SERVER.LOG` file in the user's login directory on the server system. A log file is created for each FTP client session. The previous log is overwritten when a new session starts, but you can specify a number of log files to retain.

Note

If the VSI TCP/IP FTP server process does not start or mysteriously disappears, examine the beginning of `FTP_SERVER.LOG` for error messages. Because the system-wide login command procedure (`SYS$MANAGER:SYLOGIN.COM`) and the user's `LOGIN.COM` are executed as part of the server process creation, errors in these procedures can cause the server process to terminate. In most instances, however, the reason for the process terminating appears at the beginning of the `FTP_SERVER.LOG` file.

The following sample log file contains the FTP transactions involved when the user logs in under the user name `HOLMES`, issues a **DIRECTORY** command, and then retrieves the file `FOO.BAR`.

```
-------------------------------------------------------
FTP Login request received at Wed Jun 14 19:05:10 2017
from remote IP address 127.0.0.1
-------------------------------------------------------
>>> 230 User HOLMES logged into U1:[HOLMES] at Wed 07-Jun-2017 19:05, job
 3a.
<<< TYPE A
>>> 200 Type A ok.
<<< STRU F
>>> 200 Stru F ok.
<<< MODE S
>>> 200 Mode S ok.
<<< PORT 127,0,0,1,4,14
>>> 200 Port 4.14 at Host 127.0.0.1 accepted.
<<< LIST
>>> 150 List started.
>>> 226 Transfer completed.
<<< PORT 127,0,0,1,4,15
>>> 200 Port 4.15 at Host 127.0.0.1 accepted.
<<< RETR foo.bar
>>> 150 ASCII retrieve of USERS:[HOLMES]FOO.BAR;1 started.
>>> 226 Transfer completed. 210 (8) bytes transferred.
<<< QUIT
>>> 221 QUIT command received. Goodbye.
 HOLMES  job terminated at 11-JUN-2017 19:05:23.08
```

By setting the logical name `IP$FTP_SERVER_LOG_LIMIT` in the `LOGIN.COM` file, you can specify that log files be retained. Set the logical name to a dash (-) to retain all log files, or specify a number in the range of 1 to 32000.

Directory size restrictions limit the number of potential files that can actually be created. If you do not specify a number or value, one log file is created or overwritten for each FTP session. Use the DCL **PURGE** command to delete unneeded log files. The following example specifies that 42 log files be retained:

```
$ DEFINE IP$FTP_SERVER_LOG_LIMIT 42
```

7.2.5. Managing FTP Security

Because the FTP server process is started by running `SYS$SYSTEM:LOGINOUT.EXE`, both the system-wide login command procedure (`SYS$MANAGER:SYLOGIN.COM`) and the specific user's `LOGIN.COM` are executed. As a result, any customization (default file protection, process/job logical name definitions, and so on) done in these command procedures is available under the FTP server process.

All standard OpenVMS security-checking mechanisms also validate the FTP server process creation. If a login command procedure contains any commands specific to interactive jobs (for example, **SET TERMINAL** commands), the FTP server process may crash. To avoid this problem without altering the functionality of command procedures, use the DCL lexical function F$MODE with interactive commands. For example:

```
$ IF F$MODE() .EQS. "INTERACTIVE" THEN SET TERMINAL /INQUIRE
```

You can use the following logicals in the `FTP_SERVER.COM` command procedure to restrict specific users from some types of access:

- The following logical restricts *'username'* to accessing only the specified directories when connecting to the host using FTP:

  ```
  IP$'username'_FTP_DIRECTORY "directory-spec,..."
  ```

  This logical is used in the **FTP SERVER /DIRECTORY=*directory_spec*,...** qualifier.

- The following logical restricts *'username'* to only the type of access specified when accessing the host via FTP:

  ```
  IP$'username'_FTP_CONTROL "access-spec,..."
  ```

  This logical is used in the **FTP SERVER/ACCESS=*access-spec*,...** qualifier.

  ***access-spec*=[NOLIST], [NOWRITE], [NOSPAWN], [NOREAD], or [NODELETE]**

- The following logical limits the information given out on connection or when using the **STAT** command:

  ```
  IP$FTP_CONNECT_BANNER "FTP server name"
  ```

  If this logical is defined as whitespace, operating system and TCP stack information is removed from the FTP server connection banner. If this logical is defined with a specific FTP server name, the information banner does not appear in response to the **STAT** command.

7.2.6. Accepting Wildcards Upon Delete

You can apply the logical `IP$FTP_DISALLOW_WILDCARD_DELETES` to anything to disallow the new functionality of accepting wildcards on delete. This may be done at the process, group, or system level.

7.2.7. Specifying a Message at Connect Time

The `IP$FTP_ANNOUNCE` logical provides a `SYS$ANNOUNCE`-style message along with the "220" banner at connect time. Define the logical in a fashion similar to `SYS$ANNOUNCE`, using one of the following commands:

```
$ DEFINE/SYSTEM IP$FTP_ANNOUNCE "message text"
```

In the following version, the announcement is in the specified file:

```
$ DEFINE/SYSTEM IP$FTP_ANNOUNCE "@file specification"
```

7.2.8. Specifying UNIX-Style Listings

If you define the logical name `IP$FTP_UNIX_STYLE_BY_DEFAULT`, the FTP Server starts in UNIX emulation mode.

The control of version number displays has been reworked in response to **LIST** and **NLST** commands. The default is VMS-mode output.

The logical name `IP$FTP_UNIX_STRIP_VERSION` no longer has any effect. In UNIX mode, the FTP Server always removes version numbers from directory listings.

The logical name `IP$FTP_STRIP_VERSION` causes VMS mode output to have no versions.

Note

It is recommended that you NOT use the `IP$FTP_STRIP_VERSION` logical. Stripping version numbers from the VMS mode **LIST** output can cause problems for some FTP clients (notably WS_FTP).

The logical name `IP$FTP_ALL_VERSIONS` requests the **NLST** and **LIST** commands to display all version numbers. If `IP$FTP_ALL_VERSIONS` is defined, the logical name `IP$FTP_STRIP_VERSION` has no effect.

Note

`IP$FTP_ALL_VERSIONS` is ignored if the FTP Server is in UNIX emulation mode.

The FTP Server updates UNIX emulation improving VSI TCP/IP interoperability with various FTP Clients. Features of the UNIX emulation mode are:

- The syntax you use for a directory determines the mode you want. For example, CWD / uses UNIX mode; CWD [] uses VMS mode.
- The **LIST** command returns output similar to that produced by *`ls -al`*.
- The logical name `IP$FTP_UNIX_STYLE_CASE_INSENSITIVE` allows UNIX style filename handling to be case insensitive.
- Mixed case file names and those with special characters are translated into legal OpenVMS file names using the NFS mappings.
- The directories listing uses UNIX syntax. For example, USERS:[MRUHL] becomes *`/users/mruhl`*.
- When changing directories or referencing files using an absolute UNIX pathname, directory lookups treat `SYS$LOGIN` as if they were the root directory (/). So, if `SYS$LOGIN` is `USERS:[MRUHL]`, *`/login.com`* refers to `USERS:[MRUHL]LOGIN.COM` and *`/IP$common_root/IP`* refers to `USERS:[MRUHL.IP$COMMON_ROOT.IP]` if that directory exists. If it does not exist, the first segment of the pathname is used as the device specification in a second lookup attempt, and *`/IP$common_root/IP`* refers to `IP$COMMON_ROOT:[IP]`.
- If the FTP Server is in UNIX mode, the **SYST** command displays the banner “UNIX VSI TCP/IP Unix Emulation.” If the FTP Server is in VMS mode, the **SYST** command displays the equivalence string associated with the `IP$FTP_SYST_BANNER` logical name (if defined). Otherwise, the **SYST** command displays “VMS VSI TCP/IP V*`x.y(rev)`*”:
  - V*`x.y`* is the VSI TCP/IP version number.
  - *`(rev)`* is the revision number of the FTP Server.

Note

The logical name `IP$FTP_SYST_BANNER` is ignored if the FTP Server is already in UNIX mode.

The FTP Server does not drop spaces in filenames. It converts them to the character sequence $7A.

The FTP Server protects privileged ports by not opening data connections to privileged ports.

The file open routines allow all modes to fetch data from a file open for write but marked for shared access.

The FTP Service corrects synchronization problems resulting in messages repeatedly sent to the FTP Client.

The interoperability problem with Microsoft Internet Explorer where Internet Explorer expects the server PORT reply to a **PASV** request to be enclosed in parentheses contrary to RFC stipulation that the client should search for a digit, not the left parenthesis has been fixed.

There is no **PASV** command interference with data link window sizing.

If you want the device name, the file name, and the directory name included in the results of all **NLST** commands, define the logical `IP$FTP_INCLUDE_DEVICE_IN_NLST`. This logical may be declared system wide or in the user's `LOGIN.COM` file.

The FTP Service corrects a problem with RENAME operations with UNIX-style file specifications. The RENAME operation overrides the current protection of the file to do the operation and then restores it afterwards. This is necessary because directories are created such that they cannot be deleted without changing the protection. To cause RENAME to observe the file protection, define the logical `IP$FTP_OBSERVE_VMS_PROTECTION` to true.

7.2.9. UNIX File Names

An FTP default is to rename files by changing the final dot (.) to $5N. The logical `IP$FTP_DODROP1DOT` lets you override this FTP default by dropping the final dot and NOT adding $5N.

VMS always implies that a dot is present in file names regardless of whether it is followed by an extension. OpenVMS also does not support multiple dots in a file name. The rule FTP follows is that when there is only one dot, and that dot is the final character, the dot is converted to $5N. The resultant local file is then distinguishable from a similarly named file that did not have a dot. For example, "FILE." becomes "FILE$5N" when using the FTP default; however, "FILE." becomes "FILE" with the logical defined.

The FTP server displays the creation month, day, and year of a file for a UNIX mode directory if the file is older than 1 year (365 days). If the logical `IP$FTP_UNIX_YEAR_OLD_FILES` is defined False, No, or 0 (zero), the old behavior is restored, displaying all files with Month, Day, and Time.

The logical `IP$FTP_DISALLOW_UNIX_STYLE` controls whether UNIX-style filename parsing is done. The default value for `IP$FTP_DISALLOW_UNIX_STYLE` is true (T), UNIX-style filename parsing is not handled. If you want UNIX-style filename parsing, you must define this logical as `FALSE`. When UNIX-style parsing is enabled, it is not normally done until a **CD** command has been done with a directory specification that contains a "/" in it. For example:

```
FTP> cd ../my_directory
```

Note

For some FTP clients (VSI TCP/IP is one of them) you will have to enclose the directory specification in quotes (“ ”) when it contains the “/” to prevent the client from attempting to parse it.

To exit UNIX-type filename parsing, use a **CD** command with either the “[” or “<” character in the directory specification. For example:

```
FTP> cd [-.my_directory]
$ DEFINE/SYSTEM/NOLOG/EXEC IP$FTP_DISALLOW_UNIX_STYLE FALSE
```

Some graphical display FTP clients expect the output of directory commands to be in a UNIX system format. To enable this UNIX format, use the following either at the system level or in the user's `LOGIN.COM`:

```
$ DEFINE IP$FTP_DISALLOW_UNIX_STYLE FALSE
```

and

```
$ DEFINE IP$FTP_UNIX_STYLE_BY_DEFAULT ANYTHING
```

7.2.10. Specifying the Maximum Idle Time

The `IP$FTP_MAXIMUM_IDLE_TIME` logical specifies the length of time before an idle FTP server connection times out. The value is set in seconds, with a default of 300 seconds. If this logical is set to 0, timeouts are disabled.

The logical name `IP$FTP_FAST_TIMEOUT` is equivalent to the settings in these logicals `IP$NAMESERVER_RETRANS` and `IP$NAMESERVER_RETRY` for the FTP server process to 5 and 2 respectively. This helps the FTP Server start up faster when DNS PTR records for the client are improperly defined or nonexistent.

7.2.11. Using FTP Site Commands

Table 7.2 lists the commands for controlling and configuring the FTP server from the FTP prompt.

Table 7.2. FTP Site Commands

Command	Description
SITE SHOW TIME	Shows the time on the system the FTP server is running on.
SITE NONE	This is a NO OPERATION.
SITE PRIV *`[privileges]`*	Turns ON or OFF OpenVMS privileges. If no privileges are specified, displays the current privileges.
SITE RMS BLOCK [ON|OFF]	Turns Block Image mode transfers ON or OFF, or if there is no argument, displays the current state. When image (binary) transfers are done with Block Image mode OFF (the default), the FTP server opens OpenVMS record files such that the record control information is not included in the data and a linefeed separates each record. With Block Image mode ON the record control information and the data are transferred.

Command	Description
SITE RMS RECSIZE [*value*]	With no argument, displays the current record size. With an argument, sets the record size for Image **PUT** transfers to the size specified. The default is 512.
SITE SPAWN *command*	Spawns a subprocess and uses the rest of the line as a OpenVMS DCL command. Not valid for CAPTIVE processes.
SITE RMS STREAM [ON|OFF]	With no argument, displays the current state. When ON, writes text files in Stream-LF format. When OFF (default), writes text files as OpenVMS carriage-control record files.
SITE +VMS+	Enables VMS + mode transfers.
SITE VMS	Disables VMS + mode transfers.
SITE WINDOW-SIZE	With no argument, displays the TCP window size. With an argument, sets the window size for the data channel. Default value is 32768 bytes

7.2.12. Defining FTP Messages

The `IP$FTP_221_REPLY` logical lets you define the message users see when they end the FTP session. If you do not define this logical, VSI TCP/IP uses the default message instead “221 **QUIT** command received. Goodbye.” You can define a text string or file. If the string starts with the “at” sign @, it specifies the name of a file containing text to be displayed. For example:

```
$ DEFINE/SYSTEM/EXEC IP$FTP_221_REPLY -
_$ "Connection to FTP server has been closed"
```

Now, when the user closes the FTP connection, the following message appears:

```
221 Connection to FTP server has been closed
```

The `IP$FTP_230_REPLY` logical defines a message to appear when a user successfully logs in. If you do not define this logical, VSI TCP/IP uses the default message instead. As with `IP$FTP_221_REPLY`, you can define a text string or file. For example:

```
$ DEFINE/SYSTEM/EXEC IP$FTP_230_REPLY-
_$ "Login successful"
```

Now, when the user logs in using FTP, the following message appears:

```
230 Login successful
```

The `IP$FTP_ANONYMOUS_230_REPLY` logical defines a message to appear when an ANONYMOUS user successfully logs in. If you do not define this logical, VSI TCP/IP uses the default message instead. As with `IP$FTP_230_REPLY`, you can define a text string or file. For example:

```
$ DEFINE/SYSTEM/EXEC IP$FTP_ANONYMOUS_230_REPLY-
_$ "ANONYMOUS login successful"
```

Now, when a user logs in using the ANONYMOUS account, the following message appears:

```
230 ANONYMOUS login successful
```

The `IP$FTP_421_REPLY` logical lets you defines the message user's see when they connect to the server but should not log in. After sending the message, the connection closes. For example, you can define this logical to prevent FTP access for a short time period. Be sure to deassign the logical after this period to allow FTP access again. If the string starts with the "at" sign @, it specifies the name of a file containing text to be displayed. For example:

```
$ DEFINE/SYSTEM/EXEC IP$FTP_421_REPLY-
_$ "System maintenance in progress until 17:30"
```

Now, when the user connects to the host using FTP, the following message appears and then the connection closes:

```
421 System maintenance in progress until 17:30
```

7.2.13. Specifying the Name of a Log File

The `IP$FTP_LOGFILE` (system level, executive mode) logical can be defined to specify the name of a log file. For example:

```
$ DEFINE/SYSTEM/EXEC IP$FTP_LOGFILE-
_$ SYS$COMMON:[SYSMGR]FTPLOGIN.LOG
```

If this logical exists, the FTP server writes a record to the specified file each time a user attempts to log in. Each record includes the date and time, the remote host's internet address, and whether the login succeeded. This is especially useful to use if you suspect someone has tried to break into the FTP server.

This logical specifies the name of the file to which *all* commands and responses to ANONYMOUS FTP services are logged. If `IP$FTP_LOG_ALL_USERS` is also defined, then commands and responses for all users are logged. If VSI TCP/IP is already running, the Master Server must be restarted (`IP$SYSTARTUP.COM`) for these definitions to take effect.

The logical `IP$FTP_LOG_ALL_USERS` causes all commands and responses to be logged to the file defined by `IP$FTP_LOGFILE`. The default (when this logical is not defined) is to just log the commands and responses for anonymous users.

```
$ DEFINE/SYSTEM/EXEC IP$FTP_LOG_ALL_USERS TRUE
```

The FTP client and the FTP server normally check the record size of an ASCII transfer and disallow more than 8192 byte records (as a sanity check). However, you can define the `IP$FTP_MAXREC` logical to override the default of 8192. The definition of the `IP$FTP_MAXREC` logical is commented out but defined in the `FTP CONTROL.COM` file as follows:

```
$ DEFINE/SYSTEM/NOLOG/EXEC IP$FTP_MAXREC 8192
```

Note that the maximum record size supported by OpenVMS is 65535.

7.2.14. Defining a File Name

The `IP$DIRECTORY_MESSAGE_FILENAME` logical can be defined to name the file you want to appear when a session enters a directory. For example:

```
$ DEFINE/SYSTEM/EXEC IP$DIRECTORY_MESSAGE_FILENAME flowers.txt
```

7.3. Password Lifetime Warnings

This section describes how to define password messages in VSI TCP/IP.

7.3.1. Defining Password Messages

The `IP$FTP_PASSWORD_WARNING_MESSAGE` logical lets you define the message users see when their password is going to expire. If you want the amount of time before the password expires to be included, add %s to the logical.

```
$ DEFINE/SYSTEM/EXEC IP$FTP_PASSWORD_WARNING_MESSAGE %s
$ DEFINE/SYSTEM/EXEC IP$FTP_PASSWORD_WARNING_MESSAGE message text string
```

The `IP$FTP_PASSWORD_WARNING_TIME` logical uses the OpenVMS delta time to specify the minimum remaining lifetime for the user's password. If the remaining lifetime is greater than the OpenVMS delta time then no message appears. It is necessary to define this value to enable checking for the remaining lifetime of a password.

```
$ DEFINE/SYSTEM/EXEC IP$FTP_PASSWORD_WARNING_TIME dddd hh:mm:ss.hh
```

The `IP$FTP_RECEIVE_THRESHOLD` logical specifies the amount of buffer space that can be used to buffer transmitted data on the data socket. The default value if 6144. If this logical is defined and it begins with a /, then it specifies the fraction of the window size; if only a fraction is specified, then it indicates the number of bytes to be used. The ? in the logical represents where defined values go.

```
$ DEFINE/SYSTEM/EXECUTIVE IP$FTP_RECEIVE_THRESHOLD ?
```

The `IP$FTP_NOLOGIN_EXPIRED_PASSWORD` logical lets you prevent accounts with expired passwords from logging in.

```
$ DEFINE/SYSTEM/EXEC IP$FTP_NOLOGIN_EXPIRED_PASSWORD TRUE
```

will prevent a user with an expired password from logging in and displays the following message:

```
<Your password has expired; contact your system manager>
```

7.3.2. Checking IP Address

By default, the VSI TCP/IP FTP server checks the IP address given in the port command and does not make the connection if the IP address does not match that of the control connection. This can be disabled by defining the logical `IP$FTP_SERVER_RELAXED_PORT_COMMAND`.

```
$ DEFINE IP$FTP_SERVER_RELAXED_PORT_COMMAND 197.0.0.1
```

7.4. Configuring the FTP server for TLS (FTPS)

Follow these steps to configure the VSI TCP/IP FTP server to allow TLS authentication:

- Generate or obtain certificate and key files. On OpenVMS V8.3 `SSL1$COM:SSL1$CERT_TOOL` can be used to do this.
- Place the certificate and key file where you want them and verify that the protection is set such that world has no access.
- Using **IP CONFIGURE /SERVER** select FTP and set the RFC-4217-CERTIFICATE and RFC-4217-KEY parameters to the location and filename of the appropriate files. see the following example. Optionally set the REQUIRE_TLS parameter to YES.

  ```
  $ IP CONFIGURE/SERVER
  ```

```
SERVER-CONFIG> SELECT FTP
SERVER-CONFIG> SET PARAMETER
Add Parameter: RFC-4217-CERTIFICATE SSL$CERT:SERVER.CRT
Add Parameter: RFC-4217-KEY SSL$CERT:SERVER.KEY
Add Parameter:
[Service specific parameters for FTP changed]
SERVER-CONFIG>exit
[Writing configuration to SYS$COMMON:[IP.CONFIG]SERVICES.MASTER_SERVER]
```

- Restart the VSI TCP/IP master server `IP$:START_SERVER RESTART`.

7.4.1. FTP server parameters for TLS

RFC-4217-CERTIFICATE Specifies the certificate file to be used with RFC 4217 negotiation. The certificate and key files must be created by an external means such as the SSL certificate tool and be in PEM format. Both a certificate and key file must be specified set up to allow TLS negotiation. On OpenVMS V8.3 you can use **@SSL1$COM:SSL1$CERT_TOOL**.

RFC-4217-KEY Specifies the private key file to be used with RFC 4217 negotiation. The certificate and key files must be created by an external means such as the SSL certificate tool and be in PEM format. Both a certificate and key file must be specified set up to allow TLS negotiation. On OpenVMS V8.3 you can use **@SSL1$COM:SSL1$CERT_TOOL**.

REQUIRE-TLS YES Specifies that user authentications other than anonymous and users that have no password must use TLS authentication. The FTP **USER** command will get a 530 response if it is issued before TLS authentication has been done. This prevents passwords from being exchanged in clear text mode with the users.

7.5. Network Service Monitoring

FTP’s network service monitoring is based on RFC 2788 (Network Services Monitoring MIB). Information is maintained only while the service is active. The following items from the Network Services Monitoring MIB (RFC 2788) are available in the enterprises.105.2.25 MIB:

Table 7.3. Network Services Monitoring Items

Item	Description
ApplAccumulatedInboundAssociations	(Counter) the total number of connections that the FTP Listener program has serviced since it was started. enterprises.105.2.21.10
ApplDescription	(String) Description of the program/application. This banner is printed when the client connects to the FTP Listener program. enterprises.105.2.21.16
ApplInboundAssociations	(Counter) The number of connections currently active. enterprises.105.2.21.8
ApplIndex	(Integer) unique application index. The port FTP is offered on (21). enterprises.105.2.21.1
ApplLastChange	(TimeTicks) the value of sysUpTime when the FTP Listener program entered the current state. enterprises.105.2.21.7
ApplLastInboundActivity	(TimeTicks) the value of sysUpTime at the time the most recent connection was established. enterprises.105.2.21.12
ApplName	(String) FTP. enterprises.105.2.21.2

Item	Description
ApplOperStatus	(Integer) the operational status of the FTP Listener program; the values are: up(1), down(2), halted(3), congested(4), restarting(5), quiescing(6). Some of these values may not be used. enterprises.105.2.21.6
ApplRejectedInboundAssociations	(Counter) the number of connections that have been rejected (due to not being allowed from the access list values). enterprises.105.2.21.14
ApplUptime	(TimeTicks) the value of the SNMP variable sysUpTime when the FTP Listener program was started. This time has a resolution of 5 minutes. enterprises.105.2.21.5
ApplVersion	(String) the version of the FTP Listener program. enterprises.105.2.21.4

This feature requires the SNMP Agent X functionality. To use this SNMP must be configured to have Agent X service enabled, and to allow the system's IP and the local host addresses (127.0.0.1) to each be an AGENTX_PEER. See Chapter 11 for more information. This information can be displayed with the **IP SHOW /SNMP** command and can be displayed with a MIB browser.

To enable network service monitoring, do the following:

```
$ IP CONFIGURE /SERVER
 SELECT FTP
 SET FLAGS SNMP_MONITORED | PASS_FOREIGN_SOCKET
 WRITE
 EXIT
$ @IP$:IP$SYSTARTUP
```

Any service using TCP_INIT, TCP_LISTEN, and TCP_CONNECTED routines may use **SET FLAGS SNMP_MONITORED**. The level of functionality may vary with the service.

7.6. Session Accounting

VSI TCP/IP can record accounting information from services that have been enabled. Currently this includes FTP and SMTP. The accounting information includes information about when a network session took place and how much data was transferred. The accounting facility is enabled by setting the accounting port and the accounting host and reading `IP$:ACCOUNTING.CONF` for additional configuration information. The format of the accounting records is described in `IP$ROOT:[IP.EXAMPLES]ACCOUNTING.H`.

A sample program using this is in `IP$ROOT:[IP.EXAMPLES]ACC_DUMP.C`.

You must configure FTP and session accounting in order to activate the accounting function. FTP-ACCOUNTING-HOST is the name of the system running the accounting program. `FTP-ACCOUNTING-PORT` is the port number that the program was set up to listen on. FTP accounting can be configured with the following:

```
IP CONFIGURE /NETWORK
SET FTP-ACCOUNTING-HOST name
SET FTP-ACCOUNTING-PORT number
WRITE
EXIT
```

In order for accounting to be activated before your next reboot, you can define the logicals as follows:

```
$ DEFINE/SYSTEM/EXECUTIVE IP$FTP_ACCOUNTING_HOST lillies
$ DEFINE/SYSTEM/EXECUTIVE IP$FTP_ACCOUNTING_PORT 1234
```

Note

The accounting port must be set to an unused port, not the port for the service on which accounting is being enabled.

The next section explains how to configure the file.

The collected accounting information can be displayed with the **IP ACCOUNTING** command. See the *VSI TCP/IP Administrator's Reference* for more information about the **IP ACCOUNTING** command.

7.6.1. Configuration File

The Accounting configuration file is `IP$:ACCOUNTING.CONF`. The Accounting configuration file defines:

- The Port the Accounting program listens on. This should be an unused port, not the port for the service on which logging is being enabled, and the same port specified to FTP or SMTP.
- The name of the file used for accounting records. This file is opened shareable and new records are always appended to it. To start a new file stop the Accounting program, delete (or rename) the existing file, and restart the Accounting program.
- The IP addresses of systems that are allowed to write accounting records to this host.

Note

After editing the configuration, stop and restart the Accounting program so that the changes can take effect.

7.6.2. File Format

Follow these guidelines when entering data in the Accounting configuration file:

- Allow one line for each item.
- Enter information in any order; in upper- or lowercase.
- Use a pound sign (#) or exclamation point (!) to denote comments. The Accounting facility ignores all information following these characters.

The commands that can be in `IP$:ACCOUNTING.CONF` are:

- PORT *`port_number`*: The TCP port that the accounting program should listen on.
- PEER *`ip-address`*: The IP address of a host that is allowed to log records with the accounting software.
- FILENAME *`filename`*: The name of the file that the accounting records will be written to. The `IP$:` device is assumed if a device is not specified as part of the file specification.

7.6.3. Enabling the Accounting Logger

To enable the FTP accounting logger, do the following:

```
$ IP CONFIGURE/SERVER
SERVER-CONFIG> ENABLE ACCOUNTING
SERVER-CONFIG> WRITE
SERVER-CONFIG> EXIT
IP$SYSTARTUP.COM
```

7.6.4. Displaying the Contents of the Logging File

To view accounting information, do the following:

```
$ IP ACCOUNTING/INPUT=accounting_data_file [/output=output filename] -
_$ [/since=start_date] [/before=end_date] [/protocol={SMTP, FTP, MAIL}] [/
CSV]
```

- *`accounting_data_file`* is the name of the logging file you want to see.
- *`output filename`* is the name of the file you want to call this information. If this field is omitted, the information displays to the terminal screen.
- *`start_date`* is the beginning date you want the command to start with. The date format is **[DD-MMM-YYYY [:]] [hh:mm:ss]cc]**. If not specified, all records display up to the end of the data found.
  - **DD** is the day of the month, counting from 01.
  - **MMM** is the abbreviation for the month, like JAN, FEB, MAR.
  - **YYYY** is the number of the year, including the century (2013, 2014, 2015, 2016, 2017).
  - **hh** is the hour, from 00 to 23.
  - **mm** is the minute, from 00 to 59.
  - **ss** is the second, from 00 to 59.
  - **cc** is hundredths of seconds.

  The time is always in local time.
  - *`end_date`* is the ending date you want the command to end with. The date format is **[DD-MMM-YYYY [:]] [hh:mm:ss]cc]**. If not specified, all records display until the end of the file.
  - *`protocol`* is any combination of SMTP, FTP, or MAIL.
  - *`CSV`* is the Comma Separated Values, for input to products like Excel.

7.6.5. Accounting File Record Format

The accounting file is written using OpenVMS RMS records. The format of these records is defined in `IP$ROOT:[IP.EXAMPLES]ACCOUNTING.H`, and listed below:

```
/*
```

```
 * PDU format
 */
struct accountingPDU {
 char version;
 char type;           /* type of record */
/*
 * FTP:
 *  C - Client
 *  S - Server
 *
 * SMTP:
 *  N - Network delivery (send)
 *  L - Local delivery (received)
 *  F - Forwarded
 *  R - Returned
 *  D - Delivery Receipt
 *  Q - ReQueued
 *
 */
 char flags;          /* not currently used */
 char reserved;       /* for future use */
 int payload_length;  /* length (in bytes) of data after header */
 int port; /* IP port of reporting service - 25 SMTP, 21 - FTP */
 int reporterIP;      /* IP address of reporter */
};
struct FTPaccounting_data {
 struct accountingPDU header;
 int start_time[2];   /* OpenVMS time that session started */
 int end_time[2];     /* OpenVMS time that session ended */
 int datasent;        /* KBytes of file data sent */
 int datarecv;        /* KBytes of file data received */
 int filessent;       /* Number of files sent */
 int filesrecv;       /* Number of files received */
 int partnerIP;       /* IP address of partner */
 char user[12];       /* username that operations were done under */
};
struct SMTPaccounting_data {
 struct accountingPDU header;
 int date[2];         /* Time of activity */
 int msg_size;        /* size of message in bytes */
 int from_str_size;   /* size of From: string */
 int to_str_size;     /* size of To: string */
 char from_to_str[1]; /* text of From & To string */
};
#define accounting_Close 1
typedef struct accounting_peer_info {
 struct accounting_peer_info *next;
 ulong ia;
} accounting_peer_info;
#define MAX_STRING_LEN 255
```

7.7. FTP and IPv6

A separate service entry (FTP_INET6) is available to support FTP over IPv6. The Network Service Monitoring and Session Accounting have not yet been updated for IPv6. The same logicals and command files are used for both FTP over IPv4 and FTP over IPv6. When IPv6 is in use FTP uses the EPSV and EPRT commands. Other than the differences noted, FTP over IPv6 should be the same as

FTP over IPv4. FTP is capable of accepting mapped IPv4 connections, so it is possible to disable the FTP service and enable FTP_INET6 to take care of both IPv4 and IPv6.

Chapter 8. Configuring the Font Server

This chapter explains how to use the VSI TCP/IP font server to provide fonts for X11R5 (and later) X servers on your network. To understand the material in this chapter, you should be familiar with font administration on X11R5 servers.

8.1. Understanding the Font Server

The VSI TCP/IP font server makes fonts on your OpenVMS system available to remote X11R5 (and later) X servers without using a distributed file system, such as NFS, or file transfer via FTP or TFTP.

The main advantages of font servers over distributed file systems or file transfer are:

- Simplicity of font administration.
- Redundancy. X servers can use multiple font servers. If one font server fails or is unavailable, the X server can request fonts from another font server.

You can add font servers to an X server font search path the same way you add directories to the font search path. For example, you can add a font server to the font search path of X servers running on UNIX systems with the **xset +fp** and **xset fp+** commands.

- When an X server needs a font, it sends a request to the font server.
- If the requested font is on the font server, the font is transferred to the X server.
- If the font is not on the font server, the X server continues to search the rest of the font search path, which may include other font servers.

You can also configure the font server to return the names of other font servers (known as *`alternates`*) that the X server can search when the font server fails to find a requested font.

8.2. The Font Server Configuration File

The VSI TCP/IP font server obtains its configuration parameters from the configuration file `IP$:FONT_SERVER.CONFIGURATION`, which is equivalent to the /usr/lib/X11/fs/config configuration file used by font servers on UNIX systems. Although the file names are different, the file formats are identical; you can use configuration files from UNIX systems on your OpenVMS host.

The configuration file is an ASCII text file that contains a list of configuration parameter names and values. Each parameter name is followed by an equals sign (=) and the desired value.

Table 8.1 describes the font server configuration parameters.

Table 8.1. Font Server Configuration Parameters

Parameter	Accepted Values	Description
cache size	cardinal number	Specifies the size of the font server's font cache. To improve font access speed, specify a large font cache. Default: 10000.

Parameter	Accepted Values	Description
catalogue	list of strings	Lists font path element names, delimited by commas. The VSI TCP/IP font server supports only a single catalogue (*all*), which contains all specified fonts.
alternate-server	list of strings	Lists alternate servers for this font server.
client-limit	cardinal number	Specifies the number of clients this font server supports before refusing service.
default-point-size	cardinal number	Specifies the default point size (in decipoints) for fonts that do not specify a point size.
default-resolutions	comma-delimited list of integers	Specifies the default resolutions the server supports. This information may be used as a hint for pre-rendering, and substituted for scaled fonts which do not specify a resolution.
error-file	string	Specifies the log file into which all warnings and errors are written.
port	cardinal number	Specifies the TCP port on which the server listens for connections.
trusted-clients	comma-delimited list of host names	Determines which hosts can use the font server. If you do not specify any hosts (the default), all hosts can use the font server. If you specify hosts, they are the only ones that can use the font server.

An example font server configuration file follows.

```
# Font server configuration file
# VSI TCP/IP for OpenVMS Font Server
# Specify the font directories to export.
#
# WARNING: The file DECW$FONT_DIRECTORY.DAT must exist. If it does not, you
# can create it with the command $ IP FONT MKFONTDIR [directory,...]
#
catalogue = sys$common:[sysfont.decw.100dpi],
 sys$common:[sysfont.decw.75dpi],
 sys$common:[sysfont.decw.common],
 sys$common:[sysfont.decw.cursor16],
 sys$common:[sysfont.decw.cursor32]
#
# Uncomment this line to start logging errors to a file on disk.
# Restart the font server to put logging into effect.
#
# error-file = VSI TCP/IP for OpenVMS:fs.errors
#
# in decipoints
default-point-size = 120
default-resolutions = 75,75,100,100
```

8.3. Specifying Font Servers

All X11R5 (and later) servers use the same syntax for specifying font servers:

```
transport/host_name:port_number[/catalogue]
```

- `transport` is "TCP."
- `host_name` is the name of the host on which the font server is running.
- `port_number` is the port on which the VSI TCP/IP font server listens for requests from remote X servers. By convention, the VSI TCP/IP font server listens on port 7000.
- `catalogue` is the catalogue the VSI TCP/IP font server provides (by default, "all"). Catalogues are the equivalent of search paths on the font server. For details on defining catalogues, see Section 8.8.

8.4. Supported Font Types

The VSI TCP/IP font server supports the following font formats:BDF, MIT SNF, Speed, DECwindows, PCF.

VSI TCP/IP also includes two commands for converting font formats:

Command	Description
`IP FONT COMPILE`	Compiles BDF fonts into PCF format. Type HELP IP FONT COMPILE for online help.
`IP FONT UNCOMPILE`	Converts fonts supported by the font server into BDF format. Type HELP IP FONT UNCOMPILE for online help.

The default font alias file name is `DECW$FONT_ALIAS.DAT` to match the default font values used by VSI.

8.5. Enabling the Font Server

To enable the VSI TCP/IP font server, use the SERVER-CONFIG utility (**IP CONFIGURE / SERVER**). For example, to enable the font server on a single OpenVMS system, enter:

```
$ IP CONFIGURE /SERVER
SERVER-CONFIG>ENABLE FONTSERVER
SERVER-CONFIG>RESTART
Configuration modified, do you want to save it ? [YES] YES
SERVER-CONFIG>QUIT
$
```

8.6. Getting Information About the Font Server

This section describes how to get information about a specific font server. Use the **IP FONT** command to obtain the following information about the font server:

- Current font server configuration (see Section 8.6.1)
- Names of available fonts (see Section 8.6.2)
- Font file data (see Section 8.6.3)

8.6.1. Checking the Font Server Configuration

To check the status of the VSI TCP/IP font server, enter:

```
$ IP FONT INFO /SERVER=font_server_name:port_number)
```

- *font_server_name* is the name of the font server you want to check. Use the font server name syntax described in Section 8.3.
- *port_number* is the port on which the font server listens. By default, the VSI TCP/IP font server listens on port 7000.

The following example shows the information generated by this command on a system that acts as a font server with no alternates.

```
$ IP FONT INFO /SERVER=WHORFIN:7000
name of server: WHORFIN:7000
version number: 2
vendor string: ABC, Incorporated
vendor release number: 5001
maximum request size:  16384 longwords (65536 bytes)
number of catalogues:  1
all
Number of alternate servers: 0
number of extensions:  0
$
```

For more information, type HELP IP FONT INFO.

8.6.2. Listing Available Fonts

To list the names of available fonts on a font server, enter:

```
$ IP FONT LIST /SERVER=font_server_name font_spec
```

- *font_server_name* is the name of the font server from which you want to obtain the list of fonts. Use the font server name syntax described in Section 8.3.
- *font_spec* is a font specification, in which you may include wildcard characters.

The following example shows the command that lists all "fixed" fonts on the font server.

```
$ IP FONT LIST /SERVER=WHORFIN:7000 *FIXED*
-misc-fixed-bold-r-normal--0-0-75-75-c-0-iso8859-1
-misc-fixed-bold-r-normal--13-120-75-75-c-70-iso8859-1
-misc-fixed-bold-r-normal--13-120-75-75-c-80-iso8859-1
-misc-fixed-bold-r-normal--15-140-75-75-c-90-iso8859-1
-misc-fixed-bold-r-semicondensed--0-0-75-75-c-0-iso8859-1
-misc-fixed-bold-r-semicondensed--13-120-75-75-c-60-iso8859-1
-misc-fixed-medium-r-normal--0-0-75-75-c-0-iso8859-1
-misc-fixed-medium-r-normal--10-100-75-75-c-60-iso8859-1
-misc-fixed-medium-r-normal--13-120-75-75-c-70-iso8859-1
-misc-fixed-medium-r-normal--13-120-75-75-c-80-iso8859-1
-misc-fixed-medium-r-normal--14-130-75-75-c-70-iso8859-1
-misc-fixed-medium-r-normal--15-140-75-75-c-90-iso8859-1
-misc-fixed-medium-r-normal--20-200-75-75-c-100-iso8859-1
-misc-fixed-medium-r-normal--8-80-75-75-c-50-iso8859-1
```

```
-misc-fixed-medium-r-normal--9-90-75-75-c-60-iso8859-1
-misc-fixed-medium-r-semicondensed--0-0-75-75-c-0-iso8859-1
-misc-fixed-medium-r-semicondensed--12-110-75-75-c-60-iso8859-1
-misc-fixed-medium-r-semicondensed--13-120-75-75-c-60-iso8859-1
-sony-fixed-medium-r-normal--0-0-100-100-c-0-iso8859-1
-sony-fixed-medium-r-normal--16-120-100-100-c-80-iso8859-1
-sony-fixed-medium-r-normal--24-170-100-100-c-120-iso8859-1
fixed
fixed
```

For more information, type HELP IP FONT LIST.

8.6.3. Viewing Font Data

To list the data that comprises a specific font, log into the host running the font server, and enter:

```
$ IP FONT SHOW /SERVER=font_server_name font_spec
```

- *font_server_name* is the name of the font server from which you want to obtain the font data. Use the font server name syntax described in Section 8.3.
- *font_spec* is a font specification, in which you may include wildcard characters.

For information on other **IP FONT SHOW** qualifiers, refer to the *VSI TCP/IP Administrator's Reference.*

The following example shows the command that lists the data for two characters in the Courier font:

```
$ IP FONT SHOW /SERVER=WHORFIN:7000 /START=52 /END=53 *COURIER*
opened font *COURIER*
Direction: Left to Right
Range: 32 to 255
Default char: 32
Min bounds:
Left: -2   Right: 1   Ascent: -1   Descent: -5   Width: 6
Max bounds:
Left: 3   Right: 7   Ascent: 9   Descent: 2   Width: 6
Font Ascent: 8 Font Descent: 2
FONT  -Adobe-Courier-Bold-O-Normal--11-80-100-100-M-60-ISO8859-1
FOUNDRY Adobe
FAMILY_NAME   Courier
WEIGHT_NAME   Bold
SLANT  O
SETWIDTH_NAME  Normal
ADD_STYLE_NAME
PIXEL_SIZE   11
POINT_SIZE   80
RESOLUTION_X  100
RESOLUTION_Y  100
SPACING M
AVERAGE_WIDTH  60
CHARSET_REGISTRY    ISO8859
CHARSET_ENCODING    1
CAP_HEIGHT   6
X_HEIGHT    5
FACE_NAME    Courier Bold Oblique
COPYRIGHT    Copyright (c) 1984, 1987 Adobe Systems Incorporated. All
Rights Reserved. Copyright (c) 1988, 1991 Digital Equipment Corporation.
```

```
All Rights Reserved.
NOTICE No mark
_DEC_DEVICE_FONTNAMES  PS=Courier-BoldOblique
_DEC_PRODUCTINFO    DECwindows Fonts V2.2, 07-Nov-1991
RELATIVE_SETWIDTH    50
RELATIVE_WEIGHT 70
CHARSET_COLLECTIONS   ASCII ISO8859-1 ADOBE-STANDARD
FULL_NAME    Courier Bold Oblique
RESOLUTION   138
QUAD_WIDTH   6
char #52 '4'
Left: 0   Right: 5   Ascent: 7   Descent: 0   Width: 6
---##
--###
-#-##
#--#-
#####
--##-
--##-
char #53 '5'
Left: 0   Right: 6   Ascent: 7   Descent: 0   Width: 6
--####
-##---
-###--
--##-
---##-
#--##-
###---
$
```

For more information, type HELP IP FONT SHOW.

8.7. Controlling the VSI TCP/IP Font Server

This section describes how to control the font server.

Use the **IP NETCONTROL FONTSERVER** command for the following tasks:

- Starting the font server (see Section 8.7.1)
- Stopping the font server (see Section 8.7.2)
- Restarting the font server (see Section 8.7.3)
- Reloading the font server configuration (see Section 8.7.4)
- Flushing the font server cache (see Section 8.7.5)
- Resetting the font server (see Section 8.7.6)

8.7.1. Starting the Font Server

To start the VSI TCP/IP font server, enter:

```
$ IP NETCONTROL FONTSERVER START
< FS Server Started, process id pid
```

When the font server starts, it reads the master configuration file, `IP$:FONT_SERVER.CONFIGURATION`. For information about the master configuration file, see Section 8.2.

8.7.2. Stopping the FS Server

To stop the VSI TCP/IP font server, enter:

```
$ IP NETCONTROL FONTSERVER SHUTDOWN
< FS Server Shutdown
```

8.7.3. Restarting the Font Server

Restarting the font server is a convenient alternative to first stopping and then starting it as described in Section 8.7.1 and Section 8.7.2.

When the font server restarts, it reads the configuration file, `IP$:FONT_SERVER.CONFIGURATION`. For information about the font server configuration file, see Section 8.2.

To restart the VSI TCP/IP font server, enter:

```
$ IP NETCONTROL FONTSERVER RESTART
< FS Server Started, process id pid
```

Note

Because the font server provides fonts on request, restarting does not disrupt any connections.

8.7.4. Reloading the Font Server Configuration

Changes to the font server configuration file (`IP$:FONT_SERVER.CONFIGURATION`) only take effect when the font server is started, restarted, or when the configuration files are reloaded. Reloading the font server allows you to reload font server configuration files without restarting the font server.

For information on the font server configuration file, see Section 8.2. To reload the font server configuration file, enter:

```
$ IP NETCONTROL FONTSERVER RELOAD
< WHORFIN.FLOWERS.COM Network Control 10.5 (nnn) at Wed 26-Apr-2017 1:33PM-
PST
< OK: FS server configuration reloading
```

Note

Because the font server provides fonts on request, reloading does not disrupt active connections.

8.7.5. Flushing the Font Server Cache

To improve performance, the font server keeps copies of requested fonts in a cache. To flush the font cache, enter:

```
$ IP NETCONTROL FONTSERVER FLUSH
< WHORFIN.FLOWERS.COM Network Control 10.5(nnn) at Wed 26-Apr-2017 1:36PM-
PST
< OK: Font Server cache flushed
```

The size of this cache is defined in the font server configuration file, `IP $:FONT_SERVER.CONFIGURATION`. For details about the font server configuration, see Section 8.2.

8.7.6. Resetting the Font Server

For convenience, VSI TCP/IP provides a **RESET** command to flush and reload the font server. To reset the font server, enter:

```
$ IP NETCONTROL FONTSERVER RESET
< WHORFIN.FLOWERS.COM Network Control 10.5(nnn) at Wed 26-Apr-2017 1:37PM-
PST
< OK: Font Server reset
```

8.8. Defining Font Catalogues

Font catalogues are the font server equivalent of X server *`font search paths`*. To make fonts available via the font server, add the directories in which they reside to the catalogue line in the font server configuration file `IP$:FONT_SERVER.CONFIGURATION`.

For example, the default catalogue definition supplied with VSI TCP/IP is defined as:

```
catalogue = sys$common:[sysfont.decw.100dpi],
sys$common:[sysfont.decw.75dpi],
sys$common:[sysfont.decw.common],
sys$common:[sysfont.decw.cursor16],
sys$common:[sysfont.decw.cursor32]
```

If you modify the font server configuration file, the changes only take effect when you start, restart, reload, or reset the font server.

8.9. Adding Fonts to the Font Server

To make a new font available via the font server:

1. Install the font file in the appropriate font directory on the font server host. If the font is in BDF format, you may want to convert the font into PCF format with the **IP FONT COMPILE** command to improve font server performance (type HELP IP FONT COMPILE for online help).

2. Update the font directory's `DECW$FONT_DIRECTORY.DAT` file with the **IP FONT MKFONTDIR** command (type **HELP IP FONT MKFONTDIR** for online help). This command creates the `DECW$FONT_DIRECTORY.DAT` file.

Note

If the `DECW$FONT_DIRECTORY.DAT` file is not found, the font server fails. Be sure to run **IP FONT MKFONTDIR** *manually* in each DECwindows font directory in which you add a font file. Failing to do so may result in the font server not serving the standard DECwindows fonts.

Note

When using **IP FONT MKFONTDIR** you must specify directories, not logical disks. For example, **IP FONT MKFONTDIR** `IP$:` is not valid.

3. If desired, add an alias for the new font to the font directory's `DECW$FONT_ALIAS.DAT` file.

4. Make sure the font directory is included in the catalogue statement in the font server configuration file `IP$:FONT_SERVER.CONFIGURATION`. For details, see Section 8.8. If you must modify the configuration file, reload the font server configuration (see Section 8.7.4).

   For example, to configure the VSI TCP/IP font server to provide the fonts included in the NCDware 3.0 distribution for OpenVMS, include the following font directories in your catalogue definition:

   ```
   NCD_ROOT:[FONTS.PCF.100DPI]
   NCD_ROOT:[FONTS.PCF.75DPI]
   NCD_ROOT:[FONTS.PCF.DW100DPI]
   NCD_ROOT:[FONTS.PCF.DW75DPI]
   NCD_ROOT:[FONTS.PCF.MISC]
   NCD_ROOT:[FONTS.PCF.XOL]
   ```

Chapter 9. Configuring Remote Systems with RARP, BOOTP, and DHCP Server

This chapter explains how to configure VSI TCP/IP to supply network configuration data to remote client systems when they boot.

Three services of VSI TCP/IP provides configuration data to remote systems:

- RARP (Reverse Address Resolution Protocol)
- BOOTP (Bootstrap Protocol) (responds only to BOOTP clients)
- DHCP (Dynamic Host Configuration Protocol) (responds to both BOOTP and DHCP clients)

The BOOTP and DHCP servers allow a network administrator to configure various hosts on the network from a single location. In addition to the management of IP addresses, BOOTP and DHCP also provide configuration parameters to clients, such as default gateway, domain name server, and subnet mask.

9.1. Choosing a Network Configuration Server

This section presents a brief description of the services and some criteria for deciding which protocol and services to use. The following table lists the advantages and disadvantages of the three protocols:

Service	**Advantages and Disadvantages**
RARP	Supplies IP addresses only.
BOOTP	Lets you provide vendor-specific configuration data. Works in conjunction with TFTP. Provides static configuration data only.
DHCP	Lets you provide vendor-specific configuration data. Provides both BOOTP and DHCP services. Provides dynamic configuration for mobile computing, but does not solve all problems of mobile computing.

9.2. RARP (Reverse Address Resolution Protocol)

RARP's sole function is to provide IP addresses to hosts that broadcast RARP requests with their hardware addresses.

9.3. BOOTP (Bootstrap Protocol)

BOOTP sends IP addresses and other configuration data to hosts that broadcast BOOTP requests. Because some BOOTP clients require more data to boot than can fit in a BOOTP response, BOOTP provides a means for specifying the location of a boot file. The BOOTP client can then load the file using TFTP (Trivial File Transfer Protocol). Usually, the data in the boot file (such as an X server for an X terminal) is specific to the vendor of the BOOTP client software.

The BOOTP service responds to BOOTP requests only. If you are using the BOOTP-only service, DHCP services are not available. The BOOTP server is provided for backwards compatibility for those sites not wanting to change their configuration.

9.4. DHCP (Dynamic Host Configuration Protocol)

DHCP is an extension of the BOOTP protocol. DHCP sends IP addresses and other configuration data to hosts that broadcast DHCP requests. DHCP "leases" an IP address to a remote system for a finite time. DHCP lets you manage IP addresses and configuration data for a "pool" of remote systems, which makes DHCP useful for mobile computers that connect to multiple subnets.

As with BOOTP, DHCP provides a means for specifying the location of a boot file, which the DHCP client can load, using TFTP. For details on creating a downloadable boot file for a specific type of host, refer to the vendor's documentation.

If you are using the DHCP server, all BOOTP services are available as well. A VSI TCP/IP host can have only one of the servers (BOOTP or DHCP) enabled because both use the same port.

9.5. Using RARP

RARP (Reverse Address Resolution Protocol) is commonly used by diskless hosts to determine their Internet address. While ARP (Address Resolution Protocol) lets hosts resolve Internet addresses into Ethernet addresses, RARP lets them resolve Ethernet addresses into Internet addresses. Configuring the VSI TCP/IP RARP server consists of:

1. Obtaining the data needed by each RARP client (see Section 9.5.1).
2. On Ethernet interfaces only, enabling RARP packet reception (see Section 9.5.2).
3. Enabling and starting RARP (see Section 9.5.3).
4. Adding client systems to the RARP configuration file (see Section 9.5.4).
5. Reloading the RARP configuration (see Section 9.5.5).

Note

Because RARP clients send their requests in a link-layer broadcast (Ethernet, for example) and most routers do not forward link-layer broadcasts, make sure the VSI TCP/IP system and all its RARP clients are on the same physical network.

9.5.1. Obtaining Data for RARP Clients

Obtain the IP and Ethernet addresses for each client you want to use with RARP. To obtain a client's Ethernet interface address, refer to the interface documentation.

Ethernet addresses are expressed as six hexadecimal numbers (ranging from 0 to ff) separated by colons. IP addresses are expressed in dotted-decimal format.

9.5.2. Enabling RARP Packet Reception on Ethernet Interfaces

If your VSI TCP/IP system has an Ethernet interface, enable RARP packet reception with the **IP SET /INTERFACE** command. For example, to enable RARP packet reception on the se0 interface:

```
$ IP SET/INTERFACE/VMS_DEVICE=XQA0:/LINK_LEVEL=ETHERNET/RARP SE0
```

To automatically enable RARP packet reception when VSI TCP/IP starts, make sure to add the /**RARP** qualifier to the **IP SET /INTERFACE** command line in the custom initialization command procedure for the interface.

9.5.3. Enabling and Starting RARP Service

To enable RARP, use SERVER-CONFIG. To enable the RARP server with SERVER-CONFIG:

```
$ IP CONFIGURE /SERVER
VSI TCP/IP for OpenVMS Server Configuration Utility 10.5(nnn)
[Reading in configuration from IP$:SERVICES.MASTER_SERVER]
SERVER-CONFIG>ENABLE RARP
SERVER-CONFIG>EXIT
[Writing configuration to IP$COMMON_ROOT:[IP]SERVICES.MASTER_SERVER]
$
```

Once you have enabled RARP, start it by restarting the VSI TCP/IP Master Server:

```
$ @IP$:IP$SYSTARTUP.COM
```

9.5.4. Adding Clients to the RARP Configuration File

The VSI TCP/IP RARP server uses Ethernet-to-IP address translations from the `IP$:RARP.CONFIGURATION` file. Each single-line entry in `RARP.CONFIGURATION` contains an Ethernet address and the corresponding Internet address.

The following `RARP.CONFIGURATION` sample shows the IP addresses assigned to the hosts with Ethernet addresses aa:00:04:00:45:12 and 00:0c:00:17:12:67.

```
#   This is a sample RARP database. It provides the mapping between
#   ethernet addresses and IP addresses as used by RARP.
#
# ethernet address   ip address
# ----------------   ----------
 aa:00:04:00:45:12   192.0.0.1
 00:0c:00:17:12:67   192.0.0.2
```

9.5.5. Reloading RARP Configuration

After modifying the `IP$:RARP.CONFIGURATION` file, reload the RARP configuration with the following command:

```
$ IP NETCONTROL RARP RELOAD
```

9.6. Using BOOTP

The OpenVMS BOOTP (Bootstrap Protocol) service lets your OpenVMS system help diskless hosts and other network devices establish network connectivity. The remote system broadcasts a BOOTP request over the network with its Ethernet address. The BOOTP server looks up the host's address in a configuration file (`IP$:BOOTP-SERVER.CONFIGURATION`) and responds with the host's IP address, subnet mask, gateway address, initial load file, and any other data needed by the client. Using this information, the client can boot from the network.

Configuring the BOOTP server involves:

1. Obtaining the data required by each BOOTP client (see Section 9.6.1).
2. Enabling and starting BOOTP (see Section 9.6.2).
3. Modifying the BOOTP configuration file (see Section 9.6.3).
4. Reloading the BOOTP configuration (see Section 9.6.7).
5. Disabling debug messages, if desired (see Section 9.6.8).

Note

While BOOTP is often used with clients and servers on the same network, they can be on different physical networks. Most routers can be configured to forward BOOTP requests; refer to your router documentation.

9.6.1. Obtaining Data for BOOTP Clients

Make a list of the configuration parameters (known as BOOTP options) required by the devices you want to configure using BOOTP. Table 9.1 lists BOOTP options.

Because some network devices require large amounts of information or vendor-specific configuration at boot time, BOOTP lets you specify the path names of additional configuration files the client can download from TFTP servers. For details on creating downloadable configuration files for a specific host, refer to the vendor's documentation.

Note

If you are running DNS, make sure you use the same IP address and host name data used by your primary site's DNS servers. If you are using host tables instead of DNS, make sure you use the same IP address and host name data listed in `IP$:HOSTS.LOCAL`.

9.6.2. Enabling and Starting BOOTP

You can enable BOOTP with SERVER-CONFIG.

To enable BOOTP with SERVER-CONFIG, enter the following:

```
$ IP CONFIGURE /SERVER
VSI TCP/IP for OpenVMS Server Configuration Utility 10.5(nnn)
[Reading in configuration from IP$:SERVICES.MASTER_SERVER]
SERVER-CONFIG>ENABLE BOOTP
SERVER-CONFIG>EXIT
[Writing configuration to IP$COMMON_ROOT:[IP]SERVICES.MASTER_SERVER]
$
```

Note

BOOTP cannot run while DHCP is enabled because both services use the same port. You can use SERVER-CONFIG to disable DHCP.

After enabling BOOTP, start it by restarting the VSI TCP/IP Master Server:

```
$ @IP$:IP$SYSTARTUP.COM
```

9.6.3. Modifying the BOOTP Configuration File

VSI TCP/IP supplies a `IP$:BOOTP-SERVER.CONFIGURATION` file that contains comments and a number of examples to help you enter information for your hosts.

9.6.4. BOOTP Options for the BOOTP Server

Table 9.1 describes the options you can define for each host and an example of each option.

Table 9.1. BOOTP Options

Field	Description	Example
bf	File downloaded by TFTP to the client at boot time. This file is supplied by the device vendor. The file must exist and be world-readable. If the file is not found, a null file specification is returned.	bf="mom $load:xncd16_lt"
bs	Bootfile size. If the value is the string "auto" or no value is given, the server automatically determines the size. Otherwise, the specified value is passed verbatim. The size is expressed in 512-byte blocks.	bs=auto or bs=24
cs	Space-separated list of "quote of the day" server IP addresses. The cookie (as in "fortune cookie") server is described in RFC-865.	cs=192.41.228.92
ds	Space-separated list of domain name server IP addresses	ds=192.41.228.65
gw	IP address of the default gateway	gw=128.2.13.1
ha	Hardware address of the client. The format of the hardware address depends on the hardware type (**ht**). Specify the hardware type (**ht**) before the hardware address (**ha**).	ha=00DD00C88900
hd	Home directory for the boot files	hd="sys$sysroot: [bootp-boot files]"
hn	Flag requesting the host name to be sent to the client. When an entry contains this tag, the contents of the name field (the initial string of characters on each record up to, but not including the first colon) are sent to the client. If the name field is greater than 58 characters, only the host field (up to the first period) is sent. If the host field by itself does not fit, no value is sent.	hn
ht	Hardware address type. The hardware type must be interpreted before the hardware address (**ha**). Valid values are the hardware type, expressed as a decimal number	ht=ethernet, ht=6

Field	**Description**	**Example**
	as defined by the RFCs or a text string that maps to the hardware type number:	

ethernet	1	ieee803	6	chaos	5
ethernet3	2	tr	6	arcnet	7
ether	1	token-ring	6	ax.25	3
ether3	2	pronet	4		

im	Space-separated list of Imagen-type "Impress" server IP addresses	im=192.168.228.92 192.168.228.93
ip	IP address of the host	ip=192.168.228.82
lg	Space-separated list of MIT-LCS UDP log server IP addresses	lg=192.168.228.42
lp	Space-separated list of LPR server IP addresses	lp=192.168.228.37
ns	Space-separated list of IEN-116 name server IP addresses	ns=192.168.228.77
rl	Space-separated list of RLP (Resource Location Protocol) server IP addresses	rl=192.168.228.19
sa	IP address of a boot server	sa=192.168.228.222
sm	Subnetwork mask	sm=255.255.255.192
tc	Template host label. Use the **tc** field to "include" information from another entry in the configuration file. You may create a common entry for a group of hosts, such as a specific vendor's X terminals. Use the **tc** field to combine information specific to each model. Information in the current entry overrides information included by the **tc** field. A **tc** entry may also "include" another entry with a **tc** field of its own.	tc=global.dummy
td	TFTP directory. Used to reference part of a directory that may be hidden from the client via the TFTP server.	td="TFTP$DIR:"
to	Time offset (in seconds) east of GMT for the client. Table 9.2 lists accepted values. BOOTP uses negative numbers west of GMT and positive numbers east of GMT. See Table 9.2 for the time offset values you can specify in this field.	to=25200
ts	Space-separated list of time server IP addresses	ts=192.41.228.77
T*n*	"Generic" tag of the type "T*n*=*value*," • *n* is the number assigned the option. • *value* is either ASCII data enclosed in quotes or binary data expressed as hexadecimal digits. When expressing binary data that represents short or long values, be sure to check the byte order to compensate for the difference between OpenVMS byte order and network byte order. For values with known tags, the server can convert between the two. For values in generic tags, however, the	T123="Hello World" or T124=FFFE2CEF

Field	Description	Example
	server cannot tell the difference between a four-byte binary string and an unsigned long value.	
vm	"Vendor magic" to send: "auto", "rfc1048", "rfc1084", "cmu", or a dotted-decimal value. Vendor magic is always "rfc1084" when using DHCP. Default: auto.	vm="rfc1048"

Table 9.2 provides time offset values you can specify in the **to** field.

Table 9.2. BOOTP “to” Option Values

Timezone	Time Offset	DST Time Offset	Timezone	Time Offset	DST Time Offset
AST/ADT	-14400	-10800	MET/MET-DST	3600	7200
BST	0	3600	MST/MDT	-25200	-21600
CET/CET-DST	3600	7200	NST/NDT	-12600	-9000
CST/CDT	-21600	-18000	NZST	86400	90000
EET/EET-DST	10800	14400	PST/PDT	-28800	-25200
EST/EDT	-18000	-14400	SST	+28800	none
GMT	0	none	UTC	0	none
HST	-36000	none	WET/WET-DST	3600	7200
JST	32400	none	YST/YDT	-32400	-28800

9.6.5. Guidelines for the BOOTP Configuration File

The following guidelines govern modification of the `IP$:BOOTP-SERVER.CONFIGURATION` file:

- Edit the configuration file with any text editor.
- Use a pound sign (#) in the first column of the line to designate a comment line. Comment and blank lines are ignored by the server.
- Specify the hardware type (**ht**) before the hardware address (**ha**).
- Specify IP addresses in dotted-decimal notation.

Note

If you enter an IP address with leading zeros as part of the address (for example, 192.41.012.011), the octets with leading zeros are interpreted as octal values rather than decimal values.

- For readability, limit each entry to one line when possible. Otherwise, put each field on a separate line.
- Separate entry fields with a colon (:). When lines are continued on another line, separate fields with a colon followed by a backslash. You should start each new line with a tab followed by a colon. Here are examples of the two different entry styles:

```
    ncd16s:\
    :ht=ethernet:\
    :bf="mom$load:xncd16_lt":\
    :gw=192.41.228.71:\
    :sm=255.255.255.192:\
    :ds=192.41.228.65:\
    :to=25200:
tree:tc=ncd16s:ha=0000C0545F24:ip=192.41.228.75:
```

- Use the **tc** field as an "include" statement to succinctly provide additional information for an individual device, as shown in the example entries above. The entry called "tree" is for an individual NCD terminal. Including the **tc** option adds all of the information in the "ncd16s" entry to the "tree" entry.

  The **tc** field lets you create a common entry for a class of hosts (such as a vendor's X terminals) that conveys generic information. Entries that include **tc** options supply information specific to an individual terminal, such as its IP address.

  Information in the individual entry overrides the information included by the **tc** field.

- When specifying more than one server for the **cs**, **ds**, **im**, **lg**, **lp**, **ns**, **rl**, and **ts** fields, separate subsequent server values with spaces.

9.6.6. Using a UNIX bootptab File

If you are also running a BOOTP server on a UNIX system, you can use the UNIX system's bootptab configuration file after making the following changes:

- Copy the bootptab file to `IP$:BOOTP-SERVER.CONFIGURATION`.
- Change the syntax of directories and file names to OpenVMS format.
- Do not add names that conflict with existing entries.

9.6.7. Reloading the BOOTP Configuration

After modifying `IP$:BOOTP-SERVER.CONFIGURATION`, reload the BOOTP configuration with the following command:

```
$ IP NETCONTROL BOOTP RELOAD
```

9.6.8. Disabling BOOTP OPCOM Messages

After you test your BOOTP configuration, you may want to suppress some of the messages the BOOTP server sends to OPCOM by changing the debug level of the BOOTP server, as shown in this example:

```
$ IP NETCONTROL BOOTP DEBUG -1
```

If you want this change to take place each time VSI TCP/IP is started, use the SERVER-CONFIG **SET PARAMETERS** command as follows:

```
$ IP CONFIGURE /SERVER
VSI TCP/IP for OpenVMS Server Configuration Utility 10.5(nnn)
[Reading in configuration from IP$:SERVICES.MASTER_SERVER]
SERVER-CONFIG>SELECT BOOTP
```

```
[The Selected SERVER entry is now BOOTP]
SERVER-CONFIG>SET PARAMETERS
Delete parameter "bootfile IP$:BOOTP-SERVER.CONFIGURATION" ? [NO]
You can now add new parameters for BOOTP. An empty line terminates.
Add Parameter: debug -1
Add Parameter:
[Service specific parameters for BOOTP changed]
SERVER-CONFIG>RESTART
Configuration modified, do you want to save it first ? [YES]
[Writing configuration to IP$COMMON_ROOT:[IP]
SERVICES.MASTER_SERVER]
%RUN-S-PROC_ID, identification of created process is 20600046
SERVER-CONFIG>EXIT
[Configuration not modified, so no update needed]
$
```

9.7. Using DHCP

The VSI TCP/IP DHCP (Dynamic Host Configuration Protocol) server lets your OpenVMS system help diskless hosts and other network devices establish network connectivity. The DHCP server provides all of the functions of BOOTP plus dynamic addressing and additional configuration options.

The DHCP server offers a network host a temporary lease of an IP address rather than an ownership of an IP address, such as BOOTP does. The lease identifies the length of time the client can safely use its assigned IP address. The network administrator sets the lease length using parameters in the configuration file. It is recommended that the network administrator assign lease lengths based on the number of network users and the number of available IP addresses the DHCP server can assign. To configure the DHCP server:

1. Obtain the data required by each DHCP client (see Section 9.7.2).
2. Modify the DHCP configuration file (see Section 9.10).
3. Enable and start the DHCP server (see Section 9.7.3).
4. If you modify the configuration file after starting the DHCP server, reload the DHCP server (see Section 9.9).

Note

DHCP uses DNS for host names and IP addresses; thus, a malfunction in your DNS server can affect the DHCP server.

9.7.1. DHCP Process

DHCP goes through an initializing, selecting, requesting, binding, renewal, rebinding, and expiration cycle when negotiating for an IP address, as shown in Figure 9.1. The process is as follows:

1. The client just added or relocated on the network requests an IP address by broadcasting a DHCPDISCOVER message to the local subnet over the well-known BOOTP server port. (The client can also go through a BOOTP router or relay agent to forward the DHCPDISCOVER to additional remote DHCP servers.) This is the initializing state.
2. The participating DHCP servers respond with a DHCPOFFER message if they have a valid configuration for the client. The client may get many of these messages, which contain the IP

address and configuration data. (The servers make sure to reserve the addresses so as not to accidentally offer them to another client.) At this point the client enters the selecting state.

3. After selecting an address, the client broadcasts the selected address and name of the "winning" server (DHCP Server 1 in the Figure 9.1) using a DHCPREQUEST message. This is the requesting state. All the other servers can now safely unreserve their addresses.

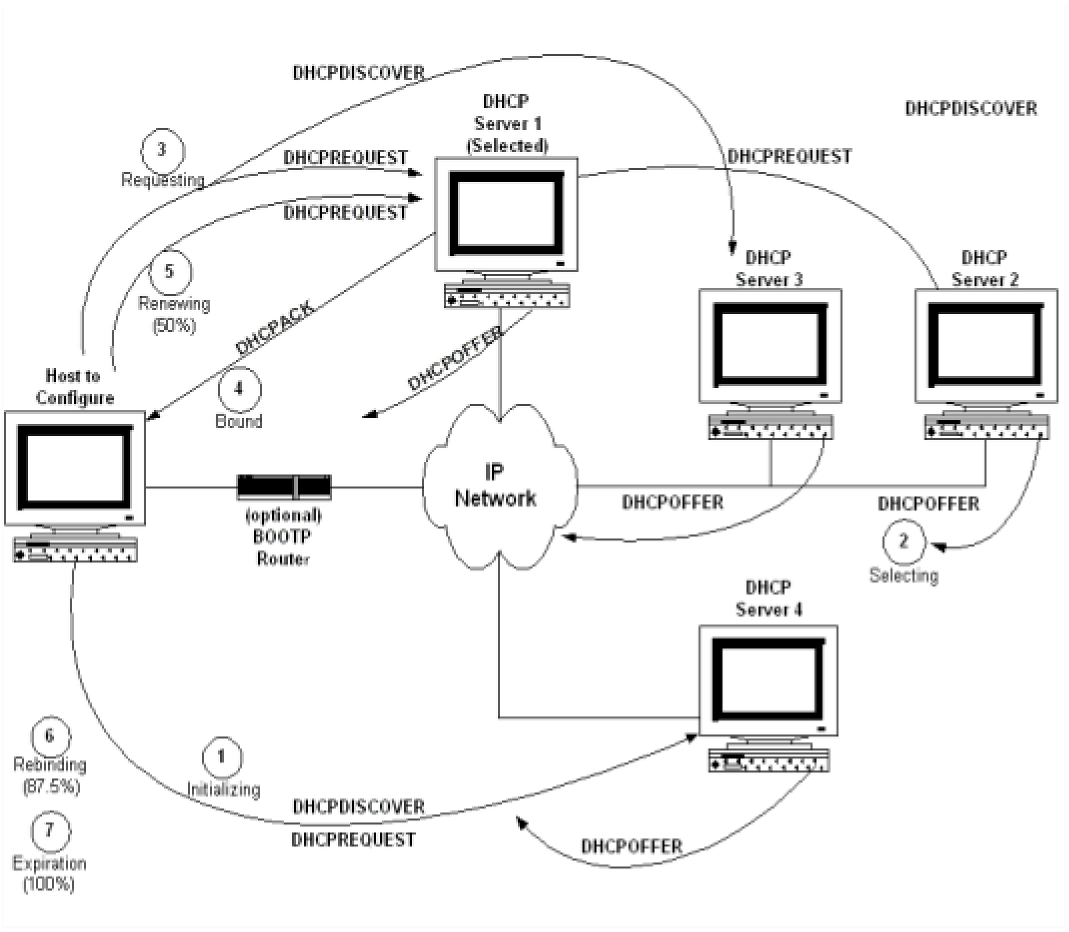
4. Server 1 sends the client a DHCPACK (acknowledgement) message with the negotiated IP address, the lease, and the network configuration parameters. The client now enters the binding state and can fully use the assigned IP address.

5. About halfway through the lease, the client sends Server 1 another DHCPREQUEST for a lease renewal, and enters the renewal state. If the server deems the lease renewable, it sends back another DHCPACK to update the lease (including any new parameters). The client now returns to the binding state, as in step 4.

6. If the client cannot renew the lease (such as if Server 1 is down), the client waits until about 87.5% of the way through the lease and broadcasts another DHCPREQUEST to all DHCP servers. Any server can now return a DHCPACK containing the extended lease and updated parameters. This is the rebinding state.

7. When the lease reaches 100% expired or a server sends back a DHCPNAK negative acknowledgement message, the client must give up the IP address. It then returns to the initializing state and has to start the address negotiation over again.

Figure 9.1. DHCP Address Request and Allocation Process



See the RFC 2131 and RFC 2132 for more information.

Two DHCP servers are recommended for a network. The benefit of having more than one server is if one fails another is available to continue processing requests, ensuring that all hosts (old and new) are serviced continuously. Refer to Section 9.20 for more information.

9.7.2. Obtaining Data for DHCP Clients

Make a list of the configuration parameters (known as DHCP options) required by the devices you want to configure using DHCP.

9.7.3. Enabling and Starting DHCP

You can enable the DHCP server with SERVER-CONFIG. To enable the DHCP server with SERVER-CONFIG, do the following:

```
$ IP CONFIGURE /SERVER
VSI TCP/IP for OpenVMS Server Configuration Utility 10.5(nnn)
[Reading in configuration from IP$:SERVICES.MASTER_SERVER]
SERVER-CONFIG>ENABLE DHCP
SERVER-CONFIG>EXIT
[Writing configuration to IP$COMMON_ROOT:[IP]SERVICES.MASTER_SERVER]
$
```

Note

DHCP cannot run while the BOOTP server is enabled because both servers use the same port. Because the DHCP server provides BOOTP service as well, there is no need to run the BOOTP service.

After you have enabled DHCP, start the DHCP server by restarting the VSI TCP/IP Master Server. If the DHCP server is already running, shut it down first.

```
$ IP NETCONTROL DHCP SHUTDOWN
$ @IP$:IP$SYSTARTUP.COM
```

9.8. Checking the DHCP Configuration

After modifying the configuration file, it is good practice to verify the syntax by running the DHCP server interactively specifying the `-t` flag, as follows:

```
$ dhcpd :== $IP$:dhcpd4.exe
$ dhcpd -t [-cf config-file]
```

You can test both the configuration file and the lease file using the -T flag:

```
$ dhcpd “-T” [-cf config-file] [-lf lease-file]
```

The -t flag causes the DHCP server to run just far enough to read and parse the configuration file. The DHCP server displays a copyright notice as well as a message for each syntax error encountered. If the DHCP server displays only the copyright notice, the configuration file has no syntax errors.

The -T flag causes the DHCP server to run just far enough to read and parse the configuration and lease files.

DHCPD can be made to use an alternate configuration file with the *-cf* flag, or an alternate lease file with the *-lf* flag. If you do not specify the *-cf* flag, the DHCP server reads the default configuration file `IP$:DHCPD.CONF`. If you do not specify the *-lf* flag, the DHCP server reads the default lease file `IP$:DHCPD.LEASES`. Because of the importance of using the same lease database at all times when running DHCPD in production, these flags should be used **only** for testing lease files or database files in a non-production environment.

9.9. Reloading the DHCP Configuration

If you modify `IP$:DHCPD.CONF` after starting the DHCP server, restart DHCP with the following command so the DHCP server rereads `DHCPD.CONF`:

```
$ IP NETCONTROL DHCP RESTART
```

9.10. Introducing the Configuration File

VSI TCP/IP supplies a `IP$:DHCPD.CONF` file that contains comments and a number of examples to help you enter information for your hosts. You can edit the configuration file with any text editor. Add or remove entries as needed.

The `DHCPD.CONF` file is a free-form ASCII text file. The file may contain extra tabs and newlines for formatting purposes. Keywords in the file are case-insensitive. Comments may be placed anywhere within the file (except within quotation marks), beginning with the # character *line*. See Example 9.3.

The file consists of a list of statements. Statements fall into two categories: *parameters* and *declarations*.

Parameter statements always say one of the following:

- How to do something (e.g., how long a lease to offer).
- Whether to do something (e.g., should the DHCP server provide addresses to unknown clients).
- What parameters to provide to the client (e.g., use gateway 220.177.244.7).

Global parameters are at the beginning of the file. Some examples of global parameters are the organization's domain name and the addresses of the name servers (if they are common to the entire organization).

It is legal to specify host addresses in parameters as domain names rather than as numeric IP addresses. If a given *hostname* resolves to more than one IP address (for example, if that host has two ethernet interfaces), both addresses are supplied to the client.

Both the *shared-network* statement and the *subnet* statement can have parameters.

The most obvious reason for having subnet-specific parameters is that each subnet, of necessity, has its own router. For example, something like:

```
option routers 204.254.239.1;
```

Note

The address here is specified numerically. This is not required. If you have a different domain name for each interface on your router, it is perfectly appropriate to use the domain name for that interface instead of the numeric address. However, there may be only one domain name for all of a router's IP addresses, and it would not be appropriate to use that name here.

Parameters starting with the `option` keyword correspond to actual DHCP options. Parameters that do not start with the `option` keyword either control the behavior of the DHCP server (e.g., how long a lease the DHCP server will give out), or specify client parameters that are not optional in the DHCP protocol (for example, server-name and filename).

Each host can have host-specific parameters. These could include such things as the:

- Hostname option.
- Name of a file to upload (the `filename` parameter).
- Address of the server from which to upload the file (the `next-server` parameter).

In general, any parameter can appear anywhere that parameters are allowed, and will be applied according to the scope in which the parameter appears.

All parameters must be specified first before you can specify any declarations that depend on those parameters. Parameters should be set inside declarations so they can be set on a per-subnet or a per-host basis.

`Declarations` are used to:

- Describe the topology of the network.
- Describe clients on the network.
- Provide addresses that can be assigned to clients.
- Apply a group of parameters to a group of declarations.

Declarations about network topology include the `subnet` and the `shared-network` declarations.

For every subnet to be served, and for every subnet connected to the DHCP server, there must be one `subnet` declaration. This declaration tells the DHCP server how to recognize that an address is on that particular subnet. A `subnet` declaration is required for each subnet even if no addresses will be dynamically allocated on that subnet.

There are different declarations required for different situations. The following is a list of the basic declarations in a configuration file.

- For clients with dynamically assigned addresses, a `range` declaration must appear within the `subnet` declaration, or a `pool` declaration.
- For clients with statically assigned addresses, or for installations where only known clients will be served, each client must have a `host` declaration.
- If parameters are to be applied to a group of declarations that are not related strictly on a per subnet, class, or pool basis, the `group` declaration can be used.

Some installations have physical networks allowing more than one IP subnet to operate. For example, if your site has a requirement that 8-bit subnet masks be used, but a department with a single physical ethernet network expands beyond 254 nodes, you may have to run two 8-bit subnets on the same ethernet until a new physical network is added. In this case, you can enclose the `subnet` declarations for these two networks in a `shared-network` declaration.

Some sites may have departments that have clients on more than one subnet, but it may be desirable to offer those clients a uniform set of parameters that are different than what would be offered to clients from other departments on the same subnet.

- For clients declared explicitly with *`host`* declarations, enclose these declarations in a *`group`* declaration using the parameters that are common to that department.
- For clients with dynamically assigned addresses, one way to group parameter assignments is by network topology. Alternately, host declarations can provide parameters and if they have no fixed-address. See Example 9.1 for more information.
- Clients can be grouped into *`classes`* and assigned IP addresses from specific *`pools`*.

When a client is to be booted, its boot parameters are determined by consulting the following *`scopes`* in this order:

1. Client's *`host`* declaration (if any).
2. *`Group`* declaration (if any) that enclosed the host declaration.
3. *`Subclass`* declaration for the subclass the client belongs to (if any).
4. *`Class`* declaration for the class the client belongs to (if any).
5. *`Pool`* declaration that the assigned IP address comes from (if any).
6. *`Subnet`* declaration for the subnet on which the client is booting.
7. *`Shared-network`* declaration (if any) containing that subnet.
8. Top-level parameters that may be specified outside of any declaration.

When searching for a *`host`* declaration, the DHCP server looks for one with a fixed-address parameter that matches the subnet or shared network on which the client is booting.

Imagine that you have a site with a lot of NCD X-Terminals. These terminals come in a variety of models, and you want to specify the boot files for each model. One way to do this would be to have *`host`* declarations for each server and group them by model:

Example 9.1. Host Declarations

```
group {
 filename "Xncd19r";
 next-server ncd-booter;
 host ncd1 { hardware ethernet 0:c0:c3:49:2b:57; }
 host ncd4 { hardware ethernet 0:c0:c3:80:fc:32; }
 host ncd8 { hardware ethernet 0:c0:c3:22:46:81; }
}
group {
 filename "Xncd19c";
 next-server ncd-booter;
 host ncd2 { hardware ethernet 0:c0:c3:88:2d:81; }
 host ncd3 { hardware ethernet 0:c0:c3:00:14:11; }
}
group {
 filename "XncdHMX";
 next-server ncd-booter;
 host ncd1 { hardware ethernet 0:c0:c3:11:90:23; }
 host ncd4 { hardware ethernet 0:c0:c3:91:a7:8; }
 host ncd8 { hardware ethernet 0:c0:c3:cc:a:8f; }
}
```

9.10.1. Address Allocation

Address allocation is done when a client is in the INIT state and has sent a DHCPDISCOVER message. When the DHCP server is looking for an IP address to allocate to a client, it checks first

- if the client has an active lease on an IP address, or
- if the client has an expired lease on an IP address that has not been reassigned.

It then follows these rules:

- If a lease was found but the client is not permitted to use it, then the lease is freed (if it was not expired already).
- If no lease is found or a lease was found and the client is not permitted to use the address, then the server looks for an address that is not in use and that the client is permitted to have among the list of address pools on the client's subnet.
- If no addresses are found that can be assigned to the client, then no response is sent to the client.
- If an address is found that the client is permitted to have, then the address is allocated to the client.

Note

IP addresses that have never been assigned are chosen over those that have previously been assigned to other clients.

If the client thinks it has a valid lease and sends a DHCPREQUEST to initiate or renew that lease, the server has three choices. It can

- Ignore the DHCPREQUEST.
- Send a DHCPNAK, telling the client to stop using the address.
- Send a DHCPACK, telling the client to use the address.

If the server finds the requested address and that address is available to the client, the server sends a DHCPACK.

If the address is no longer available or the client is not permitted to have it, the server sends a DHCPNAK.

If the server knows nothing about the address, the server remains silent. However, if the address is incorrect for the network segment to which the client is attached and the server is authoritative for that segment, the server sends a DHCPNAK.

9.10.2. Address Pools

Pool declarations let you have different allocation policies for different address allocation pools. A client may be denied access to one pool, but allowed access to another pool on the same network segment.

A *pool* declaration is used to specify how a group of addresses should be treated differently than another group of addresses, even if they are on the same network segment or subnet.

For example, you can provide a large set of addresses assigned to DHCP clients that are known to your DHCP server, while at the same time providing a small set of addresses that are available for unknown clients. If you have a firewall, you can arrange for addresses from one pool to have access to the Internet, while addresses in another pool do not have access to the Internet. The following example illustrates how you could set up a pair of `pool` declarations.

```
subnet 10.0.0.0 netmask 255.255.255.0 {
 option routers 10.0.0.254;
 # Unknown clients get this pool.
 pool {
 option domain-name-servers bogus.example.com;
 max-lease-time 300;
 range 10.0.0.200 10.0.0.253;
 allow unknown clients;
 }
 # Known clients get this pool.
 pool {
 option domain-name-servers ns1.example.com, ns2.example.com;
 max-lease-time 28800;
 range 10.0.0.5 10.0.0.199;
 deny unknown clients;
 }
}
```

You can also set up entirely different subnets for known and unknown clients. This is possible because address pools exist at the level of shared networks, so address ranges within pool declarations can be on different subnets, as long as they are on the same shared network.

9.10.3. Pool Permit Lists

The above example shows that address pools can have permit lists. A permit list controls which clients are allowed access to the address pool and which clients are not allowed access. Each entry in a permit list is introduced with the *`allow`* or *`deny`* keyword. The following table describes the four possibilities for eligibility to addresses from the address pool.

If a pool has...	**Then...**
a permit list	only those clients that match specific entries on the permit list are eligible for addresses from the pool.
a deny list	only those clients that do not match any entries on the deny list are eligible for addresses from the pool.
both a permit list and a deny list	only clients that match the permit list and do not match the deny list are eligible for addresses from the pool.
neither a permit list nor a deny list	all clients are eligible for addresses from the pool.

`Range` declarations that appear outside of *`pool`* declarations in the same shared-network are grouped into two pools: one which allows all clients for *`range`* statements with the *`"dynamic-bootp"`* keyword and one which denies dynamic-bootp clients for *`range`* statements without the *`"dynamic-bootp"`* keyword.

The DHCP server checks each IP address to see if the client is permitted to use it, in response to both DHCPDISCOVER and DHCPREQUEST messages. The DHCP server checks both the address pool permit lists and the relevant in-scope allow and deny statements.

Recognized allow and deny statements can be used to permit or refuse access to known or unknown clients, members of a class, dynamic bootp clients, or all clients.

Note

The DHCP v2.0 style allow and deny statements (e.g., *`allow/deny unknown-clients`*) and *`range`* statement *`dynamic-bootp`* keyword do not mix well with pool permit lists. A v2.0-style *`deny`* statement overrides the pool permit lists, and the *`dynamic-bootp`* keyword is ignored inside of pools. Note also that the default for *`dynamic-bootp`* changes from *`deny`* to *`allow`* when pools are used.

9.11. Client Classing

You can separate clients into classes, treating each client differently depending on what class it is in. To separate clients into classes, use conditional statements (see Section 9.12) or a *`match`* statement within a *`class`* declaration. You can specify a limit on the total number of clients within a particular class or subclass that may hold leases at one time using the *`lease limit`* statement. You can specify automatic subclassing based on the contents of the client packet using the *`spawn with`* statement.

To add clients to classes based on conditional evaluation, write a conditional statement to match the clients you want in the class. Then, put an *`add`* statement in the conditional's list of statements. For example, to identify requests coming from Microsoft(R) NT RAS servers:

```
if substring (option dhcp-client-identifier, 1, 3) = "RAS" {
 add "ras-clients";
}
```

An equivalent way to do this is to specify the conditional expression as a matching expression in the *`class`* statement. For example:

```
class "ras-clients" {
 match if substring (option dhcp-client-identifier, 1, 3) = "RAS";
}
```

Note

Whether you use matching expressions or *`add`* statements (or both) to classify clients, you must write a *`class`* declaration for any class that you use.

If you want no *`match`* statement and no in-scope statements for a class, the declaration looks like this, for example:

```
class "ras-clients" {
}
```

Important

The add statement adds the client to the class after the address assignment has been completed. This means the client will not be affected by pool permits related to that class if the client is a member of a class due to an add statement.

9.11.1. Subclasses

In addition to classes, you can declare subclasses. A subclass is a class having the same name as a regular class but with a specific submatch expression that is hashed for quick matching. It is quicker to find five subclasses within one class than it is to find five classes with match expressions. The following example illustrates how to code for subclasses:

```
class "allocation-class-1" {
 match hardware;
}
class "allocation-class-2" {
 match hardware;
}
subclass "allocation-class-1" 1:0:0:c4:aa:29:44;
subclass "allocation-class-1" 1:8:0:2b:4c:39:ad;
subclass "allocation-class-2" 1:8:0:2b:a9:cc:e3;

subnet 10.0.0.0 netmask 255.255.255.0 {
 pool {
  allow members of "allocation-class-1";
  range 10.0.0.11 10.0.0.50;
 }
 pool {
  allow members of "allocation-class-2";
  range 10.0.0.51 10.0.0.100;
 }
}
```

The data following the class name in the `subclass` declaration is a constant value used in matching the match expression for the class. During class matching, the server evaluates the match expression and looks up the result in the hash table. If a match if found, the client is considered a member of both the class and the subclass.

You can specify subclasses with or without scope (i.e., statements). In the above example, the sole purpose of the subclass is to allow some clients access to one address pool, while other clients are given access to the other pool. Thus, these subclasses are declared without any statements (scope). If you wanted to define different parameter values for some clients, you would declare those subclasses with scopes.

For example: if you had a single client needing some configuration parameters, while most did not, you might write the following `subclass` declaration for that client:

```
subclass "allocation-class-2" 1:08:00:2b:a1:11:31 {
 option root-path "samsara:/var/diskless/alphapc";
 filename "/tftpboot/netbsd.alphapc-diskless";
}
```

In the previous examples, subclassing is being used as a way to control address allocation on a per-client basis. However, it is possible to use subclassing in ways that are not specific to clients. For example, to use the value of the `vendor-class-identifier` option to determine what values to send in the `vendor-encapsulated-options` option. See Section 9.17.4.

Note

If you are using <`match hardware`>, the hardware address is preceded by the hardware type. In this example, the “1:” indicates Ethernet.

9.11.2. Per-Class Limits on Dynamic Address Allocation

The number of clients in a class that can be assigned leases can be limited. This limiting makes it difficult for a new client in a class to get an address. Once a class has reached its limit, the only way a new client in that class can get a lease is for an existing client to relinquish its lease, either by

- letting it expire, or
- sending a DHCPRELEASE packet.

The following example illustrates how to specify classes with lease limits.

```
class "limited-1" {
 lease limit 4;
}
```

This produces a class in which a maximum of four members may hold leases at one time.

If you want to provide clients at a particular site with more than one IP address, but do not want to provide these clients with their own subnet, nor give them an unlimited number of IP addresses from the network segment to which they are connected, you can create a spawning class and use lease limits. A spawning class is a class that produces subclasses automatically based on what the client sends.

Many cable modem head-end systems can be configured to add a Relay Agent Information option to DHCP packets when relaying them to the DHCP server. These systems typically add a circuit ID or remote ID option that uniquely identifies the customer site. The following example illustrates how to write a class declaration to take advantage of these relay agent options to create lease limited classes on the fly:

```
class "customer" {
 match if exists agent.circuit-id;
 spawn with option agent.circuit-id;
 lease limit 4;
}
```

With this class declaration, whenever a request comes in from a customer site, the circuit ID option is checked against the class's hash table.

- If a subclass matches the circuit ID, the client is classified in that subclass.
- If no subclass matches the circuit ID, a new subclass is created and logged in the `dhcpd.leases` file and the client is classified in the new subclass.

Once a client is classified, it is treated according to the rules of the class; as in the example above, being subjected to the per-site limit of four leases.

Note

The use of the subclass spawning mechanism is not restricted to relay agent options. This particular example is given only because it is a straightforward one.

9.12. Conditional Behavior

The DHCP server can be configured to perform conditional behavior depending on the packets it receives.

Conditional behavior is specified using the `if` statement and the `else` or `elsif` statements. A conditional statement can appear anywhere that a regular statement can appear, and can enclose one or more such statements. The following is an example of a conditional statement.

```
if option dhcp-user-class = "accounting" {
 max-lease-time 17600;
 option domain-name "accounting.example.org";
 option domain-name-servers ns1.accounting.example.org,
    ns2.accounting.example.org;
} elsif option dhcp-user-class = "engineering" {
 max-lease-time 17600;
 option domain-name "engineering.example.org";
 option domain-name-servers ns1.engineering.example.org,
   ns2.engineering.example.org;
} else {
 max-lease-time 600;
 option domain-name "misc.example.org";
 option domain-name-servers ns1.misc.example.org,
   ns2.misc.example.org;
}
```

Both the `if` statement and the `elsif` continuation statement take expressions that, when evaluated, produce a boolean result. See Section 9.16 for more information.

- If the expression evaluates to true, then the statements enclosed in braces following the `if` statement are executed. All subsequent `elsif` and `else` clauses are skipped.
- If the expression evaluates to false, then the statements enclosed in braces following the `if` statement are not executed and each subsequent `elsif` clause is checked until an `elsif` clause is encountered that evaluates to true.
- If such an `elsif` clause is found, then the statements in braces following it are executed. Any subsequent `elsif` and `else` clauses are skipped.
- If all the `if` and `elsif` clauses are checked but none of their expressions evaluate to true, then if there is an `else` clause, then the statements enclosed in braces following the `else` clause are evaluated.

Note

Boolean expressions that evaluate to null are treated as false in conditionals.

9.13. DNS Dynamic Updates Within DHCP

The DHCP server performs dynamic updates to DNS using DNS's dynamic updating functionality. To be sure that updates are allowed from the DHCP server, see the *VSI TCP/IP Administrator's Guide: Volume I*. The `allow-update { <address_match_list> };` statement in the Zone section enables the DNS server to allow updates from that system.

The following statements in the DHCP server's configuration file are related to dynamic updating:

- allow/deny dynamic-update;
- allow/deny update-A-record;
- allow/deny name-by-client;
- invalid-ddns-chars {fail | discard | replace [“chars”]};

Dynamic updates can be enabled or disabled by using the *allow/deny dynamic-update* statement in the configuration file. The default is to not perform dynamic updates. Dynamic updates can be turned on or off on a per subnet basis.

Note

Dynamic updates are not done at all for static assignments to BOOTP clients, and the support for static assignments to DHCP clients is to add DNS entries only.

When dynamic updating is enabled, the DHCP server determines the client's Fully Qualified Domain Name (FQDN) and assigns it an IP address. The FQDN is determined either by what the client sends or by what is in the configuration file. This behavior is controlled by the *`allow/deny name-by-client`* statement in the configuration file.

If you use the *`deny name-by-client`* statementor if the client does not send a name, you must specify the host name in the configuration file using one of the following methods:

- Using *`option host-name “name”`* (see Section 9.14)
- Specifying *`use-host-decl-names on`* in conjunction with host declarations.

If the hostname specified by the client contains invalid characters for DNS, the DHCP server can handle them one of three ways:

- Consider it a failure and not do the dynamic update.
- Throw away the invalid characters.
- Replace the invalid characters with specified valid characters.

This behavior is controlled by the *`invalid-ddns-chars`* statement in the configuration file.

The FQDN and IP address are used in the dynamic update to create a PTR resource record (RR). The DHCP server also optionally creates an A RR. This option is enabled or disabled by using the *`allow/deny update-A-record`* statement in the configuration file. The default is to not create the A RR. This can be set on a per subnet basis.

When dynamic updating is allowed, the DHCP server adds the resource records whenever an IP address is leased to a client. The RRs are deleted if the IP address is released or if the IP address is leased to a different client. Also, the RRs are deleted for expired leases periodically.

9.13.1. Transaction Signatures (TSIG)

The DHCP server supports using Transaction Signatures (TSIG) on dynamic updates to DNS. Note that you need a DNS server that supports TSIG, such as VSI TCP/IP BIND 8.2 server.

The use of TSIG can be enabled or disabled by using the *`secure-ddns`* statement in the configuration file. The default is to not use TSIG. The use of TSIG can be turned on or off on a per subnet basis. Turn on the use of TSIG using:

```
secure-ddns on;
```

For each DNS server that you want to use TSIG with, you must specify a key using the `key` declaration:

```
key ip-address {
[ algorithm "hmac-md5"; ]
key-id "key-name";
secret "key";
}
```

- ip-address is the IP address of the DNS server
- `algorithm` specifies the algorithm to use. The only supported algorithm is "hmac-md5". This statement is optional.
- `key-id` specifies the name of the key as a string. This must match the key name being used by the DNS server (e.g., configured in named.conf).
- `secret` specifies the secret key to use in base-64 format. This must match the secret key used by the DNS server (in named.conf).

An example `key` declaration for the DNS server at IP address 10.9.8.7 is:

```
key 10.9.8.7 {
key-id "dhcp-tsig";
secret "A5vhC+DjsocELGEYhj0iBBSQRgJvxnY/emD0C3kRtEpo";
};
```

9.14. Host Name Generation

Some DHCP clients require that the server send them a host name. The VSI TCP/IP DHCP server can generate a host name if it cannot get the host name in another way. The generated host name can contain parts of the host's IP address, client ID, and/or MAC address. This host name is sent to the client and is combined with the domain name to create the Fully Qualified Domain Name (FQDN) required for dynamic DNS updates. See Section 9.13 for more information.

The DHCP server generates a host name if it is enabled to do so and either

- `allow name-by-client` is specified and the client does not send a host name.

or

- `deny name-by-client` is specified and the DHCP server does not find a host name in the configuration file or in DNS (if `get-lease-hostnames` is set).

To enable the DHCP server to generate host names, specify in the configuration file an `option host-name` statement with a value containing certain key values in addition to any characters that are valid for the `host-name` option (see Table 9.4). The `option host-name` statement can be specified for example at the top level, in a `subnet` statement, or in a `host` statement.

The key values are as follows. You can include more than one in the same host-name value.

Note

Some of these do not by themselves generate a unique identifier.

Key	Meaning
%A	First byte of the host's IP address. Example: for address 10.24.25.201, the key would return 10.
%B	Second byte of the host's IP address. Example: for address10.24.25.201, the key would return 24.
%C	Third byte of the host's IP address. Example: for address 10.24.25.201, the key would return 25.
%D	Fourth byte of the host's IP address. Example: for address 10.24.25.201, the key would return 201.
%H	Host part of the host's IP address. Example: for address 10.24.25.201 with subnet mask 255.255.0.0, the key would return 6601.
%I	Client Identifier sent by the host. (in hex). For example: 0174657374.
%-I	Client ID as above, except that hyphens (-) are used to separate each byte.
%M	MAC address of the host.
%-M	MAC address of the host, as above, except that hyphens (-) are used to separate each byte.
%N	Host name sent by the client, if any. If none, "Host".
%P	Printable characters from the client ID. For example: if the client ID was 0174657374, the 01 is thrown away and the resulting hostname is "test".
%S	Subnet part of the host's IP address. Example: for address 10.24.25.201 with subnet mask 255.255.0.0, the key would return 102400.
%-S	Subnet part of the host's IP address, as above, except that hyphens (-) are used to separate each byte. For example: 10-24-0-0.

You can intersperse string constants such as hyphens between key definitions. However, if the generated host name exceeds 63 characters, it is truncated. Here is an example host-name statement:

```
option host-name "Host%H-%-S";
```

For a lease pool defined with an address range of 192.168.11.6 through 192.168.11.10 and a subnet mask of 255.255.255.0, the DHCP server generates the following host names:

```
Host6-192-168-11-0
Host7-192-168-11-0
Host8-192-168-11-0
Host9-192-168-11-0
Host10-192-168-11-0
```

The %N key allows you to use the host name as sent by the client (option 12) and then add something unique to it to generate a unique name. For example, if multiple clients all send the name "dilbert" you can make them unique by appending the MAC (hardware) address, as follows:

```
deny name-by-client;
```

```
option host-name "%N-%M";
```

This would generate the host name "dilbert-010203040506" for a client with hardware address 01:02:03:04:05:06.

9.15. Configuration File Declarations and Parameters

Table 9.3 describes the declarations and parameters you can use in a configuration file.

See Table 9.10 for a list of DHCP Safe-failover-related configuration file statements.

Table 9.3. DHCP Statements

Statement	Description
add	Use the `add` statement to add a client to the class whose name is specified in *`class-name`*. **Note** Because this statement executes after IP address allocation is completed, class membership caused by this statement cannot be used in the address allocation process. `add "class-name";`
algorithm	Used only inside of `key` declarations, the `algorithm` statement specifies the algorithm to use for *Transaction Signatures (TSIG)* on dynamic DNS updates. The only supported algorithm is "hmac-md5". This statement is optional. `algorithm "hmac-md5";`
allow and deny	Use the `allow` and `deny` statements to control the behavior of the DHCP server. The allow and deny keywords have different meanings depending on the context. • In a pool context, use these keywords to set up access lists for address allocation pools. • In other contexts, use these keywords to control general server behavior with respect to clients based on scope.
allow and deny in scope	These allow and deny statements work the same way whether the client is sending a DHCPDISCOVER or a DHCPREQUEST message, • an address is allocated to the client (either the old requested address or a new address), and then, • that address is tested to see if it is okay for the client to have it. If the client requested it, and it is not okay, the server sends a DHCPNAK message. Otherwise, the server does not respond to the client. If it is okay to give the address to the client, the server sends a DHCPACK message.

Statement	Description

	Note These are not recommended for use inside pool declarations. See Section 9.10.3 for more information.
	
	Use the *unknown-clients* flag to tell the DHCP server to dynamically assign addresses to unknown clients or to not assign addresses to unknown clients. An unknown client is one that does not have a *host* declaration. The default is to *allow* dynamic address assignments to unknown clients.
	
	`allow unknown-clients; deny unknown-clients;`
	
	Use the *bootp* flag to tell the DHCP server to respond to bootp queries or to not respond to bootp queries. The default is to *allow* bootp queries.
	
	`allow bootp; deny bootp;`
	
	Use the *dynamic-bootp* flag to tell the DHCP server to dynamically assign addresses to bootp clients or to not do so. The default is to *allow* dynamic bootp for IP addresses declared in pool declarations. The default for *range* statements outside of pool declarations is set by the presence or absence of the *dynamic-bootp* keyword. *Deny dynamic-bootp* overrides the dynamic-bootp range key word.
	
	`allow dynamic-bootp; deny dynamic-bootp;`
	
	Use the *booting* flag to tell the DHCP server to respond to queries from a particular client or to not respond to queries from a particular client. The default is to *allow* booting. If it is disabled for a particular client, that client will not be able to get an address from the DHCP server.
	
	`allow booting; deny booting;`

allow and deny in scope (cont'd)	Use the *dynamic-update* flag to tell the DHCP server to perform dynamic DNS updates or to not perform them. The default is to *deny* dynamic DNS updates.
	
	`allow dynamic-update; deny dynamic-update;`
	
	Use the *name-by-client* flag to tell the DHCP server to determine the hostname and Fully Qualified Domain Name (FQDN) for dynamic DNS updates from information sent by the client or from information in the configuration file. The default is to *deny* use of client-specified information.
	
	`allow name-by-client; deny name-by-client;`
	
	Use the *dhcpinform* flag to tell the DHCP server to respond to DHCPINFORM messages or to not respond. The default is to *allow* DHCPINFORM messages for authoritative subnets, and to *deny* DHCPINFORM messages for non-authoritative subnets.
	
	`allow dhcpinform; deny dhcpinform;`

Statement	Description

	Use the *update-A-record* flag to tell the DHCP server to update the A resource record or not when performing DNS updates (the PTR resource record is always updated). The default is to *deny* updating the A resource record.
	
	`allow update-A-record; deny update-A-record;`
	
	Use the *ras-servers* flag to tell the DHCP server to respond to queries from Microsoft (R) NT RAS Servers or to not respond to NT RAS queries. The default is to *allow* NT RAS queries.
	
	`allow ras-servers; deny ras-servers;`
	
	Allow/deny ras-servers is supported for backward compatibility. The way to do *deny ras-servers* in version 3.0 of DHCP is to use a conditional statement:
	
	`if substring (option dhcp-client-identifier, 1,3) = “RAS” {`
	`  deny booting;`
	`}`

allow and deny in pool declarations	See Section 9.10.3 for discussion, defaults, and important notes.
	
	Use *known clients* to allow or prevent allocation from this pool to any client that has a host declaration. A client is known if it has a host declaration in **any** scope.
	
	`allow known clients; deny known clients;`
	
	Use *unknown clients* to allow or prevent allocation from this pool to any client that has no host declaration.
	
	`allow unknown clients; deny unknown clients;`
	
	Use *members of “class”* to allow or prevent allocation from this pool to any client that is a member of the named class.
	
	`allow members of “class-name”;`
	`deny members of “class-name”;`
	
	Use *dynamic bootp clients* to allow or prevent allocation from this pool to any BOOTP client.
	
	`allow dynamic bootp clients;`
	`deny dynamic bootp clients;`
	
	Use *all clients* to allow or prevent allocation from this pool to all clients. You can use this, for example, when you want to write a pool declaration but you want to hold it in reserve; or when you want to renumber your network quickly, and thus want the server to force all clients that have been allocated addresses from this pool to obtain new addresses immediately when they next renew their leases.
	
	`allow all clients; deny all clients;`

always-broadcast	Use the *always-broadcast* statement to cause the DHCP server to always broadcast its responses. This feature is to handle clients who do not set the broadcast flag in their requests and yet require a broadcast response. We recommend you restrict the use of this feature to as few clients as possible.

Statement	Description
	`always-broadcast flag;`
always-reply-rfc1048	Some BOOTP clients expect RFC 1048-style responses, but do not follow RFC 1048 rules when sending their requests. You can determine if a client is having this problem: • if it is not getting the options you have configured for it. and • if you see in the server log the message "(non-rfc1048)" printed with each BOOTREQUEST that is logged. If you want to send RFC 1048 options to this kind of client, set the *`always-reply-rfc1048`* option in that client's host declaration. The DHCP server responds with an RFC 1048-style vendor options field. This flag can be set in any scope, and affects all clients covered by that scope. `always-reply-rfc1048 flag;`

[not] authoritative	When the DHCP server receives a DHCPREQUEST message from a DHCP client requesting a specific IP address, the DHCP protocol requires that the server determine whether the IP address is valid for the network to which the client is attached. If the address is not valid, the DHCP server should respond with a DHCPNAK message, forcing the client to acquire a new IP address.
	
	To make this determination for IP addresses on a particular network segment, the DHCP server must have complete configuration information for that network segment. Unfortunately, it is not safe to assume that DHCP servers are configured with complete information. Therefore, the DHCP server normally assumes that it does not have complete information, and thus is not sufficiently authoritative to safely send DHCPNAK messages as required by the protocol.
	
	This default assumption should not be true for any network segment that is in the same administrative domain as the DHCP server. For such network segments, the
	
	`authoritative`
	
	statement should be specified, so that the server sends DHCPNAK messages as required by the protocol. If the DHCP server receives requests only from network segments in the same administrative domain, you can specify the
	
	`authoritative`
	
	statement at the top of the configuration file (in the global scope).
	
	Note
	
	Version 2.0 of the DCHP server makes the opposite assumption: that the DCHP server is configured with all configuration information for all network segments of which it is aware. If this assumption is not valid for your configuration, you must write *`not authoritative`* statements for all network segments where this assumption is not true (or at the top of the configuration file).
	
	`authoritative;`

Statement	Description
	`not authoritative;`
class	This statement groups clients together based on information they send. A client can become a member of a class in the following ways: • through an *`add`* statement • based on the class's matching rules • because the client matches a subclass of that class *`Class-name`* is the name of the class and is used in: • *`add`* statements • *`members of`* permit statements • *`subclass`* declarations for subclasses of the named class When a packet is received from a client, every *`class`* declaration is examined for a *`match`*, *`match if`*, or *`spawn`* statement. That statement is checked to see if the client is a member of the class. The class declaration statements are *`lease limit`*, *`match`*, *`match if`*, and *`spawn with`*. `class "`*`class-name`*`" {[ statements ][ declarations ]}`
default-lease-time	*`Time`* is the length (in seconds) that the DHCP server assigns to a lease if the requesting client did not ask for a specific amount of time for the lease to be active. The infinite lease value is "infinite". The default is 43,200 seconds (12 hours). You should set the value of default-lease-time NO larger than the value of max-lease-time. `default-lease-time time;`
dynamic-bootp-lease-cutoff	Use the *`dynamic-bootp-lease-cutoff`* statement to set the ending time for all leases dynamically assigned to BOOTP clients. By default, the DHCP server assigns infinite leases to all BOOTP clients because they do not have any way of renewing leases, and do not know that their leases could expire. However, it may make sense to set a cutoff date for all BOOTP leases. For example, the end of a school term, or the time at night when a facility is closed and all machines are required to be powered off. *`Date`* should be the date all assigned BOOTP leases will end. The date is specified in the form: **W YYYY/MM/DD HH:MM:SS** where: **W** is the day of the week, from zero (Sunday) to six (Saturday). **YYYY** is the year, including the century.

Statement	Description
	MM is the number of the month, from 01 to 12. **DD** is the day of the month, counting from 01. **HH** is the hour, from 00 to 23. **MM** is the minute, from 00 to 59. **SS** is the second, from 00 to 59. The time is always in Greenwich Mean Time, not local time. `dynamic-bootp-lease-cutoff date;`
dynamic-bootp-lease-length	Use the *`dynamic-bootp-lease-length`* statement to set the length of leases dynamically assigned to BOOTP clients. You may be able to assume that a lease is no longer in use if its holder has not used BOOTP or DHCP to get its address within a certain time period. The length of the time period is your judgment call. Specify *`length`* in seconds. The infinite lease value is “infinite”. If a BOOTP client reboots during a timeout period, the lease duration is reset to *`length`* so a BOOTP client that boots frequently never loses its lease. This parameter should be adjusted with extreme caution. The default is an infinite lease. `dynamic-bootp-lease-length length;`
filename	Use the *`filename`* statement to specify the name of the initial boot file that is to be loaded by a client. The *`filename`* should be recognizable to whatever file transfer protocol the client can be expected to use. `filename filename;`
fixed-address	To make a static IP address assignment for a client, the client must match a *`host`* declaration, as described later. In addition, the *`host`* declaration must contain a *`fixed-address`* declaration. A *`fixed-address`* declaration specifies one or more IP addresses or domain names that resolve to IP addresses. If a client matches a *`host`* declaration, and one of the IP addresses specified in the *`host`* declaration is valid for the network segment to which the client is connected, the client is assigned that IP address. A static IP address assignment overrides a dynamically assigned IP address that is valid on that network segment. That is, if a new static mapping for a client is added after the client has a dynamic mapping, the client cannot use the dynamic mapping the next time it tries to renew its lease. The DHCP server will not assign an IP address that is not correct for the network segment to which the client is attached and will not override a valid dynamic mapping for one network segment based on a static mapping that is valid on a different network segment. You can specify a domain name instead of an IP address in a *`fixed-address`* declaration. However, you should do this only for long-lived domain name records — the DHCP server only looks up the record on startup. So, if the record changes while the server is running, the server continues to use the record’s former value. `fixed-address address [,...,address];`

Statement	Description
get-lease-hostnames	Use the *`get-lease-hostnames`* statement to tell the DHCP server to look up the domain name corresponding to each address in the lease pool and use that address for the DHCP hostname option. If *`flag`* is true, the lookup is done for all addresses in the current scope. If *`flag`* is false (the default), lookups are not done. `get-lease-hostnames flag;`
group	Use the *`group`* statement to apply one or more parameters to a group of declarations. You can use it to group hosts, shared networks, subnets, or other groups. `group {[statements] [declarations]}`
hardware	Use the *`hardware`* clause inside a *`host`* statement to specify the network hardware address of a BOOTP or DHCP client. *`hardware-type`* must be the name of a physical hardware interface type. Ethernet, Token-Ring, and FDDI are the only recognized types. The *`hardware-address`* should be a set of hexadecimal octets (numbers from 0 through ff) separated by colons (:). hardware hardware-type hardware-address;
host	The *`host`* declaration provides information about a particular client. *`Name`* should be a unique name for the *`host`* declaration, but a specific meaning is not required. If the *`use-host-decl-names`* flag is enabled, *`name`* is sent in the *`host-name`* option if no *`host-name`* option is specified. *`Host`* declarations match DHCP or BOOTP clients based on either the client's hardware address or the *`dhcp-client-identifier`* option that the client sends. BOOTP clients do not normally send a *`dhcp-client-identifier`* option. Therefore, you must use the hardware address for all clients that might send BOOTP protocol requests. The *`host`* declaration has three purposes: • to assign a static IP address to a client • to declare a client as "known" • to specify a scope in which statements can be executed for a specific client You can make the DHCP server treat some DHCP clients differently from others if *`host`* declarations exist for those clients. Any request coming from a client that matches a *`host`* declaration is considered to be from a "known" client. Requests that do not match any *`host`* declaration are considered to be from "unknown" clients. You can use this knowledge to control how addresses are allocated. It is possible to write more than one *`host`* declaration for a client. If you want to assign more than one static address to a given client, you can either specify more than one address in the *`fixed-address`* statement or you can write multiple *`host`* declarations.

Statement	Description
	Multiple `host` declarations are needed if the client has different requirements (scopes) on different subnets. For each IP address that requires a different scope, one `host` declaration should exist. A client can be in the scope of only one `host` declaration at a time. `Host` declarations with static address assignments are in scope for a client only if one of the address assignments is valid for the network segment to which the client is connected. If you want to boot a client using static addresses on some subnets, and using dynamically assigned addresses on other subnets, you need to write a `host` declaration with no `fixed-address` statement. There can be only one such `host` declaration per client. Its scope is used whenever that client receives a dynamically assigned address. `host name { [statements] [declarations] }`

if	The `if` statement conditionally executes statements based on the values the client sends or other information. See Section 9.12 for more information.
	
	`if boolean-expression { [statements] }`
	`[elsif boolean-expression { [statements] }]`
	`[else { [statements] } ]`

invalid-ddns-chars	This statement specifies how DHCP should handle invalid characters in the hostname for Dynamic DNS updates (DDNS).
	
	`fail` tells DHCP to display a message and not perform any DNS updates if there are any invalid characters in the hostname. This is the default.
	
	`invalid-ddns-chars fail;`
	
	`discard` tells DHCP to throw away the invalid characters in the hostname.
	
	`invalid-ddns-chars discard;`
	
	`replace` tells DHCP to replace the invalid characters with the specified character(s). If none are specified, the default replacement character is the hyphen ('-').
	
	`invalid-ddns-chars replace ["characters"];`

key	The `key` declaration specifies keys to use for Transaction Signatures (TSIG) to sign dynamic DNS updates. See Section 9.13.1 for more information.
	
	`key _ip-address_ {`
	` [ algorithm "algorithm-name"; ]`
	` key-id "key-name";`
	` secret "key";`
	`}`

key-id	Used only inside of `key` declarations, the `key-id` statement specifies the name of the key to use for *Transaction Signatures (TSIG)* on dynamic DNS updates. This key name must match the name that the DNS server is using (as specified in named.conf). `key-id "key-name";`
lease limit	This statement causes the DHCP server to limit the number of members of a class that can hold a lease at any one time. This limit applies to all addresses the DHCP server allocates in the class, not just addresses on a particular network segment.

Statement	Description
	• If a client is a member of more than one class with lease limits, the server assigns the client an address based on either class. • If a client is a member of one or more classes with limits and one or more classes without limits, the classes without limits are not considered. `lease limit limit;`
lease-scan-interval	This statement specifies how frequently to scan for expired leases. The default is 60 seconds. `lease-scan-interval seconds;`
match	*`data-expression`* is evaluated using the contents of a client's request. If it returns a value that matches a subclass of the class in which the *`match`* statement appears, the client is considered a member of both the subclass and the class. `match data-expression;`
match if	*`boolean-expression`* is evaluated when the server receives a packet from the client. If it is true, the client is considered a member of the class. The *`boolean-expression`* may depend on the contents of the packet the client sends. `match if boolean-expression;`
max-delayed-acks	Use the *`max-delayed-acks`* statement to specify the maximum number of DHCPACKs to batch up. The default is 8. To disable the delaying of DHCPACKs, specify a value of 1. To improve performance under very heavy loads, the DHCP server delays sending DHCPACK messages by up to 2 seconds. All DHCPACKs accumulated in that time are sent in a batch. `max-delayed-acks count;`
max-lease-time	Use the *`max-lease-time`* statementto assign the maximum amount of time (in seconds) to a lease. The only exception to this is Dynamic BOOTP lease lengths because they are not specified by the client and are not limited by this maximum. The infinite lease value is "infinite". The default is 86,400 seconds (24 hours). **Note** You should set the value of max-lease-time at least as large as default-lease-time. `max-lease-time time;`
min-lease-time	Use the *`min-lease-time`* statement to assign the minimum length in seconds to a lease. The infinite lease value is "infinite". By default, there is no minimum. *`min-lease-time`* should be less than or equal to *`default-lease-time`* and *`max-lease-time`*. `min-lease-time time;`
min-secs	Use the *`min-secs`* statement to assign the minimum amount of time (in seconds) it takes for the DHCP server to respond to a client's request for a new lease.

Statement	Description
	The number of seconds is based on what the client reports in the **secs** field of the requests it sends. The maximum value is 255 seconds. Usually, setting this to one second results in the DHCP server not responding to the client's first request, but always responding to the client's second request. You can use the *`min-secs`* statement to set up a secondary DHCP server to never offer an address to a client until the primary server has been given a chance to do so. If the primary server is down, the client binds to the secondary server; otherwise, clients should always bind to the primary. **Note** This does not permit a primary server and a secondary server to share a pool of dynamically-allocatable addresses. `min-secs seconds;`
next-server	Use the *`next-server`* statement to specify the host address of the server from where the client will load the initial boot file (specified in the *`filename`* statement). *`server-name`* should be a numeric IP address or a domain name. The DHCP server's IP address is used if no *`next-server`* parameter applies to a given client. `next-server name;`
one-lease-per-client	Use the *`one-lease-per-client`* statement to have the server free any other leases the client holds when the client sends a DCCPREQRUEST for a particular lease. This presumes the client has forgotten any lease not mentioned in the DHCPREQUEST. For example, the client has only a single network interface and it does not remember leases it is holding on networks to which it is not currently attached. Neither of these assumptions are guaranteed or provable, so use caution in the use of this statement. `one-lease-per-client flag;`
option	This statement specifies actual DHCP protocol options to send to the client. The option statement is described in Section 9.17.
option definition	This statement assigns a name and a type to an option code. See Section 9.17.3 for more information. `option name code code = definition;`
option space	This statement specifies a new option space. This declaration must precede all definitions for options in the space being specified. *`Space-name`* should be the name of the option space. Currently three option space names are predefined: • DHCP (default) • agent • server

Statement	Description
	If an option name is specified without an option space, it is assumed the name refers to an option in the *dhcp* option space. For example, the option names *dhcp.routers* and *routers* are equivalent. `option space space-name;`
ping	The DHCP server uses ping to check if a particular IP address is in use by sending a packet of information and waiting for a response. This statement turns ping on and off. The default is ON. ping flag;
ping-retries	This statement defines the number of times the DHCP server pings an IP address before it concludes that the address is not in use. The default is 1. `ping-retries count;`
ping-timeout	This statement defines the time (in seconds) that ping should wait for a response. The default is 1 second. `ping-timeout time;`
pool	This statement specifies an address pool from which IP addresses can be allocated. This pool can be customized to have its own permit list to control client access and its own scope to declare pool-specific parameters. You can put *pool* declarations within *subnet* declarations or within *shared-network* declarations. You can use the *range* declaration to specify the addresses in a particular pool. • For *subnet* declarations: specified addresses must be correct within the *pool* declaration within which it is made. • For *shared-network* declarations: specified addresses must be on *subnets* that were previously specified within the same *shared-network* declaration. `pool {[permit list][range declaration][statements]}`
range	For any subnet on which addresses are assigned dynamically, there must be at least one *range* declaration. The *range* declaration specifies that the server may allocate to DHCP clients every address, from *low-address* to *high-address*. You can specify a single IP address by omitting *high-address*. All IP addresses in the range should be on the same subnet. If the *range* declaration appears within a *subnet* declaration, all addresses should be on the declared subnet. If the *range* declaration appears within a *shared-network* declaration, all addresses should be on subnets already declared within the *shared-network* declaration. You may specify the dynamic-bootp flag if addresses in the specified range can be dynamically assigned to both BOOTP and DHCP clients. **Note** The dynamic-bootp flag was deprecated in version 3.0 of the DHCP server in favor of declaring the address within a pool and specifying in the permit list that dynamic allocation for BOOTP clients is permitted.

Statement	Description
	`range [dynamic-bootp] low-address [high-address];`

requested-options-only	Use the *`requested-options-only`* statement to send just the options requested by the client. To send a specific set of options, set *`requested-options-only`* to true and specify the *`dhcp-parameter-request-list`* option.
	
	The following sends only the *`subnet-mask`*, *`routers`*, and *`domain-name-servers`* options to the client (assuming they are defined in the configuration file):
	
	`host restricted {`
	`  hardware ethernet 01:02:03:04:05:06;`
	`  option dhcp-parameter-request-list 1, 3, 6;`
	`  requested-options-only true;`
	`}`
	
	We recommend you restrict the use of this feature to as few clients as possible.
	
	`requested-options-only flag;`

secret	Used only inside of *`key`* declarations, the *`secret`* statement specifies the actual secret key to use for*Transaction Signatures (TSIG)* on dynamic DNS updates. The format is base-64. The value must match the key used by the DNS server (as specified in named.conf). `secret "`*`key`*`";`
secure-ddns	Use the *`secure-ddns`* statement to cause the DHCP server to use *Transaction Signatures (TSIG)* to sign dynamic DNS updates. The default is to not use TSIG. `secure-ddns flag;`
server-identifier	The *`server-identifier`* statement is equivalent to the *`dhcp-server-identifier`* option. See the *`dhcp-server-identifier`* option for more information `server-identifier hostname;`
server-name	Use the *`server-name`* statement to inform the client of the server's name from which it is booting. *`name`* should be the name provided to the client. `server-name name;`
shared-network	Use this statement to inform the DHCP server that some IP subnets share the same physical network. Declare all subnets in the same shared network within a *`shared-network`* statement. *`Parameters`* specified in the shared-network statement will be used when booting clients on those subnets unless parameters provided at the subnet or host level override them. If more than one subnet in a shared network has addresses available for dynamic allocation, those addresses are collected into a common pool. There is no way to specify which subnet of a shared network a client should boot on. *`Name`* should be the name of the shared network. Make the name descriptive as it will be used when printing debugging messages. Give it a syntax of a valid domain name (although it will never be used as such), or any arbitrary name enclosed in quotation marks. `shared-network name {[statements] [declarations]}`

Statement	Description
site-option-space	Use the `site-option-space` statement to determine the option space from which site-local options are taken. Site-local options have codes ranging from 128 to 254. If no site-option-space is specified, site-specific options are taken from the default option space. `site-option-space option-space;`
spawn with	`data-expression` must evaluate to a non-null value for the server to look for a subclass of the class that matches the evaluation. • If such a subclass exists, the client is considered a member of both the subclass and the class. • If no such subclass exists, one is created and recorded in the lease database, and the client is considered a member of the new subclass as well as the class. `spawn with data-expression;`

subclass	This statement specifies a subclass of the class named by `class-name`. `Class-data` should be either
	
	• a text string enclosed in quotes, or
	
	• a list of bytes expressed in hexadecimal, separated by colons.
	
	Clients match subclasses after evaluating the `match` or `spawn with` statements in the `class` declaration for `class-name`. If the evaluation matches `class-data`, the client is a member of the subclass and the class.
	
	`subclass “class-name” class-data;`
	`subclass “class-name” class-data {`
	`  [statements]`
	`}`

subnet	This statement contains information specific to a subnet. The information communicates the following to DHCP: • Enough information for DHCP to determine if an IP address is on that subnet. • What the subnet-specific parameters are. • What addresses may be dynamically allocated to clients booting on that subnet. Use the `range` declaration to specify what addresses are available to be dynamically allocated to clients booting on the subnet. Two things are required to define a subnet: • The `subnet-number` • The `netmask` The `subnet-number` and the `netmask` entry is an IP address or domain name that resolves to the `subnet-number` or the `netmask` of the subnet being described. The `subnet-number` andthe `netmask` are enough to determine if any given IP address is on the specified subnet.

Statement	Description
	Note A `netmask` must be given with every `subnet` declaration. If there is any variance in subnet masks at a site, use a subnet-mask option statement in each `subnet` declaration to set the desired subnet mask. The subnet-mask option statement overrides the subnet mask declared in the subnet statement. `subnet subnet-number netmask netmask {[statements] [declarations]}`

use-host-decl-names	If the `use-host-decl-names` parameter is true, the name provided for each host declaration is given to the client as its hostname. The default is false. For example:
	
	`group {`
	` use-host-decl-names on;`
	` host joe {`
	` hardware ethernet 08:00:2b:4c:29:32;`
	` fixed-address joe.fugue.com;`
	` }`
	`}`
	`is equivalent to`
	`host joe {`
	` hardware ethernet 08:00:2b:4c:29:32;`
	` fixed-address joe.fugue.com;`
	` option host-name “joe”;`
	`}`
	
	An `option host-name` statement within a host declaration overrides the use of the name in the host declaration.
	
	`use-host-decl-names flag;`

use-lease-addr-for-default-route	If the `use-lease-addr-for-default-route` parameter is true in a given scope, the IP address of the lease being assigned is sent to the client instead of the value specified in the routers option (or sending no value at all). This causes some clients to ARP for all IP addresses, which can be helpful if your router is configured for proxy ARP. If `use-lease-addr-for-default-route` is enabled and an `option routers` statement are both in scope, `use-lease-addr-for-default-route` is preferred. `use-lease-addr-for-default-route flag;`
user-class	This statement has been deprecated in favor of the more powerful `class` statement. See Section 9.17.4.
vendor-class	This statement has been deprecated in favor of the more powerful `class` statement. See Section 9.17.4.
vendor-option-space	Use the `vendor-option-space` statement to instruct the server to construct a `vendor-encapsulated-options` option using all the defined options in the option space. If no `vendor-encapsulated-options` option is defined, the server sends this option to the client, if appropriate. `vendor-option-space option-space;`

9.16. Expressions

The DHCP server can evaluate expressions while executing statements. The DHCP server's expression evaluator returns the following types:

- A *boolean*, a true or false (on or off) value.
- An *integer*, a 32-bit quantity that may be treated as signed or unsigned, depending on the context.
- A *string of data*, a collection of zero or more bytes. Any byte value is valid in a data string — the DHCP server maintains a length rather than depending on a NUL termination.

Expression evaluation is performed when a request is received from a DHCP client. Values in the packet sent by the client can be extracted and used to determine what to send back to the client. If the expression refers to a field or option in the packet for which there is no value, the result is null. Null values are treated specially in expression evaluation. A Boolean expression that returns a null value is considered false. A data expression that returns a null value generally results in the statement using the value not having any effect.

The following is an example of using four types of expressions to produce the name of a PTR record for the IP address being assigned to a client:

```
concat (binary-to-ascii (10, 8, “.”, reverse (1, leased-address)),
    “.in-addr.arpa.”);
```

9.16.1. BOOLEAN EXPRESSIONS

The following table lists the boolean expressions supported by DHCP.

Expression	**Description**
boolean-expression-1 and boolean-expression-2	The *and* operator evaluates to true if both boolean expressions evaluate to true. The *and* operator evaluates to false if either boolean expression does not evaluate to true. If either of the boolean expressions is null, the result is null.
boolean-expression-1 or boolean-expression-2	The *or* operator evaluates to true if either of the boolean expressions evaluate to true. The *or* operator evaluates to false if both of the boolean expressions evaluate to false. If either of the boolean expressions is null, the result is null.
check “class-name”	The *check* operator evaluates to true if the packet being considered comes from a client in the specified class. *Class-name* must be a string that corresponds to the name of a defined class.
data-expression-1 = data-expression-2	The = operator compares the results of evaluating two data expressions, evaluating to true if they are the same; evaluating to false if they are not. If one of the expressions is null, the result is null.
exists option-name	The *exists* expression evaluates to true if the specified option exists in the incoming DHCP packet.

Expression	Description
known	The `known` expression evaluates to true if the client whose request is being processed is known; that is, if the client has a host declaration.
not boolean-expression	The `not` operator evaluates to true if the boolean expression evaluates to false. The `not` operator evaluates to false if the boolean expression evaluates to true. If the boolean expression evaluates to null, the result is null.

9.16.2. DATA EXPRESSIONS

The following table lists the expressions supported by DHCP that return a data string.

Expression	Description
binary-to-ascii (numeric-expr1, numeric-expr2, data-expr1, data-expr2)	*`numeric-expr1, numeric-expr2, data-expr1`*, and *`data-expr2`* are all evaluated as expressions and the results of those evaluations are used as follows. The *`binary-to-ascii`* operator converts the binary data in *`data-expr2`* into an ASCII string, using *`data-expr1`* as a separator. How the conversion is done is controlled by *`numeric-expr1`* and *`numeric-expr2`*. • *`numeric-expr1`* specifies the base to convert into. Any value 2 through 16 is supported. For example, a value of 10 would produce decimal numbers in the result. • *`numeric-expr2`* specifies the number of bits in data-expr2 to treat as a single unit. The value can be 8, 16, or 32. This example converts the binary value of an IP address into its dotted decimal equivalent: `binary-to-ascii(10, 8, ".", 168364039)` The result would be the string "10.9.8.7".
colon-separated hexadecimal list	A list of *`hexadecimal`* octet values, separated by colons, may be specified as a data expression. A single hexidecimal number, appearing in a context where a data string is expected, is interpreted as a data string containing a single byte.
concat (data-expr1, data-expr2)	*`data-expr1`* and *`data-expr2`* are evaluated and the concatenated result of these two evaluations is returned.

Expression	Description
	• If either subexpression evaluates to null, the result is the value of the expression that did **not** evaluate to null. • If both expressions evaluate to null, the result is null.
encode-int (numeric-expr, width)	`numeric-expr` is evaluated and encoded as a data string of the specified `width`, in network byte order (with the most significant byte first). If `numeric-expr` evaluates to null, the result is null.
hardware	The `hardware` operator returns a data string whose first element is the **htype** field of the packet being considered, and whose subsequent elements are the first **hlen** bytes of the **chaddr** field of the packet. • If there is no packet, or if the RFC 2131 **hlen** field is invalid, the result is null. Supported hardware types are: • ethernet (1) • token-ring (6) • fddi (8)
leased-address	In any context where the processing client request has been assigned an IP address, this data expression returns that IP address.
option option-name	The `option` operator returns the contents of the specified option in the incoming DHCP packet.
packet (offset, length)	The `packet` operator returns the specified portion of the packet being considered. The `packet` operator returns a value of null where no packet is being considered. `Offset` and `length` are applied to the contents of the packet as in the `substring` operator. The link-layer, IP, and UDP headers are not available.
reverse (numeric-expr1, data-expr2)	`numeric-expr1` and `data-expr2` are evaluated. The result of `data-expr2` is reversed in place, using chunks of the size specified in `numeric-expr1`. For example, if `numeric-expr1` evaluates to four and `data-expr2` evaluates to twelve bytes of data, the `reverse` expression evaluates to twelve bytes of data constructed in the following way:

Expression	Description
	• the last four bytes of the input data, • followed by the middle four bytes, • followed by the first four bytes.
substring (data-expr, offset, length)	The `substring` operator evaluates the data expression and returns the substring of the result of that evaluation that starts `offset` bytes from the beginning and continues for `length` bytes. `Offset` and `length` are numeric expressions. • If `data-expr`, `offset`, or `length` evaluate to null, the result is null. • If `offset` is greater than or equal to the length of the evaluated data, a zero-length data string is returned. • If `length` is greater than the remaining length of the evaluated data after `offset`, a data string containing all data from `offset` to the end of the evaluated data is returned.
suffix (data-expr, length)	The `suffix` operator evaluates `data-expr` and returns the last `length` bytes of that evaluation. `Length` is a numeric expression. • If `data-expr` or `length` evaluate to null, the result is null. • If `length` evaluates to a number greater than the length of the evaluated data, the evaluated data is returned.
“text”	A `text string`, enclosed in quotes, may be specified as a data expression. The `string` returns the text between the quotes, encoded in ASCII.

9.16.3. NUMERIC EXPRESSIONS

Numeric expressions evaluate to an integer. In general, the precision of numeric expressions is at least 32 bits. However, the precision of such integers may be more than 32 bits.

- extract-int (data-expr, width)

  The `extract-int` operator extracts an integer value in network byte order after evaluating `data-expr`. `Width` is the width in bits (either 8, 16, 32) of the integer to extract. If the evaluation of `data-expr` does not provide an integer of the specified size, a value of null is returned.

- `number`

  `Number` can be any numeric value between zero and the maximum representable size.

9.17. DHCP Options

The Dynamic Host Configuration protocol allows the client to receive options from the DHCP server describing the network configuration and various services that are available on the network. When configuring the DHCP server, options must often be declared. The syntax for declaring options, and the names and formats of the options in the default dhcp option space that can be declared, are in Table 9.4.

DHCP option statements always start with the keyword *`option`*, followed by an option name, followed by option data. Only options needed by clients must be specified.

An option name is an optional option space name followed by a period (".") followed by the option name. The default option space is *`dhcp`*. There are two other predefined option spaces: *`agent`* and *`server`*. You can also define option spaces of your own. See Section 9.17.2 and Section 9.17.3.

Option data comes in these formats:

- The **ip-address** data type can be entered either as an explicit IP address (e.g., 239.254.197.10) or as a domain name (e.g., haagen.isc.org). When entering a domain name, be sure that the domain name resolves to the single IP address.
- The **int32** and **uint32** data types specify signed and unsigned 32-bit integers.
- The **int16** and **uint16** data types specify signed and unsigned 16-bit integers.
- The **int8** and **uint8** data types specify signed and unsigned 8-bit integers. Unsigned 8-bit integers are also sometimes referred to as octets.
- The **string** data type specifies an NVT ASCII string. It must be enclosed in quotation marks. For example, *`option domain-name "isc.org";`*
- The **flag** data type specifies a boolean value. Booleans can be either true (ON) or false (OFF). You can use TRUE and FALSE, or ON and OFF.
- The **data-string** data type specifies either an NVT ASCII string enclosed in quotation marks, or a series of octets specified in hexadecimal, separated by colons. For example, *`option dhcp-client-identifier "CLIENT-FOO";`* or *`option dhcp-client-identifier 43:4c:49:54:2d:46:4f:4f;`*

Strings and data-strings when enclosed in quotation marks can contain normal C-type characters such as "\t" for a tab.

If the option value is a list (such as for the routes option), you must list them in the configuration file in the order you want the client to use the values. The DHCP server does not re-order them.

Also, option data may be specified using an expression that returns a data string (see Section 9.16). The syntax is

```
option option-name = data-expression;
```

9.17.1. Standard DHCP Options

Table 9.4 describes the standard DHCP options. Underlined items indicate user input items.

Note

All of these options could be specified with the dhcp option space listed explicitly. For example:

```
option dhcp.bootfile-name “bootfile.lis”;
```

Table 9.4. DHCP Option Space Options

Option	Description
option all-subnets-local flag;	Use this option to indicate whether or not to assume all subnets of the client’s IP network use the same MTU as the client’s subnet. ON means assume all subnets share the same MTU. OFF means assume some subnets have smaller MTUs.
option arp-cache-timeout uint32;	Use this option to identify the timeout (in seconds) for ARP cache entries.
option bootfile-name string;	Use this option to identify a bootstrap file. If this option is supported by the client, it should have the same effect as the `filename` declaration. BOOTP clients are unlikely to support this option. Some DHCP clients support it; others require it.
option boot-size uint16;	Use this option to specify the length in 512-octet blocks of the client’s default boot image.
option broadcast-address ip-address;	Use this option to identify the broadcast address in use on the client’s subnet. See STD 3 (RFC 1122) for legal values for broadcast addresses.
option cookie-servers ip-address [, ip-address ...];	Use this option to list RFC 865 cookie servers in order of preference.
option default-ip-ttl uint8;	Use this option to identify the default time-to-live the client should use on outgoing datagrams.
option default-tcp-ttl uint8;	Use this option to identify the default TTL to use when sending TCP segments. The minimum value is 1.
option dhcp-client-identifier data-string;	Use this option to specify a DHCP client identifier only in a *`host`* declaration. The DHCP server uses it to locate the *`host`* record by matching against the client identifier.
option dhcp-max-message-size uint16;	Use this option to specify the maximum length DHCP message that the client is able to accept. Use this option in the DHCP configuration file to supply a value when the client does not. **Note** Use this option with caution. Make sure that the client can accept a message of the specified size.

Option	Description
option dhcp-parameter-request-list uint8[,uint8...];	Use this option to request that the server return certain options. Use this option in the DHCP configuration file to override the client's list, or to supply a list when the client does not. The value is a list of valid DHCP option codes as listed in RFC 2132.
option dhcp-server-identifier ip-address;	Use this option to identify the value sent in the DHCP Server Identifier option. The value must be an IP address for the DHCP server, and must be reachable by all clients it is sent to. It is recommended to NOT use the dhcp-server-identifier option. The only reason to use it is to force a value other than the default value to be sent on occasions where the default value would be incorrect. The default value is the first IP address associated with the physical network interface on which the request arrived. The usual case where the dhcp-server-identifier option needs to be sent is when a physical interface has more than one IP address, and the one being sent by default is not appropriate for some or all clients served by that interface. Another case is when an alias is defined for the purpose of having a consistent IP address for the DHCP server, and it is desired that the clients use this IP address when contacting the server.
option domain-name-servers ip-address [, ip-address ...];	Use this option to list Domain Name System (STD 12, RFC 1035) name servers in order of preference.
option domain-name string;	Use this option to identify the domain name the client should use when resolving hostnames via the Domain Name System.
option extensions-path string;	Use this option to indicate the path-name of a file the client should load containing more options.
option finger-server ip-address [, ip-address ...];	Use this option to list the Finger servers in order of preference.
option font-servers ip-address [, ip-address ...];	Use this option to list X Window System Font servers in order of preference.
option host-name string;	Use this option to name the client. The name may or may not be qualified with the local domain name. It is preferable to use the domain-name option to specify the domain name. See RFC 1035 for character set restrictions.

Option	**Description**
	The host-name option is also used to specify a template for hostname generation. See Section 9.14.
option ieee802-3-encapsulation flag;	If the interface is an Ethernet, use this option to indicate whether the client uses Ethernet Version 2 (RFC 894) or IEEE 802.3 (RFC 1042) encapsulation. OFF means use RFC 894 encapsulation. ON means use RFC 1042 encapsulation.
option ien116-name-servers ip-address [, ip-address ...];	Use this option to list IEN 116 name servers in order of preference.
option impress-servers ip-address [, ip-address ...];	Use this option to list Imagen Impress servers in order of preference.
option interface-mtu uint16;	Use this option to identify what MTU value to use on this interface. The minimum legal value is 68.
option ip-forwarding flag;	Use this option to indicate if the client should configure its IP layer for packet forwarding. ON means disable forwarding. OFF means enable forwarding.
option irc-server ip-address [, ip-address ...];	Use this option to list the IRC servers in order of preference.
option log-servers ip-address [, ip-address ...];	Use this option to list MIT-LCS UDP log servers in order of preference.
option lpr-servers ip-address [, ip-address ...];	Use this option to list RFC 1179 line printer servers in order of preference.
option mask-supplier flag;	Use this option to indicate whether the client should respond to subnet mask requests using ICMP. ON means do not respond to subnet mask requests. OFF means respond to subnet mask requests.
option max-dgram-reassembly uint16;	Use this option to indicate the maximum size datagram the client should be prepared to reassemble. The minimum legal value is 576.
option merit-dump string;	Use this option to indicate the path-name of a file to which the client's core image should be dumped in the event of a client crash. The path

Option	Description
	is formatted as a character string using the NVT ASCII character set.
option mobile-ip-home-agent ip-address [, ip-address ...];	Use this option to list mobile IP home agents in order of preference. Usually there will be only one agent.
option nds-context data-string;	Use this option to identify the initial NDS context the client should use.
option nds-servers ip-address [, ip-address...];	Use this option to list Novell Directory Services servers in order of preference.
option nds-tree-name data-string;	Use this option to name the NDS tree the client will be contacting.
option netbios-dd-server ip-address [, ip-address ...];	Use this option to list RFC 1001/1002 NetBIOS Datagram Distribution servers in order of preference.
option netbios-name-servers ip-address [, ip-address ...];	Use this option to list RFC 1001/1002 NetBIOS Name Server name servers in order of preference. **Note** NetBIOS is the same as WINS.
option netbios-node-type uint8;	Use this option to configure configurable NetBIOS over TCP/IP clients as described in RFC 1001/1002. The value is a single octet identifying the client type. 1 B-node: Broadcast — No WINS 2 P-node: Peer — WINS only 4 M-node: Mixed — Broadcast, then WINS 8 H-node: Hybrid — WINS, then Broadcast
option netbios-scope string;	Use this option to specify the NetBIOS over TCP/IP scope parameter for the client as specified in RFC 1001/1002. See RFC1001, RFC1002, and RFC1035 for character-set restrictions.
option nis-domain string;	Use this option to specify the client’s NIS (Sun Network Information Services) domain. Use the NVT ASCII character set to define the domain character string.
option nis-servers ip-address [, ip-address ...];	Use this option to list NIS servers in order of preference.
option nisplus-domain string;	Use this option to specify the client's NIS+ domain. Use the NVT ASCII character set to define the domain character string.

Option	**Description**
option nisplus-servers ip-address [, ip-address ...];	Use this option to list NIS+ servers in order of preference.
option non-local-source-routing flag;	Use this option to indicate if the client should configure its IP layer to allow forwarding of datagrams with non-local source routes. ON means disable forwarding. OFF means enable forwarding.
option nntp-server ip-address [, ip-address ...];	Use this option to list NNTP servers in order of preference.
option ntp-servers ip-address [, ip-address ...];	Use this option to list NTP (RFC 1305) servers in order of preference.
option option-nnn data-string;	Use this option to identify any DHCP option not listed here. nnn is the number of the option.
option path-mtu-aging-timeout uint32;	Use this option to specify the timeout to use (in seconds) when aging Path MTU values that were discovered by the mechanism defined in RFC 1191.
option path-mtu-plateau-table uint16 [, uint16 ...];	Use this option to specify a table of MTU sizes to use when performing Path MTU Discovery as defined in RFC 1191. The table is a list of 16-bit unsigned integers. You must list them in order from smallest to largest. The minimum MTU value cannot be smaller than 68.
option perform-mask-discovery flag;	Use this option to indicate whether or not the client should perform subnet mask discovery using ICMP. ON means do not perform mask discovery. OFF means perform mask discovery.
option policy-filter ip-address ip-address [, ip-address ip-address ...];	Use this option to indicate the policy filters for non-local source routing. The filters consist of IP addresses and masks that indicate which destination/mask pairs to use when filtering incoming source routes. The client should discard any source routed datagram whose next-hop address does not match one of the filters. See STD 3 (RFC 1122) for more information.
option pop-server ip-address [, ip-address ...];	Use this option to list POP3 servers in order of preference.

Option	**Description**
option resource-location-servers ip-address [, ip-address ...];	Use this option to list RFC 887 Resource Location servers in order of preference.
option root-path string;	Use this option to specify the path-name that contains the client's root disk. The path is formatted as a character string using the NVT ASCII character set.
option router-discovery flag;	Use this option to indicate whether or not the client should solicit routers using the Router Discovery mechanism defined in RFC 1256. ON means do not perform router discovery. OFF means perform router discovery.
option routers ip-address [, ip-address ...];	Use this option to list IP addresses for routers on the client's subnet, listing the routers in order of preference.
option router-solicitation-address ip-address;	Use this option to identify the address where the client transmits router solicitation requests.
option smtp-server ip-address [, ip-address ...];	Use this option to list SMTP servers in order of preference.
option static-routes ip-address ip-address [, ip-address ip-address ...];	Use this option to specify a list of static routes that the client should install in its routing cache. If there are multiple routes to the same destination, you should list them in descending order of priority. The routes are made up of IP address pairs. The first address is the destination address; the second address is the router for the destination. The default route (0.0.0.0) is an illegal destination for a static route. Use the `routers` option to specify the default route.
option streettalk-directory-assistance-server ip-address [, ip-address ...];	Use this option to list the StreetTalk Directory Assistance (STDA) servers in order of preference.
option streettalk-server ip-address [, ip-address ...];	Use this option to list the StreetTalk servers in order of preference.
option subnet-mask ip-address;	Use this option indicate the client's subnet mask as per RFC 950. If no subnet mask option is in scope, the DHCP server uses the subnet mask from the subnet declaration on which the address is being assigned. If a subnet mask option is in scope for the address being assigned, it overrides the subnet mask specified in the subnet declaration.

Option	**Description**
option swap-server ip-address;	Use this option to identify the IP address of the client’s swap server.
option tcp-keepalive-garbage flag;	Use this option to indicate whether the client sends TCP keepalive messages with an octet of garbage for compatibility with older implementations. ON means do not send a garbage octet. OFF means send a garbage octet.
option tcp-keepalive-interval uint32;	Use this option to indicate the interval (in seconds) the client TCP waits before sending a keepalive message on a TCP connection. The time is specified as a 32-bit unsigned integer. 0 (zero) means do not generate keepalive messages unless requested by an application.
option tftp-server-name string;	Use this option to identify a TFTP server. If this option is supported by the client, it should have the same effect as the `server-name` declaration. BOOTP clients are unlikely to support this option. Some DHCP clients support it; others require it.
option time-offset int32;	Use this option to specify the offset of the client’s subnet (in seconds) from Coordinated Universal Time (UTC). Use negative numbers for West of UTC and positive number for East of UTC.
option time-servers ip-address [, ip-address ...];	Use this option to list RFC 868 time servers in order of preference.
option trailer-encapsulation flag;	Use this option to indicate if the client negotiates the use of trailers (RFC 893) when using the ARP protocol. ON means do not use trailers. OFF means use trailers.
option vendor-encapsulated-options data-string;	Use this option to specify vendor specific information. See Section 9.17.4.
option www-server ip-address [, ip-address ...];	Use this option to list WWW servers in order of preference.
option x-display-manager ip-address [, ip-address ...];	Use this option to list the systems running X Window System Display Manager in order of preference.

9.17.2. Relay Agent Information Option

A relay agent can add a series of encapsulated options to a DHCP packet when relaying that packet to the DHCP server. The server can make address allocation decisions (or whatever decisions it wants) based on these options. The server returns these options in any replies it sends through the relay agent. The relay agent can use the information in these options for delivery or accounting purposes.

The relay agent option has two suboptions. To reference these options in the DHCP server, specify the option space name "agent", followed by a period, followed by the option name.

Note

It is not useful to specify these options to be sent.

Table 9.5. Agent Option Space Options

`option agent.circuit-id string;`	The *`circuit-id`* suboption encodes an *`agent-local`* identifier of the circuit from which a DHCP *`client-to-server`* packet was received. It is intended for agents who will use it in relaying DHCP responses back to the proper circuit. The format of this option is defined to be vendor-dependent.
`option agent.remote-id string;`	The *`remote-id`* suboption encodes information about the remote host end of a circuit. Examples include the following: caller ID information, username information, remote ATM address, and cable modem ID. This is an opaque object that is administratively guaranteed to be unique to a particular remote end of a circuit.

9.17.3. Defining New Options

You can define new options with the DHCP server. Each DHCP option has the following:

- A **name**, used by you to refer to the option.
- A **code**, a number used by the DHCP server to refer to the option.
- A **structure**, describing what the contents of the option look like.

To define a new option, choose a **name** that is not in use for any other option. For example, you can not use "host-name" because the DHCP protocol already defines a host-name option. You should refer to the options listed in this chapter as these are all the DHCP options in use by VSI TCP/IP.

After choosing a name, choose a **code**. For site-local options, all codes between 128 and 254 are reserved for DHCP options, so you can use any one of these.

The **structure** of an option is the format in which the option data appears. The DHCP server supports a few simple types: for example, integers, booleans, strings, and IP addresses. The server also supports the ability to define arrays of single types or arrays of fixed sequences of types. The syntax for declaring new options is:

```
option new-name code new-code = definition ;
```

The values of new-name and new-code are the name and the code you have chosen for the new option. The `definition` is one of the following simple option type definitions.

Definition	**Description**

`BOOLEAN`	`option new-name code new-code = boolean ;`
	
	An option of type *`boolean`* is a flag with a value of either ON (true) or OFF (false). For example:
	
	`option use-zephyr code 180 = boolean;`
	`option use-zephyr on;`

`INTEGER`	`option new-name code new-code = sign integer width ;`
	
	The sign token should either be blank, unsigned, or signed. The width can be either 8, 16 or 32, referring to the number of bits in the integer. For example, a definition of the *`sql-connection-max`* option and its use:
	
	`option sql-connection-max code 192 = unsigned integer 16;`
	`option sql-connection-max 1536;`

`IP-ADDRESS`	`option new-name code new-code = ip-address ;`
	
	An option of type *`IP address`* can be expressed either as a domain name or as an explicit IP address. For example:
	
	`option sql-server-address code 193 = ip-address;`
	`option sql-server-address sql.example.com;`

`TEXT`	`option new-name code new-code = text ;`
	
	An option of type *`text`* encodes an ASCII text string. For example:
	
	`option sql-default-connection-name code 194 = text;`
	`option sql-default-connection-name "`*`PRODZA`*`";`

`DATA STRING`	`option new-name code new-code = string ;`
	
	An option of type *`data string`* is a collection of bytes. It can be specified either as quoted text, like the text type, or as a list of hexadecimal octets separated by colons whose values must be between 0 and FF. For example:
	
	`option sql-identification-token code 195 = string;`
	`option sql-identification-token 17:23:19:a6:42:ea:99:7c:22;`

`ARRAYS`	Options can contain arrays of any of the above types except for the *`text`* and the *`data string`* types. For example:
	
	`option kerberos-servers code 200 = array of ip-address;`
	`option kerberos-servers 10.20.10.1, 10.20.11.1;`

`RECORDS`	Options can contain data structures consisting of a sequence of data types, sometimes called a record type. For example:
	
	`option contrived-001 code 201 = { boolean, integer 32, text };`
	`option contrived-001 on 1772 "`*`contrivance`*`";`
	
	It is also possible to have options that are arrays of records. For example:
	
	`option new-static-routes code 201 = array of {`
	` ip-address, ip-address, ip-address, integer 8 };`

Definition	Description

	`option static-routes`
	` 10.0.0.0 255.255.255.0 net-0-rtr.example.com 1,`
	` 10.0.1.0 255.255.255.0 net-1-rtr.example.com 1,`
	` 10.2.0.0 255.255.224.0 net-2-0-rtr.example.com 3;`

9.17.4. Vendor Encapsulated Options

The DHCP protocol defines the *`vendor-encapsulated-options`* option. This allows vendors to define their own options that will be sent encapsulated in a standard DHCP option. The format of the *`vendor-encapsulated-options`* option is either a chunk of opaque data, or an actual option buffer just like a standard DHCP option buffer.

You can send this option to clients in one of two ways:

- define the data directly, using a text string or a colon-separated list of hexadecimal values
- define an option space, define some options in that option space, provide values for them, and specify that this option space should be used to generate the vendor-encapsulated-options option

To send a simple chunk of data, provide a value for the option in the right scope. For example:

```
option vendor-encapsulated-options
 2:4:AC:11:41:1:
 3:12:73:75:6e:64:68:63:70:2d:73:65:72:76:65:72:31:37:2d:31:
 4:12:2f:65:78:70:6f:72:74:2f:72:6f:6f:74:2f:69:38:36:70:63;
```

To define a new option space to store vendor options, use the *`option space`* statement. The name can then be used in option definitions. For example:

```
option space SUNW;
option SUNW.server-address code 2 = ip-address;
option SUNW.server-name code 3 = text;
option SUNW.root-path code 4 = text;
```

Once you have defined an option space and some options, you can set up scopes that define values for those options and when to use them. For example, suppose you want to handle two different classes of clients. Using the option space definition, the previous

```
option vendor-encapsulated-options example can be implemented using classes
 as follows:
class "vendor-classes" {
 match option vendor-class-identifier;
}
option SUNW.server-address 172.17.65.1;
option SUNW.server-name "sundhcp-server17-1";
subclass "vendor-classes" "SUNW.Ultra-5_10" {
 vendor-option-space SUNW;
 option SUNW.root-path "/export/root/sparc";
}
subclass "vendor-classes" "SUNW.i86pc" {
 vendor-option-space SUNW;
 option SUNW.root-path "/export/root/i86pc";
}
```

Regular scoping rules apply. This lets you define values that are global in the global scope, and define values that are specific to a particular class in the local scope.

The `vendor-option-space` declaration indicates that in that scope the `vendor-encapsulated-options` option should be constructed using the values of all the options in the SUNW option space.

9.18. DHCP Lease Format

The DHCP server keeps a persistent database of leases it has assigned. This database is a free-form ASCII file containing a series of lease declarations. Every time a lease is acquired, renewed, or released, its new value is recorded at the end of the lease file. So, if more than one declaration appears for a given lease, the last one in the file is the current one.

Currently, the only declaration that is used in the dhcpd.leases file is the `lease` declaration.

```
lease ip-address {statements...}
```

Each lease declaration includes the client's leased IP address. The statements within the braces define, for example, the duration of the lease and to whom it is assigned.

Table 9.6 describes the statements the DHCP server puts into a lease file. For a list of DHCP Safe-failover related lease file statements see Table 9.11.

Table 9.6. DHCP Lease File Statements

Lease Statement	Description
abandoned;	Records that the DHCP server saw the IP address in use on the network when it was thought to be free. The DHCP server detects active addresses with ping tests or "DHCP decline" messages from DHCP clients.
billing class “class-name”;	If this lease is a member of a class with “lease limit” set, this records that class.
billing subclass “class-name” subclass-data;	If this lease is a member of a subclass with “lease limit” set, this records the class and subclass.
client-hostname “hostname”;	Records the hostname if the client sends a hostname using the hostname option.
domain-name “domain-name”;	Specifies the DNS domain name sent to the client (if any).
dynamic-bootp;	Indicates the address was leased to a BOOTP client.
ends date;	Records the end time of a lease. Lease dates are specified by the DHCP server as follows: **W YYYY/MM/DD HH:MM:SS** where: **W** is the day of the week, from zero (Sunday) to six (Saturday). **YYYY** is the year, including the century.

Lease Statement	**Description**
	MM is the number of the month, from 01 to 12. **DD** is the day of the month, counting from 01. **HH** is the hour, from 00 to 23. **MM** is the minute, from 00 to 59. **SS** is the second, from 00 to 59. The time is always in Greenwich Mean Time, not local time.
FQDN “fully-qualified-domain-name”;	Specifies the fully qualified domain name used by the DHCP server to perform the dynamic DNS update for the lease (if any).
hardware hardware-type mac-address;	Specifies the hardware type and the MAC address as a series of hexadecimal octets, separated by colons.
hostname “hostname”;	Records the hostname if the DHCP server looks up the hostname in DNS. This happens only if the parameter *`get-lease-hostnames`* was set.
starts date;	Records the start time of a lease.
uid client-identifier;	Records the client identifier as a series of hexadecimal octets, regardless of whether the client specifies an ASCII string or uses the hardware type /MAC address format. If the client used a client identifier to acquire its address, the client identifier is recorded using the uid statement.

9.18.1. Working with DHCP Leases

The DHCP server requires that a lease database be present before it will start. The VSI TCP/IP installation supplies an empty `IP$:DHCPD.LEASES` file.

In order to prevent the lease database from growing without bound, the file is rewritten from time to time. First, a temporary lease database is created and all known leases are dumped to it. Then, the old lease database is renamed `IP$:DHCPD.LEASES_OLD`. Finally, the newly written lease database is moved into place.

Be aware of the following situation: if the DHCP server process is killed or the system crashes after the old lease database has been renamed but before the new lease database has been moved into place, the `IP$:DHCPD.LEASES` file disappears. The DHCP server will refuse to start. Do not create a new lease file when this happens. If you do, you will lose all your old bindings. Instead, rename `IP$:DHCPD.LEASES_OLD` to `IP$:DHCPD.LEASES`, restoring the old, valid lease file, and then start the DHCP server. This guarantees that a valid lease file will be restored.

9.18.2. Abandoned Leases

Abandoned leases are reclaimed automatically. When a client asks for a new address, and the server finds that there are no addresses available, it checks to see if there are any abandoned leases. The server allocates the oldest abandoned lease. The standard procedures for checking for lease address conflicts are still followed, so if the abandoned lease's IP address is still in use, it is reabandoned.

If a client requests an abandoned address, the server assumes that the address was abandoned because the lease file was corrupted, and that the client is the machine that responded when the lease was pinged, causing it to be abandoned. In that case, the address is immediately assigned to the requesting client.

9.18.3. Static Leases

Leases that are given to clients for statically assigned IP addresses are treated differently than those for dynamically assigned IP addresses. An address is statically assigned by using a *`host`* declaration with a *`fixed-address`* statement.

Static lease information is not saved by the DHCP server. This means that they are not recorded in the lease file (`DHCPD.LEASES`). If your configuration uses only statically assigned IP addresses, you will not see any entries in the lease file.

This also means that the NETCONTROL **SHOW** commands do not have any lease information to display for static assignments.

- For **SHOW CLIENT**, statically assigned IP addresses are not supported.
- For **SHOW SUBNET** and **SHOW LEASES**, statically assigned IP addresses are not shown.
- For **SHOW ALL**, **SHOW HADDR**, **SHOW CID**, and in the dump file produced by the **DUMP** command, statically assigned IP addresses are identified as "Static Assignment" and no lease information is shown.
- For **STATISTICS** and **SHOW POOLS**, statically assigned IP addresses are not included in the pool or subnet statistics.

DNS dynamic updates are supported only partially for static assignments. When the lease is granted, the appropriate A and PTR resource records are added automatically. However, since the lease information is not saved, the DHCP server does not delete the DNS entries when the lease expires or is released.

9.19. Registering Clients While the DHCP Server is Running

The DHCP server can register and unregister clients without having to restart the server. *`host`* declarations and *`subclass`* declarations can be added or removed from the running server using *`add`* and *`remove`* commands in an update file, by default `IP$:DHCPD.UPDATES`.

The commands that can be placed into the update file are described in Section 9.19.1.

You would use *`host`* declarations if you are controlling access to IP addresses via *`allow/deny unknown-clients`* statements in your `DHCPD.CONF` configuration file. You would use *`subclass`* statements if you are controlling access to IP address pools using classes configured with the *`match`* statement and using pools with *`allow/deny members of _class_`* statements.

Note

The registration or unregistration of a client via the update file only affects the running server. The host and subclass declarations **must** also be added to the `DHCPD.CONF` configuration file.

You tell the DHCP server to execute the commands in the update file using the **NETCONTROL DHCP UPDATE** command:

```
$ IP NETCONTROL DHCP UPDATE
```

A different file name can optionally be specified:

```
$ IP NETCONTROL DHCP UPDATE mydir:dhcpd.updates
```

You can verify the syntax of the update file before sending it to the DHCP server by using the -u flag on the dhcpd command line:

```
$ DHCPD :== $IP$:dhcpd4.exe
$ DHCPD -T -U filename
```

The update file name is not optional. Note that the configuration file is read in before the update file. A different configuration file can be specified using the -cf flag.

You can check the DHCP server and see if a given host or subclass is known, for example to see if you need to add it, using the following netcontrol commands:

```
$ IP NETCONTROL SHOW ISKNOWN HOST hw-addr-or-client-id
$ IP NETCONTROL SHOW ISKNOWN SUBCLASS class-name subclass-data
```

9.19.1. Update File Statements

Table 9.7 describes the commands you can use in an update file.

Table 9.7. DHCP Update File Commands

Statement	**Description**
add host	Registers a client by adding the specified host declaration. The host declaration is in the same format as in the configuration file. This makes the specified hardware address and/or client identifier "known". `add host name { [statements] }` Note that static IP address assignments can be added by specifying the *`fixed-address`* statement in the host declaration.

add subclass	Registers a client by adding the specified subclass to the specified class. The class must be declared in the `DHCPD.CONF` configuration file. The subclass declaration is the same format as in the configuration file. This adds the specified subclass value as a member of the specified class.
	
	`add subclass "class-name" subclass-data;`
	`add subclass "class-name" subclass-data {`
	`[statements]`
	`}`

delete host	Un-registers a client by removing the specified host declaration. The host is specified by hardware address or client identifier. This makes the

Statement	Description
	specified host "unknown". Note that all host declarations that match the hardware address or client identifier are deleted. `delete host hw-addr-or-client-id;`
delete subclass	Un-registers a client by removing the specified subclass from the specified class. This makes the specified subclass no longer a member of the class. `delete subclass "class-name" subclass-data;`

9.19.1.1. Examples:

```
add host fred {
 hardware ethernet 01:02:03:04:05:06;
 fixed-address 10.9.8.7;
 option routers 10.9.8.1;
}
add host wilma {
 option dhcp-client-identifier "wilma-cid";
}
delete host 01:02:03:04:05:06;
delete host "wilma-cid";
add subclass "gold" 01:01:02:03:04:05:06 {
 option host-name "fred";
}
add subclass "silver" "wilma-cid";
delete subclass "gold" 01:01:02:03:04:05:06;
delete subclass "silver" "wilma-cid";
```

9.20. DHCP Safe-failover Introduction

Since a DHCP server is responsible for the network's IP management, it can also be a potential point of network failure if it becomes unavailable. Using multiple servers with non-overlapping IP address pools is one way to provide limited fault-tolerance. For example: imagine a network with two DHCP servers. Server A has an address range of 100 IP addresses. Server B has a range of 50 different addresses. Both servers have a non-overlapping range of addresses. When a node broadcasts for an address, both servers respond, each offering an address from its own distinct range. Upon receiving both offers, the node chooses one. Typically, the response that reaches the node first is selected. In this case, Server A's. When Server B determines its offer is not needed, it returns the offered address to its own range, allowing it to be offered again.

If one of the servers is down, the other server continues to service the nodes. Now, instead of having two offers, each new node has only one offer, from the remaining server. Nodes that received their lease from the unavailable server attempt to reconnect with it. If the unavailable server does not respond in time, the nodes then attempt to get a new address from a new server. The other server can then offer an address from its own range. So, even though one server is down, the DHCP clients continue to function with the other server.

Note

The two DHCP servers operate without any communications or data sharing between them. Each server works as a standalone server.

Since most nodes select the first offer received, having two servers could result in partial use of both IP address pools. Sometimes it is preferable to have a primary DHCP server with the bulk of the IP addresses while the secondary server has a smaller range of IP addresses.

Note

One way to accomplish the above mentioned configuration is to put the secondary server behind a router on a different subnet, while the primary server stays on the same segment as the nodes. This allows the primary server to respond more quickly than the secondary server.

VSI takes the use of multiple servers to another level by offering DHCP Safe-failover. DHCP Safe-failover allows a secondary DHCP server to back up the primary DHCP server with the addition of taking into account network failure. This strategy insures that clients can reliably log into their corporate network and know they will be able to connect to corporate resources.

In safe failover mode both the primary and the backup DHCP servers share a common IP address lease pool. In the event the primary DHCP server fails the backup DHCP server automatically senses the primary server is not operating and automatically assumes complete DHCP operation. When the primary DHCP server becomes operational, it synchronizes with the backup DHCP server and resumes all the responsibilities of the primary DHCP server. All assignments performed by the backup DHCP server while acting as the primary server are transferred to the primary DHCP upon resumption of primary server responsibilities.

Safe-failover adds support for network failure, as well as server failure. In the event the network fails, the secondary server believes the primary server is out of service and begins operation. The secondary server serves leases from a reserved pool shared by the Safe-failover partner servers. This reserve pool can be adjusted by the MIS system administrator.

9.21. Configuring DHCP Safe-failover

To configure your DHCP servers to use Safe-failover, perform the following steps:

1. Choose one system to be the Primary and a second system to be the Secondary.
2. Determine the IP addresses of the Primary and Secondary systems. If a system has more than one IP address, choose one to use for DHCP Safe-failover messages.
3. On the Primary system, create the boot file at `IP$:DHCPD.BOOT` with the keyword "primary", the primary and secondary IP addresses, and configuration ID.

   On the primary system, the configuration ID would normally be "DHCPD". See Section 9.22 for more information about boot files.

   Primary system boot file syntax:

   ```
   primary primary-ip secondary-ip "config-id";
   ```

4. On the Secondary system, create the boot file at `IP$:DHCPD.BOOT` with the keyword "secondary", the secondary and the primary IP addresses, and configuration ID. On the secondary system, the configuration ID may be "dhcpd" or may be a name that refers to the primary. Either way, the names of the configuration, lease, and state files must match this name.

   Secondary system boot file syntax:

   ```
   secondary secondary-ip primary-ip "config-id";
   ```

5. If you do not already have a configuration file, write a configuration file containing the subnets, shared networks, IP address ranges, hosts, etc, that reflect your network and the hosts you want the DHCP server to service. Include any DHCP Safe-failover parameters as needed (see Section 9.24).

6. Copy the configuration file to the IP directory on both the Primary and the Secondary systems.

Note

Make sure the name of the configuration file matches the config-id parameter in the boot file for that system.

Preferably, the configuration files on the Primary and the Secondary server systems should be the same. To help ensure that the configuration file is valid for both systems, make sure it contains a subnet statement for every subnet that either the Primary or the Secondary system has a network interface on.

7. Make sure that both the Primary and the Secondary systems have lease files in the `IP$:` directory with the name that matches config-id. If the lease file does not exist, create an empty one.

8. Run the DHCP server on both the Primary and the Secondary systems. The two servers will establish communications with each other. Now the process is completed successfully.

9.22. Boot File for DHCP Safe-failover

To use Safe-failover, create a boot file at `IP$:DHCPD.BOOT`. The boot file is used to specify the following for Safe-failover operation:

- Server's mode of operation
- Server's own IP address
- Partner server's IP address
- Configuration ID

The format of the boot file is:

```
mode [server-IP partner-IP] "config-id";
```

If the boot file is not present, the server operates with DHCP Safe-failover disabled. It uses `IP$:DHCPD.CONF` and `IP$:DHCPD.LEASES` for its default configuration and lease files. In this state, the service parameters CONFIGFILE and LEASEFILE may be used to rename or move these files. The server does not use a state file to keep track and remember its state transitions when running in standalone mode (that is, with DHCP Safe-failover disabled). See Section 9.23 for a description of state files. The following server modes are possible:

Table 9.8. DHCP Safe-failover Server Modes

Mode	In this mode the server runs...
Primary	As the Primary server, with DHCP Safe-failover functionality enabled. In this mode, the server tries to "shadow" each of its lease transactions to its configured secondary server.
Secondary	As the Secondary or Backup server, with DHCP Safe-failover functionality enabled. The server receives lease transaction updates from

Mode	In this mode the server runs...
	its configured Primary server, and maintains an up-to-the-minute hot backup of the lease database. If the primary server crashes, or is shut down, the Secondary server takes over.
Standalone	With DHCP Safe-failover disabled.

The IP address following the server mode is the server's own address, the next IP address is the partner server's IP address.

The configuration ID is the file name (without the file type) of the configuration, lease, and state files. For example, if the configuration id is ALPHA, the DHCP server will look for the configuration file named `ALPHA.CONF`, a lease file named `ALPHA.LEASES`, and a state file named `ALPHA.STATE`.

Example 9.2. Boot File

```
primary 199.23.24.11 199.23.24.25 "ALPHA";
```

The example boot file designates the current server as the primary with its own IP address 199.23.24.11 and the partner server's IP address 199.23.24.25. The partner server is a Secondary server. This follows from the current server being a Primary server.

If the mode of operation is "standalone", the server's IP address and partner server's IP address are not needed. The format is as follows:

```
standalone "config-id";
```

9.23. State File for DHCP Safe-failover

The state file is written by the DHCP server when it is running with DHCP Safe-failover enabled. The purpose of the state file is to save server state changes so that a server can "remember" its last state if it crashes or is re-started. Alternately, the state file can be used by the operator to force the DHCP server to start up in a desired mode (operator override). This feature allows the operator to switch the server into partner-down mode without a lengthy time-out period, or to start up in recover mode (that is, to force the DHCP server to download the entire lease database from the partner).

The server appends a line to the state file every time its DHCP Safe-failover state changes. The last line in the file is the most current DHCP Safe-failover state.

The state file consists of one or more lines of the following format:

```
server_state transaction_count; [date-time;]
```

server_state is one of the following:

Table 9.9. DHCP Safe-failover Server States

failover-disabled	primary-comint	primary-conflict
startup	backup-comint	backup-conflict
primary-normal	primary-partnerdown	primary-recover
backup-normal	backup-partnerdown	backup-recover

Server-to-server messages are each assigned a monotonously increasing transaction number, which is recorded in the transaction_count field. This is an unsigned 32 bit number.

If the date-time stamp is present, the DHCP server assumes that the state was recorded by the server itself. In this case, the server, upon starting up, calculates the safest state based on the recorded state and the time elapsed. If the date-time stamp is not present, the DHCP server treats the entry as an operator-written override command and starts up in the specified state.

9.24. DHCP Safe-failover Configuration File Statements

The statements shown in the Table 9.10 have been added to the DHCP configuration file for DHCP Safe-failover. These are in addition to the DHCP configuration file statements listed in the Table 9.4. All of the parameters in Table 9.10 must be placed in the configuration file's global scope. With the exception of the backup-pool-size parameter. It may also be specified within a shared-network or subnet declaration.

Table 9.10. DHCP Safe-failover Configuration File Statements

Statement	Description
backup-ack-interval	The number of seconds used as the basis of the DHCP server's logarithmic back-off algorithm used for resending ACK messages to the secondary server. The default is 1 second. `backup-ack-interval seconds;`
backup-pool-size	This is the percentage of the IP address pool set aside for the Secondary server’s emergency use. The DHCP server will reserve no more than 50% of the available addresses. The default is 10%. `backup-pool-size percent;`
com-int-timeout	The number of seconds the server should wait for an expected response from its partner before switching to communication interrupted mode. The default is 600 seconds (10 minutes). `com-int-timeout seconds;`
failover-port	The UDP port the Primary and Secondary servers should use for DHCP Safe-failover messages. The default is 647. `failover-port port;`
mclt	Maximum Client Lead Time: This is the length of lease in seconds to give out on a first time basis, or the client lead time for renewals. See the DHCP failover internet draft for a more detailed explanation. The default is 3600 seconds (1 hour). `mclt seconds;`
safe-period-timeout	The number of seconds spent in communication interrupted state before automatic switch over to partner-down state. A value of 0 means no automatic switch over. The default is 0 seconds. `safe-period-timeout seconds;`
startup-delay	The number of seconds to wait during startup before the server moves from STARTUP state to the state specified in the state file. The default is 5 seconds. `startup-delay seconds;`

9.25. DHCP Safe-failover Lease File Statements

The statements shown in the Table 9.11 have been added to the DHCP lease file for DHCP Safe-failover.

Table 9.11. DHCP Safe-failover Lease File Statements

Statement	Description
acked-sec-interval seconds;	Acknowledged secondary lease interval. For information, see the DHCP failover internet draft.
acked-sec-interval-start date;	The time when the partner was notified of the lease.
active;	This IP address has a current lease.
backup;	This IP address is reserved for use by the secondary (backup) server.
desired-interval seconds;	The length of the desired lease.
expired;	The lease for this IP address has expired.
free;	This IP address is available to be assigned.
last-partner-transaction date;	The time when the partner last updated the lease.
released;	The lease for this IP address has been released by the client or by the operator.
reset;	The DHCP server had marked this IP address as abandoned. The operator changed its status to available.
revoked;	The lease for this IP address has been revoked by the operator.
safe-lease;	This is used in the Partner Down state to indicate that the IP address belongs to this server.
transaction-id number;	This is the transaction number that was assigned to this lease when it was queued or sent as an update to the partner server.
update-count n;	For each lease, the server which issues the lease sends an update to its partner server. This statement records the state of that update. 0 — means no update is required 1 — means that no update has been sent 2+ — means 1 or more updates have been sent

9.26. Transitioning to DHCP Safe-failover Partner Down State

There are three ways that the DHCP server can transition to Partner Down state, which indicates that its DHCP Safe-failover partner is down.

1. If the parameter safe-period-timeout is specified in the configuration file, the DHCP server transitions to Partner Down state automatically after it has been in Communication Interrupt state for the specified time.

2. The operator can put the DHCP server into Partner Down state by executing the following netcontrol command:

   ```
   $ IP NETCONTROL DHCP PARTNERDOWN
   ```

3. The operator can edit the DHCP server's state file and add a line to the end containing the Partner Down state and transaction number desired:

   ```
   primary-partnerdown transaction-number;
   ```

   or

4. `backup-partnerdown transaction-number;`

   The next time the DHCP server is restarted, it starts up in Partner Down state. The operator can restart the DHCP server by executing the following netcontrol command:

   ```
   $ IP NETCONTROL DHCP RESTART
   ```

9.27. Setting DHCP Parameters

The DHCP service uses the parameters listed in the Table 9.12.

Table 9.12. DHCP Server Parameters

Parameter	Description
`ACCOUNTING`	This parameter determines if the DHCP server process performs OpenVMS accounting. The default is 0 (OFF).
`CONFIGFILE`	The name of the configuration file. The default is `IP$:DHCPD.CONF`. Not used if DHCP Safe-failover is being used.
`DEBUG`	A decimal integer that is a bitmask of debugging levels used to select messages to pass to OPCOM and the debug log file specified in the DEBUG-FILE parameter. The debugging levels are (in decimal): 1 Fatal Errors 3 Errors and Warnings 7 Informationals 15 Debug Messages 31 Dump Packets (Formatted) 63 Dump Packets (Hex) By default, Fatal Errors, Errors, and Warnings are logged.
`DEBUG-FILE`	The name of the debug log file. Use this parameter to capture debug logging in a file. The default is that debug logging is not written to any file if LOG-TO-OPCOM is 1. If LOG-TO-OPCOM is 0, the default file is `IP$:DHCPDEBUG.LOG`.
`DUMPFILE`	The name of the "dump" file. This is the output of the NETCONTROL **DUMP** command. The default is `IP$:DHCPD.DUMP`.

Parameter	Description
`IMAGE-NAME`	The name of the image to run in the DHCP server process. The default is `IP$:DHCPD.EXE`.
`LEASEFILE`	The name of the file DHCP uses to save client lease information to survive across reboots. To completely clear the lease information, delete the file specified and create an empty version. The default is `IP$:DHCPD.LEASES`. Not used if DHCP Safe-failover is being used.
`LOG-DATE`	This parameter determines whether the date is included in each entry in the debug log file. The default is 0; the date is not included.
`LOG-TO-OPCOM`	This parameter determines whether debug logging messages are written to OPCOM. Debug messages are also written to DEBUG-FILE parameter value if DEBUG-FILE is specified or this parameter is 0. Severe errors and warnings are always logged to OPCOM. The default is 1, everything is logged to OPCOM.
`PROCESS-NAME`	The name of the DHCP server process. The default is `DHCP_SERVER.`
`SYS-ERROR`	The name of a file that will contain anything written to `SYS$ERROR` by the DHCP server. This information could be helpful in diagnosing problems. The default is `_NL`:. This means `SYS$ERROR` is not directed to any file.
`SYS-OUTPUT`	The name of a file that will contain anything written to `SYS$OUTPUT` by the DHCP server. This information could be helpful in diagnosing problems. The default is `_NL`: meaning `SYS$OUTPUT` not directed to any file.
`SWAP`	This parameter determines whether process swapping is inhibited for the DHCP server process. The default is 1, swapping is enabled.
`UPDATEFILE`	The name of the update file. This file contains update commands that the DHCP server executes upon a NETCONTROL **UPDATE** command. The default is `IP$:DHCPD.UPDATES`.

You may set any of the parameters listed in the Table 9.6, as shown in the following example:

```
$ IP NETCONTROL DHCP SHUTDOWN (if DHCP is running)
$ IP CONFIGURE /SERVER

VSI TCP/IP for OpenVMS Server Configuration Utility 10.5(nnn)
[Reading in configuration from IP$:SERVICES.MASTER_SERVER]
SERVER-CONFIG>SELECT DHCP
[The Selected SERVER entry is now DHCP]
SERVER-CONFIG>SET PARAMETERS
Delete parameter "configfile IP$:DHCPD.CONF" ? [NO]
You can now add new parameters for DHCP. An empty line terminates.
Add Parameter: debug 3
Add Parameter:
[Service specific parameters for DHCP changed]
SERVER-CONFIG>RESTART
Configuration modified, do you want to save it first ? [YES]
[Writing configuration to IP$COMMON_ROOT:[IP]
SERVICES.MASTER_SERVER]
%RUN-S-PROC_ID, identification of created process is 20600046
SERVER-CONFIG>EXIT
[Configuration not modified, so no update needed]
```

```
$
```

9.28. Viewing DHCP Information

VSI TCP/IP provides two NETCONTROL commands for displaying information about the DHCP server:

- NETCONTROL **SHOW** (see Section 9.28.1)
- NETCONTROL **STATISTICS** (see Section 9.28.2)

9.28.1. NETCONTROL SHOW Command

The DHCP NETCONTROL **SHOW** command has seven subcommands: **SHOW CLIENT**, **SHOW LEASES**, **SHOW SUBNET**, **SHOW ALL**, **SHOW HADDR**, **SHOW CID**, and **SHOW POOLS**.

Viewing Lease Information for Specific IP Addresses

The DHCP NETCONTROL **SHOW CLIENT** command displays the current lease binding details on a particular IP address. It must be an IP address in the dynamic pool. Statically-bound IP addresses are not supported. The syntax for **SHOW CLIENT** is:

```
$ IP NETCONTROL DHCP SHOW CLIENT dotted-decimal-ip-address
```

`dotted-decimal-ip-address` is the IP address of a client.

For example:

```
$ IP NETCONTROL DHCP SHOW CLIENT 10.5.64.1
Connected to NETCONTROL server on "LOCALHOST"
< x.process.com Network Control V10.5(10) at Fri 3-Mar-2017 3:23PM-EDT
< DHCP Client: 10.5.64.1
<  IP Address=10.5.64.1
<  State=Bound (expired)
<  Subnet Mask=255.255.255.0
<  Default Gateway=10.5.64.100
<  Hardware Address=00004400AABB
<  Client ID=74657374 (“test”)
<  Lease=300 secs, Obtained 06-Mar-2017 22:21:22 GMT Expires 16-Mar-2017
 22:26:22 GMT (-75426 secs)
< End of Show DHCP Client
```

Viewing Lease Information for all Leased IP Addresses

The DHCP NETCONTROL **SHOW LEASES** command takes no arguments. It displays for all subnets the IP addresses that have leases (pending, active, or expired). Lease information for statically-assigned IP addresses is not available. For example:

```
$ IP NETCONTROL DHCP SHOW LEASES
Connected to NETCONTROL server on "LOCALHOST"
< x.process.com Network Control V10.5(10) at Fri 3-Mar-2017 3:23PM-EDT
< DHCP Client: 10.5.64.1
<  IP Address=10.5.64.1
<  State=Bound (expired)
<  Subnet Mask=255.255.255.0
<  Default Gateway=10.5.64.100
<  Hardware Address=00004400AABB
<  Client ID=74657374 (“test”)
<  Lease=300 secs, Obtained 06-Mar-2017 22:21:22 GMT Expires 16-Mar-2017
 22:26:22 GMT (-75426 secs)
< End of Show DHCP Client
```

Viewing Address Pools for Specific Subnets

The DHCP NETCONTROL **SHOW SUBNET** command displays all of the DHCP address pools for the shared network that *`ip-address`* is in. It lists each subnet that is on the shared network and each IP address in each pool. Statically-bound IP addresses are not shown. The syntax for **SHOW SUBNET** is:

```
$ IP NETCONTROL DHCP SHOW SUBNET dotted-decimal-ip-address
```

`dotted-decimal-ip-address` is any IP address in that subnet.

You would typically use the subnet value or an IP address from a pool for the subnet. For example:

```
$ IP NETCONTROL DHCP SHOW LEASES
Connected to NETCONTROL server on "LOCALHOST"
< x.process.com Network Control V10.5(10) at Fri 3-Mar-2017 2:32PM-EST
< List all leases
< Shared Network local
< Subnet 10.5.64.0 netmask 255.255.255.0
< Subnet 10.5.165.0 netmask 255.255.255.0
< Pool 1
<  IP Addr=10.5.64.1, State=Bound (expired), Lease Expires
03-Mar-2017 19:32:43 GMT (-4 secs)
<  IP Addr=10.5.64.13, State=Offered, Lease Expires 06-Mar-2017 19:34:07
 GMT (80 secs)
<  IP Addr=10.5.64.2, State=Bound, Lease Expires 06-Mar-2017 19:36:52 GMT
 (245 secs)
< End of lease list
```

Viewing Address Pools for All Subnets

The DHCP NETCONTROL **SHOW ALL** command takes no arguments. It shows the **SHOW SUBNET** output for all subnets in the DHCP server configuration. Then it prints information about all static assignments.

For static assignments, lease information is not available. For example:

```
$ IP NETCONTROL DHCP SHOW ALL
Connected to NETCONTROL server on "LOCALHOST"
< x.process.com Network Control V10.5(10) at Mon 12-Jun-2017 11:24AM-EDT
< List all Subnet pools
< Shared Network local
< Subnet 10.5.64.0 netmask 255.255.255.0
< Subnet 10.5.165.0 netmask 255.255.255.0
< Pool 1
<  IP Addr=10.5.64.3, State=Free, No Lease
<  IP Addr=10.5.64.1, State=Bound, Lease Expires 12-Jun-2017 22:21:14 GMT
 (2867 secs)
< DHCP Static Assignments
<  IP Addr=10.5.64.15, State=(Static Assignment)
<  IP Addr=10.5.165.17, State=(Static Assignment)
<  IP Addr=10.5.165.200, State=(Static Assignment)
< End of Subnet pool lists
```

Viewing Matched Leases for Hardware Addresses

The DHCP NETCONTROL **SHOW HADDR** command shows all client entries that match a given hardware address. The clients can have leases on multiple subnets simultaneously.

Note

For hardware addresses that have static assignments, lease information is not available.

The syntax for the NETCONTROL **SHOW HADDR** command is

```
$ IP NETCONTROL DHCP SHOW HADDR MAC_address
```

For example:

```
$ IP NETCONTROL DHCP SHOW HADDR 00004400AABB
Connected to NETCONTROL server on "LOCALHOST"
< x.process.com Network Control V10.5(10) at Fri 07-Apr-2017 4:41PM-EDT
< DHCP Hardware Address: 00004400AABB
<  IP Address=10.5.64.1
<  State=Bound (expired)
<  Subnet Mask=255.255.255.0
<  Default Gateway=10.5.64.100
<  Hardware Address=00004400AABB
<  Client ID=74657374 (“test”)
<  Lease=300 secs, Obtained 07-Apr-2017 22:21:22 GMT Expires 07-Apr-2017
 22:26:22 GMT (-80091 secs)
< End of Show DHCP HAddr
```

Viewing Matched Leases for Client ID

The DHCP NETCONTROL **SHOW CID** command shows all client entries that match a given Client ID. The clients can have leases on multiple subnets simultaneously.

Note

For client IDs that have static assignments, lease information is not available, as shown in this example. The syntax for the NETCONTROL **SHOW CID** command is

```
$ IP NETCONTROL DHCP SHOW CID Client_ID_in_hex
```

For example:

```
$ IP NETCONTROL DHCP SHOW CID 7465737431
Connected to NETCONTROL server on "LOCALHOST"
< x.process.com Network Control V10.5(10) at Mon 12-Jun-2017 6:36PM-EDT
< DHCP Client ID: 7465737431
<  IP Address=10.24.25.1
<  State=(Static Assignment)
<  Subnet Mask=255.255.255.0
<  Default Gateway=<none>
<  Hardware Address=<none>
<  Client ID=7465737431 (“test1”)
< End of Show DHCP Client ID
```

Viewing IP Address Pool Availability

The DHCP NETCONTROL **SHOW POOLS** command takes no arguments. It displays a table showing for each IP address pool the number of IP addresses that are available. An IP address pool corresponds to a *`shared-network`* statement, a *`subnet`* statement, or a *`pool`* statement in the DHCP configuration file. For each pool the following information is displayed:

Shared Network	The name from the *`shared-network`* statement or the subnet number from the *`subnet`* statement.
Pool	“Total” for the complete information for the shared network, otherwise a number identifying the pool. You can see which IP addresses are in which pools using the **SHOW ALL** or **SHOW SUBNET** command.
Total	The total number of IP addresses in the pool.
Abandoned	The number of IP addresses in the pool which were found in use on the network when they were thought to be free.
Reserved	If DHCP Safe-failover is in use, the number of IP addresses in the pool reserved for the secondary DHCP server. These addresses are unassigned but reserved for the secondary.
Available	The number of IP addresses in the pool available to be leased.

For example:

```
Connected to NETCONTROL server on "LOCALHOST"
<   x.process.com Network Control V10.5(10) at Sat 11-Feb-2017 5:25PM-EST
<   List Pool Availability
<   Shared Network   Pool   Total   Abandoned   Reserved   Available
<   --------------   -----  -----   ---------   --------   ---------
<   local            total  44      5           0          15
<                    1      44      5           0          15
<   10.12.1.0        total  128     2           0          57
<                    1      111     0           0          54
<                    2      11      2           0          0
<                    3      6       0           0          3
<   End of pool availability list
```

9.28.2. NETCONTROL STATISTICS Command

The DHCP NETCONTROL **STATISTICS** command displays the number of clients in each shared network. This is supplied for backward compatibility only. Use **SHOW POOLS** instead.

9.29. Address Lease States in DHCP Dump Files

After obtaining a DHCP dump using **IP NETCONTROL DHCP DUMP**, you will see fields in the dump preceded by a pound sign (#). Those fields are values created while the server is running. These fields are provided to help you troubleshoot problems. The lease states (denoted by *`# State`*= in the dump) are described in the Table 9.13.

Table 9.13. DHCP Address Lease States

State	Description
Abandoned	The address was seen in use on the network when it was thought to be free. The DHCP server detects active addresses with ping tests or "DHCP decline" messages from DHCP clients.
Bound	The address was assigned in response to a DHCP Request message. If the address lease expires, it remains in "bound" state to help the client regain the same IP address it previously used. The address is actually free. The "(expired)" modifier on the state value indicates this state.
Free	The address has never been bound, and is available in the pool.
Offered	The address has been offered to a client in response to the client's DHCP Discover message, but the client has not asked for the address via a DHCP Request message.
Pinging	The address is in the middle of a ping test.
Reserved for Secondary	Used for DHCP Safe-failover: The address is set aside for the secondary server’s emergency use.
Static Assignment	The client identifier or hardware address is statically assigned; the binding does not expire.

9.29.1. Sample DHCPD.CONF File

Example 9.3 shows a sample `DHCPD.CONF` file.

Example 9.3. Sample DHCPD.CONF File

```
#
# IP$:DHCPD.CONF -- sample DHCP configuration file
#
# option definitions common to all supported networks...
option domain-name “fugue.com”;
option domain-name-servers toccato.fugue.com;
default-lease-time 43200;
option subnet-mask 255.255.255.0;
option time-offset 18000;
use-host-decl-names on;
# Shared network declaration is used to group subnets which share the
 same physical network together. The name is specified so that the shared
 network can be referred to in log messages - it serves no other function.
#
# Note: You must have a subnet declaration for the subnet that the DHCP
 server system is on even if you do not want any address pool for the same
 subnet (or multiple subnets if the system is multi-homed)shared-network
 FUGUE {
# option definitions common to this shared network.
 option subnet-mask 255.255.255.224;
 default-lease-time 600;
 max-lease-time 7200;
# One of the two IP subnets that share this physical network
#
# Address ranges can be specified for each subnet attached to a shared
 network. Since these subnets share the same physical network, addresses
 are pooled together, and assignments are made without regard to the actual
 subnet. If the optional dynamic-bootp keyword is given in the address
 range declaration, then addresses in that range can be assigned either
 with the DHCP protocol or the BOOTP protocol; otherwise, only DHCP clients
 will have addresses allocated from the address range.
#
# Note that each IP subnet can have its own options specific to that
 subnet. Options that are not specified in the subnet are taken from the
 shared network (if any) and then from the global option list.
 subnet 204.254.239.0 netmask 255.255.255.224 {
  range 204.254.239.10 204.254.239.20;
  option broadcast-address 204.254.239.20;
  option routers prelude.fugue.com;
}
# The other subnet that shares this physical network
 subnet 204.254.239.32 netmask 255.255.255.224 {
  range dynamic-bootp 204.254.239.42 204.254.239.52;
  option broadcast-address 204.254.239.31;
  option routers snarg.fugue.com;
}
# Subnets can have no pooled ip addresses.
 subnet 10.10.10.0 netmask 255.255.255.0 {
 }
}
# IP subnets that are alone on their physical wire should be declared by
 themselves. The DHCP server still refers to them as shared networks in log
 messages, but this is simply an artifact of the underlying data structure.
#
# Note that options can be specified in the subnet declaration that
 supercede the global options specified earlier.
subnet 192.5.5.0 netmask 255.255.255.224 {
 range 192.5.5.26 192.5.5.40;
 option domain-name-servers bb.home.vix.com, gw.home.vix.com;
 option domain-name “vix.com”;
 option routers 192.5.5.1;
 option subnet-mask 255.255.255.224;
 option broadcast-address 192.5.5.41;
 default-lease-time 600;
 max-lease-time 7200;
}
# Hosts that require special configuration options can be listed in host
 statements. If no address is specified, the address will be allocated
 dynamically (if possible), but the host-specific information will still
 come from the host declaration.
host passacaglia {
 hardware ethernet 0:0:c0:5d:bd:95;
 filename “vmunix.passacaglia”;
 server-name “toccato.fugue.com”;
}
# Fixed IP addresses can also be specified for hosts. These addresses
 should not also be listed as being available for dynamic assignment. Hosts
 for which fixed IP addresses have been specified can boot using BOOTP or
 DHCP. Hosts for which no fixed address is specified can only be booted
 with DHCP, unless there is an address range on the subnet to which a BOOTP
 client is connected which has the dynamic-bootp flag set.
host fantasia {
 hardware ethernet 08:00:07:26:c0:a5;
 fixed-address fantasia.fugue.com;
}
```

```
# If a DHCP or BOOTP client is mobile and might be connected to a variety
 of networks, more than one fixed address for that host can be specified.
 Hosts can have fixed addresses on some networks, but receive dynamically
 allocated address on other subnets; in order to support this, a host
 declaration for that client must be given which does not have a fixed
 address. If a client should get different parameters depending on what
 subnet it boots on, host declarations for each such network should be
 given. Finally, if a domain name is given for a host’s fixed address
 and that domain name evaluates to more than one address, the address
 corresponding to the network to which the client is attached, if any, will
 be assigned.
host confusia {
 hardware ethernet 02:03:04:05:06:07;
 fixed-address confusia-1.fugue.com, confusia-2.fugue.com;
 filename “vmunix.confusia”;
 server-name “toccato.fugue.com”;
}
host confusia {
 hardware ethernet 02:03:04:05:06:07;
 fixed-address confusia-3.fugue.com;
 filename “vmunix.confusia”;
 server-name “snarg.fugue.com”;
}
host confusia {
 hardware ethernet 02:03:04:05:06:07;
 filename “vmunix.confusia”;
 server-name “bb.home.vix.com”;
}
# Some other examples
host host1 {
 option dhcp-client-identifier “host1”;
 fixed-address 10.10.11.101, 10.11.22.101;
}
# Do not allow this one to boot
host host2
 hardware ethernet aa:cc:04:00:33:11;
 deny booting;
}
```

9.30. DHCP Client

This section describes the Dynamic Host Configuration Protocol (DHCP) client.

9.30.1. General Description

The DHCP client resides on the client host and dynamically sets the network configuration. The VSI TCP/IP DHCP client communicates with a DHCP server to get an IPv4 address and other configuration information. It uses this information to configure the network parameters of the host and to start up the network.

When the network starts on the host, the DHCP client communicates dynamically and automatically with the DHCP server in case reconfiguration is needed. The configuration information the client uses is defined by the policy stored in the DHCP server.

For more general DHCP information, see RFC2132 and RFC2131.

VSI TCP/IP 10.5 provides support for a DHCP client. Because it supports a single network interface on the host, you can only use the DHCP client to configure a single network line in VSI TCP/IP.

VSI does not recommend you to use the DHCP client with other VSI TCP/IP components that usually need a static IPv4 address on the same host, such as DHCP server, authoritative DNS server, and GateD.

9.30.2. Setting DHCP Client Parameters

The DHCP Client service uses the parameters listed in the following table.

Table 9.14. DHCP Client Parameters

Parameters	Description
`CONFIGFILE`	The name of the configuration file. The default is `IP $CONFIG:DHCLIENT.CONF`.
`DEBUG`	A decimal integer that is a bitmask of debugging levels used to select messages to pass to OPCOM and the debug log file specified in the DEBUG-FILE parameter. The debugging levels are (in decimal): 1 Fatal Errors 3 Errors and Warnings 7 Informationals 15 Debug Messages 31 Dump Packets (formatted) 63 Dump Packets (hex) By default, Fatal Errors, Errors, and Warnings are logged.
`DEBUG-FILE`	The name of the debug log file. Use this parameter to capture debug logging in a file. The default is that debug logging is not written to any file if LOG-TO-OPCOM is 1. If LOG-TO-OPCOM is 0 (zero), the default file is `IP $CONFIG:DHCLIENT.LOG`.
`INTERFACE-ROUTE`	A decimal integer 1 (one). Use this parameter to add a static IP route to the VSI TCP/IP Kernel routing table. The optional INTERFACE keyword forces the routing to be for a locally connected interface. If the parameter is not set, the default is to create a gateway route.
`LEASEFILE`	The name of the file DHCP uses to save client lease information to survive across reboots. To completely clear the lease information, delete the file specified and create an empty version. The default is `IP $CONFIG:DHCLIENT.DB`. This is not used if DHCP Safe-failover is used.
`LOG-DATE`	This parameter determines whether the date is included in each entry in the debug log file. The default is 0 (zero); the date is not included.
`LOG-TO-OPCOM`	This parameter determines whether debug logging messages are written to OPCOM. Debug messages are also written to DEBUG-FILE parameter value if DEBUG-FILE is specified or this parameter is 0 (zero). Severe errors and warnings are always logged to OPCOM. The default is 1, everything is logged to OPCOM.

9.30.3. Setting Up the DHCP Client

Before setting up a DHCP client, you should talk to your network administrator. The administrator may want to assign a host name to your DHCP client

If this is your first time using the DHCP client on the host, you need to do the following:

```
$ COPY IP$CONFIG:DHCLIENT_CONF.DEFAULT IP$CONFIG:DHCLIENT.CONF
```

Then, edit the file “`IP$CONFIG:DHCLIENT.CONF`” to replace this line:

```
#Send host-name “testing”;
```

with this line:

```
Send host-name “any hostname you want”;
```

You can configure your local host to use the DHCP client when you run the VSI TCP/IP configuration utilities SERVER-CONFIG.

To enable the DHCP client service and to set a parameter, use:

```
$ IP CONFIGURE /SERVER
```

To set up the ethernet interface and enable it for DHCP client, refer to the following example:

```
$ IP CONFIGURE /NETWORK
VSI TCP/IP Network Configuration Utility
[No checking is done against the MAXIMUM configuration]
[Reading in configuration from IP$CONFIG:NETWORK_DEVICES.CONFIGURATION]
NET-CONFIG>MODIFY SE0
[Modifying configuration entry for device "se0"]
VMS Device [EWA0]:
Link Level Encapsulation Mode [ETHERNET]:
BSD Trailer Encapsulation: [DISABLED]
IP Address [0.0.0.0]: 10.10.2.6
IP SubNet Mask [NONE]: 255.255.255.0
Non-Standard IP Broadcast Address [10.10.2.255]:
DHCP CLIENT [ENABLED]: ENABLED
Jumbo Frames [DISABLED]:
IPv6 on this interface [DISABLED]:
[se0 (Shared VMS Ethernet/FDDI): Csr=NONE, Flags=%X0]
NET-CONFIG>EXIT
[Writing configuration to IP$CONFIG:NETWORK_DEVICES.CONFIGURATION]
[Writing Startup file SYS$STARTUP:IP$SYSTARTUP.COM]
[Changes take effect after the next VSI TCP/IP reload]
```

After configuring the local host to use the DHCP client, you can run **IP$SYSTARTUP.COM** to start VSI TCP/IP as in the following example:

Example 9.4. Using IP CONFIGURE /SERVER

```
$ IP CONFIGURE /SERVER
VSI TCP/IP for OpenVMS Server Configuration Utility V10.5(nn)
[Reading in configuration from IP$:SERVICES.MASTER_SERVER]
SERVER-CONFIG>ENABLE DHCLIENT4
SERVER-CONFIG>EXIT
[Writing configuration to IP$COMMON_ROOT:[IP]SERVICES.MASTER_SERVER]
[Configuration not modified, so no update needed]
```

```
$
```

This procedure creates the Network configuration data file, `IP$:NETWORK_DEVICES.CONFIGURATION`, as well as creating the Network Startup file, `IP$SYSTARTUP.COM` to reflect your system's configuration.

Follow these steps to activate your DHCP Client interface on a running system:

1. Enter the following command :

   ```
   $ DEFINE/SYSTEM/EXECUTIVE IP$DHCP_CLIENT "1"
   ```

2. After enabling the DHCP Client, start the DHCP Client service by restarting the VSI TCP/IP Master Server. If the DHCP Client service is already running, shut it down first by issuing this command:

   ```
   $ IP NETCONTROL DHCLIENT4 SHUTDOWN
   ```

3. Start the VSI TCP/IP Master Server with this command:

   ```
   $ @IP$STARTUP:START_SERVER
   ```

9.30.4. Disabling the DHCP Client

To disable the DHCP Client, do the following:

1. Run $ **IP CONFIGURE/SERVER**

2. Issue the commands:

   ```
   SERVER-CONFIG>DISABLE DHCLIENT4
   SERVER-CONFIG>EXIT
   ```

3. Assign an IP address to your system and disable DHCP as shown in the follwing example:

   ```
   $ IP CONFIGURE /NETWORK
   VSI TCP/IP Network Configuration Utility
   [No checking is done against the MAXIMUM configuration]
   [Reading in configuration from IP$CONFIG:NETWORK_DEVICES.CONFIGURATION]
   NET-CONFIG>MODIFY SE0
   [Modifying configuration entry for device "se0"]
   VMS Device [EWA0]:
   Link Level Encapsulation Mode [ETHERNET]:
   BSD Trailer Encapsulation: [DISABLED]
   IP Address [0.0.0.0]: 10.10.2.6
   IP SubNet Mask [NONE]: 255.255.255.0
   Non-Standard IP Broadcast Address [10.10.2.255]:
   DHCP CLIENT [ENABLED]: DISABLED
   Jumbo Frames [DISABLED]:
   IPv6 on this interface [DISABLED]:
   [se0 (Shared VMS Ethernet/FDDI): Csr=NONE, Flags=%X0]
   NET-CONFIG>EXIT
   [Writing configuration to IP$CONFIG:NETWORK_DEVICES.CONFIGURATION]
   [Writing Startup file SYS$STARTUP:IP$SYSTARTUP.COM]
   [Changes take effect after the next VSI TCP/IP reload]
   ```

To avoid rebooting the system, deassign the logical `IP$DHCP_CLIENT` after disabling the DHCP client and restart the VSI TCP/IP server with the `@IP$STARTUP:START_SERVER` command.

9.30.5. DHCP Client Functions and Logicals

The DHCP Client is started as a OpenVMS detached process (`IP$DHCP_CLIENT`) when VSI TCP/IP starts.

When the client starts, it configures the network interface (the line) with an IP address of "0.0.0.0", and then sends a DHCP discover packet to contact any DHCP server on the net. After getting an IP address and other net configuration information back from a DHCP server, it restarts the network interface with the IP address and configures VSI TCP/IP on the host with the information it received. That information may include the default gateway, DNS domain name, host name, DNS servers' IP addresses, and other things. After the network interface is configured and started, the DHCP client goes to sleep and waits for specified events (lease expired, renewal time reached) to wake it up again for possible re-configuration.

If the DHCP client cannot get the information it needs from the DHCP server, it may re-try until it succeeds. The re-try frequency can be controlled by the configuration file.

The DHCP client process sets the following items only when configuring the network interface, if it received the appropriate information from the DHCP server:

- IP address of the network interface
- Host name of the network interface
- Domain Name
- DNS client (Resolver)
- Routes/Gateway

It may change or set `IP$NAMESERVERS` VSI TCP/IP logical.

It may change the following related OpenVMS logicals:

`UCX$BIND_DOMAIN`	`UCX$BIND_SERVER000`
`UCX$BIND_SERVER00x`	`UCX$INET_DOMAIN`
`UCX$INET_HOST`	`UCX$INET_HOSTADDR`
`TCPIP$BIND_DOMAIN`	`TCPIP$BIND_SERVER000`
`TCPIP$BIND_SERVER00x`	`TCPIP_DOMAINNAME`
`TCPIP_NAMESERVERS`	

9.30.6. DHCP Client Configuration

The VSI TCP/IP DHCP client uses the configuration file `IP$:DHCLIENT.CONF` to control the behavior of the client. Use the template file `IP$:DHCLIENT_CONF.DEFAULT` to start with.

The `IP$CONFIG:DHCLIENT4_CONF.DEFAULT` file is a free-form ASCII text file. The file may contain extra tabs and new lines for formatting purposes. Keywords in the file are case-insensitive. Comments begin with the # character and end at the end of the line and may be placed anywhere within the file (except within quotation marks). You can use the `DHCLIENT.CONF` file to configure the behavior of the client in the following ways:

- Protocol timing
- Information requested from the server
- Information required of the server
- Defaults to use if the server does not provide certain information
- Values with which to override information provided by the server
- Values to prepend or append to information provided by the server

The configuration file can also be preloaded with addresses to use on networks that do not have DHCP servers.

9.30.7. Protocol Timing

The timing behavior of the client need not be configured by the user. If no timing configuration is provided by the user, a reasonable timing behavior will be used by default – one which results in timely updates without placing an inordinate load on the server. The following statements can be used to adjust the timing behavior of the DHCP client if required.

Table 9.15. DHCP Client Protocol Timing Statements

Statement	**Description**
backoff-cutoff time	The client uses an exponential backoff algorithm with some randomness, so that if many clients try to configure themselves at the same time, they will not make their requests in lockstep. The *`backoff-cutoff`* statement determines the maximum amount of time the client is allowed to backoff. The default is two minutes
initial-interval time	The *`initial-interval`* statement sets the amount of time between the first attempt to reach a server and the second attempt to reach a server by recalculating the interval between messages. It is incremented by twice the current interval multiplied by a random number between zero and one. If it is greater than the backoff-cutoff amount, it is set to that amount. The default is ten seconds.
reboot time	When the client is restarted, it first tries to reacquire the last address it had. This is called the INIT-REBOOT state. This is the quickest way to get started if it is still attached to the same network it was attached to when it last ran. The *`reboot`* statement sets the time that must elapse after the client first tries to reacquire its old address before it gives up and tries to discover a new address. The reboot timeout default is ten seconds.
retry time	The *`retry`* statement determines the time that must pass after the client has determined that there is no DHCP server present before it tries again to contact a DHCP server. By default, this is 5 minutes.
select-timeout time	It is possible to have more than one DHCP server serving any given network. It is also possible that a client may receive more than one offer in response to its initial lease discovery message. It may be that one of these offers is preferable to the other (e.g., one offer may have the address the client previously used, and the other may not). The *`select-timeout`* is the time after the client sends its first lease discovery request at which it stops waiting for offers from servers, assuming that it has received at least one such offer. If no offers have been received by the time the select-timeout

Statement	Description
	has expired, the client will accept the first offer that arrives. By default, the select-timeout is zero seconds — that is, the client will take the first offer it sees.
timeout time	The `timeout` statement determines the amount of time that must pass between when the client begins to try to determine its address and the time it decides that it is not going to be able to contact a server. The default is 60 seconds. After the timeout has passed, if there are any static leases defined in the configuration file, or any leases remaining in the lease database that have not yet expired, the client loops through these leases attempting to validate them. If it finds one that appears to be valid, it uses that lease's address. If there are no valid static leases or unexpired leases in the lease database, the client restarts the protocol after the defined retry interval.

9.30.8. Lease Requirements and Requests

The DHCP protocol allows the client to request the server to send it specific information. The protocol also allows the client to reject offers from servers if they do not contain information the client needs, or if the information provided is not satisfactory. There is a variety of data contained in offers that DHCP servers send to DHCP clients. The DHCP client can request any of the DHCP options, see Table 9.16 for a list of options.

Table 9.16. DHCP Client Lease Options

Lease Option	Description
request [option] [, ... option];	The *`request`* statement causes the client to request that any server responding to the client send the client its values for the specified options. Only the option names should be specified in the request statement, not option parameters. For example, `request subnet-mask, routers;`
require [option] [, ... option];	The *`require`* statement lists options that must be sent in order for an offer to be accepted. Offers that do not contain all the listed options are ignored.
send { [option declaration] [, ... option declaration]}	The *`send`* statement causes the client to send the specified options to the server with the specified values. Options that are always sent in the DHCP protocol should not be specified here. The one exception is that the client can specify a requested-lease-time option other than the default requested lease time, which is two hours. The other obvious use for this statement is to send information to the server that allows it to differentiate between this client and other clients or kinds of clients. For example, `send host-name “my-name”;`

9.30.9. Option Modifiers

In some cases, a client may receive option data from the server that is not appropriate for that client, or may not receive information that it needs, and for which a useful default value exists. It may also receive information that is useful, but needs to be supplemented with local information. To handle these needs, these option modifiers are available.

Table 9.17. DHCP Client Option Modifiers

Modifier	Description
append [option declaration];	Use the *`append`* statement if the client should use the values supplied by the server followed by a value you supply. The append statement can only be used for options that allow more than one value to be given. This restriction is not enforced. If you ignore it, the behavior is unpredictable.
default [option declaration];	Use the *`default`* statement to specify a default value if no value was supplied by the server.
prepend [option declaration];	Use the *`prepend`* statement if the client should use a value you supply followed by the values supplied by the server. The prepend statement can only be used for options that allow more than one value to be given. This restriction is not enforced. If you ignore it, the behavior is unpredictable.
supersede [option declaration];	Use the *`supersede`* statement if the client should always use a locally-configured value or values rather than whatever is supplied by the server.

9.30.10. Lease Declarations

A *`lease`* statement consists of the **lease** keyword, followed by a left curly brace ({), followed by one or more lease declaration statements, followed by a right curly brace (}).

```
lease { lease-declaration [ ... lease-declaration ] }
```

The DHCP client may determine after some period of time (see Section 9.30.7) that it is not going to succeed in contacting a server. At that time, it consults its own database of old leases and tests each one that has not yet timed out by pinging the listed router for that lease to see if that lease could work. It is possible to define one or more fixed leases in the client configuration file for networks where there is no DHCP or BOOTP service, so that the client can still configure its address automatically. This is done with the **lease** statement.

Note

The lease statement is also used in the `DHCLIENT.DB` file in order to record leases that have been received from DHCP servers. Some of the syntax for leases as described below is only needed in the `DHCLIENT.DB` file. Such syntax is documented here for completeness.

The following lease declarations are possible:

Table 9.18. DHCP Client Lease Declarations

Declaration	Description
bootp;	The *`bootp`* statement indicates that the lease was acquired using the BOOTP protocol rather than the DHCP protocol. It is never necessary to specify this in the client configuration file. The client uses this syntax in its lease database file.
filename "string";	The `filename` statement specifies the name of the boot filename to use. This is not used by the standard client configuration script, but is included for completeness.

Declaration	Description
fixed-address ip-address;	The `fixed-address` statement sets the IP address of a particular lease. This is required for all lease statements. The IP address must be specified as a dotted quad (e.g., 12.34.56.78).
interface "string";	The `interface` statement indicates the interface on which the lease is valid. If set, this lease will be tried only on a particular interface. When the client receives a lease from a server, it always records the interface number on which it received that lease. If predefined leases are specified in the `DHCLIENT.CONF` file, the interface should also be specified, although this is not required.
option option-declaration;	The `option` statement specifies the value of an option supplied by the server, or, in the case of predefined leases declared in `DHCLIENT.CONF`, the value that the user wants the client configuration script to use if the predefined lease is used.

renew date;
 	
rebind date;
 	
expire date;
	The `renew` statement defines the time at which the DHCP client should begin trying to contact its server to renew a lease that it is using.
	
	The `rebind` statement defines the time at which the DHCP client should begin to try to contact any DHCP server in order to renew its lease.
	
	The `expire` statement defines the time at which the DHCP client must stop using a lease if it has not been able to contact a server in order to renew it.
	
	These declarations are set automatically in leases acquired by the DHCP client, but must be configured in predefined leases: a predefined lease whose expiration time has passed will not be used by the DHCP client. Dates are specified as follows:
	
	`weekday year/month/day hour``:`*`minute`*`:`*`second`*
	
	W YYYY/MM/DD HH:MM:SS
	
	W is the day of the week, from zero (Sunday) to six (Saturday).
	
	YYYY is the year, including the century.
	
	MM is the number of the month, from 01 to 12.
	
	DD is the day of the month, counting from 01.
	
	HH is the hour, from 00 to 23.
	
	MM is the minute, from 00 to 59.
	
	SS is the second, from 00 to 59.
	
	The time is always in Greenwich Mean Time, not local time.

server-name "string";	The `server-name` statement specifies the name of the boot server name to use. This is not used by the standard client configuration script.
script "script-name";	The `script` statement specifies the file name of the DHCP client configuration script. This script is used by the DHCP client to set the interface's initial configuration prior to requesting an address, to test the address once

Declaration	Description
	it has been offered, and to set the interface's final configuration once a lease has been acquired. If no lease is acquired, the script is used to test predefined leases, if any, and also called once if no valid lease can be identified. The default value for “script-name” is `IP$:DHCLIENT-SCRIPT.COM`.

9.30.11. Other Declarations

Table 9.19. DHCP Client Other Declarations

Declaration	Description
reject ip-address;	The *`reject`* statement causes the DHCP client to reject offers from servers who use the specified address as a server identifier. This can be used to avoid being configured by rogue or misconfigured DHCP servers, although it should be a last resort; better to track down the bad DHCP server and fix it.

9.30.12. Example

```
# template of IP$:DHCLIENT.CONF
send host-name "lambda2";
send dhcp-lease-time 3600;
prepend domain-name-servers 127.0.0.1;
request subnet-mask, broadcast-address, time-offset, routers,
 domain-name, domain-name-servers, host-name;
require subnet-mask, domain-name-servers;
timeout 60;
retry 60;
reboot 10;
select-timeout 5;
initial-interval 2;
script "IP$:dhclient-script.com";
reject 10.10.10.10;
# reject the offer from this DHCP server
```

The first line, starting with the #, is a comment line. The last line rejects the offer from the DHCP server with an IP address of 10.10.10.10. This is not a simple `DHCLIENT.CONF` file. In many cases, it is sufficient to just create an empty `DHCLIENT.CONF` file and let the DHCP client use default values.

9.30.13. Troubleshooting the DHCP Client

9.30.13.1. How do I know the DHCP client has configured my network successfully?

There are two ways to do it:

Check if the `IP$DHCP_CLIENT` logical is equal to "1", you can do:

```
$ SHOW LOGICAL IP$DHCP_CLIENT
"IP$DHCP_CLIENT" = "1" (LNM$SYSTEM_TABLE)
$
```

Run the **$ IP CONFIGURE /INTERFACE** command.

1. Check the line DHCP Client for which the interface is enabled.

2. Check that the line DHCP Client Flag: is set.

```
$ IP CONFIGURE /INTERFACE
VSI TCP/IP for OpenVMS Network Configuration Utility V10.5(nn)
[Reading in MAXIMUM configuration from IP$:IP.EXE]
[Reading in configuration from IP$:NETWORK_DEVICES.CONFIGURATION]
NET-CONFIG>SHOW
Interface                            Adapter   CSR Address   Flags/
Vector
---------                            -------   -----------
 ------------
se0 (Shared OpenVMS Ethernet/FDDI)   -NONE-    -NONE-        -NONE-
 [TCP/IP$: 10.10.10.10]
 [VMS Device: EWA0, Link Level: Ethernet]
 DHCP Client for the interface is enabled
Official Host Name:                 dumdum.fuges.com
Domain Nameserver:                  127.0.0.1
Timezone:                           EST
Timezone Rules:                     US/EASTERN
Load UCX $QIO driver:               TRUE
Load PWIP (Pathworks) driver:       TRUE
DHCP Client Flag:                   1
NET-CONFIG>QUIT
$
```

9.30.13.2. What if I cannot ping an IP address on the internet?

If you can ping the same IP address from another host and the network interface has been configured by the DHCP client, check the gateway and route configuration on the host.

9.30.13.3. What if I can ping a host by its IP address but not by its name?

- The DNS client on the host may not be configured right. Type

  ```
  $ SHOW LOGICAL IP$NAMESERVERS
  ```

  to make sure the DNS client information is correct.

- The DNS server may be down.

9.30.13.4. Why is the local address "0.0.0.0" when I use "$ IP CONFIGURE /INTERFACE" and then use “SHOW”?

The DHCP client has failed to allocate an IP address. The possible reasons and solutions are:

Reason	**Solution**
There is no DHCP server on the net.	Set up a DHCP server.
The DHCP server is not configured correctly.	Modify the DHCP server configuration.
The DHCP client is configured to reject the DHCP server.	Reconfigure the DHCP client to not reject the DHCP server.
The hostname selection process failed.	Use another host name.

Reason	Solution
There are no IP addresses available in the DHCP server.	Increase the IP address on the DHCP pool server.

9.30.13.5. Where can I find the status information of the DHCP client?

- The file `IP$:DHCLIENT-SCRIPT-ENV.TMP` contains the most recent environment variables used by the DHCP client script file to configure the network.
- The file `IP$:DHCLIENT.DB` contains the DHCP client lease history.
- The file `IP$:DHCLIENT.LOG` contains information about the DHCP client process.

The `IP$:DHCLIENT.LOG` file is not created by the default setting of the DHCP client. To create this log file, configure the DHCP client to enable the error and debug logging:

```
$ IP CONFIGURE /SERVER
SERVER-CONFIG>SELECT DHCLIENT
SERVER-CONFIG>SET PARAMETERS
Add Parameter: DEBUG 7
Add Parameter:
SERVER-CONFIG>WRITE
SERVER-CONFIG>RESTART
```

(Debug levels are listed in Table 9.14.)

Chapter 10. Managing the XDM Server and X11-Gateway Configuration

This chapter explains how to configure the XDM (X Display Manager) server of VSI TCP/IP to manage remote systems running X (X Window System) servers. Also, this chapter includes information about the X11-Gateway program

10.1. Understanding X Display Management

The XDM server of VSI TCP/IP provides login services to X servers through graphical dialog boxes. From a user's perspective, logging in at a XDM-managed X server of VSI TCP/IP is no different than logging in at an OpenVMS system console.

Initially, the user sees the standard DECwindows banner and login dialog. After the user supplies a valid user name and password, the DECwindows Session Manager starts and launches any applications configured for automatic startup. After the user logs out from the Session Manager, the banner and login dialog reappear.

Users of X11R4 (and later) servers benefit from XDM because they can take advantage of the DECwindows Session Manager as if they were using a host.

Users can also specify which host they want to use. This feature is provided by XDMCP (X Display Management Communications Protocol), used for communication between X servers and XDM servers.

From a system manager's perspective, XDM provides a convenient way to extend the LOGINOUT service to users on remote X servers and to specify which X servers are managed by each XDM server host.

The XDM server of VSI TCP/IP is based on X11R6 sources from MIT.

10.2. Accessing the XDM Server

Before an X server can be managed, you must grant it access to your XDM server. In a network of dozens or hundreds of X servers, it may not be practical to manage every X server that requests XDM service. In these cases, you can restrict access to your XDM server. The manner in which access is granted depends on the version of the X server and how it requests XDM service.

X11R4 (and later) X servers can generate three basic types of management requests, as explained in Table 10.1.

Table 10.1. XDMCP Requests

Request Type	**Description**	**XDM Server Response**
Direct	Sent by X server to a specific host	If the XDM server is configured to accept requests from the user's X server, the host produces a login dialog on that server. If the XDM server is configured to refuse service to the user's X server, no login dialog appears, and the user must try a different host. For information on selectively rejecting XDMCP direct requests, see Section 10.7.1.

Request Type	Description	XDM Server Response
Broadcast	Broadcast by X server to all hosts on the local network	The XDM server responds to broadcast requests as if they were direct requests. Broadcasts can result in several XDM servers accepting the request; it is then up to the X server to determine which host to use. Some X servers simply use the first host to respond. Others present a list of responding hosts in a chooser dialog box, and the user selects a host from the list.
Indirect	Sent by X server to a specific host	If the host's XDM server is configured to handle indirect requests from the X server, the host either forwards the request to another XDM server host or returns a list of XDM servers from which the user can choose. **Note** Only X11R5 and X11R6 XDM servers can properly handle forwarded requests. *The XDM server of VSI TCP/IP does not handle indirect requests.*

From the user's perspective, broadcast and indirect requests provide similar benefits. From an administrator's perspective, however, indirect requests allow the XDM server administrator to select an XDM server for users.

Consider a situation in which `BROWN.FLOWERS.COM,` an X terminal, is configured to make indirect requests for display management to an OpenVMS host, `WHORFIN.FLOWERS.COM`, running the XDM server of VSI TCP/IP. The administrator of WHORFIN can specify which XDM servers will manage BROWN by forwarding inquiries to another XDM server.

X11R3 servers cannot take advantage of XDMCP, which was not introduced until X11R4. Instead, X11R3 servers do not allow users to choose a host. For information on managing X11R3 servers, see Section 10.8.

10.2.1. Special Features of the XDM Server of VSI TCP/IP

Unlike many XDM servers, the VSI TCP/IP XDM server does not control X sessions after users have logged in. Instead, it starts the processes that produce the login dialog and authenticate the user, then passes control to the DECwindows Session Manager. When the user terminates the session using the Session Manager, the X session ends and XDM starts a new login cycle. This method allows you to change XDM server configuration, stop the XDM server, and restart the XDM server without interrupting any X sessions already in progress.

Configuration of a remote user's session is identical to the customization of a local DECwindows user's session. Traditionally, XDM servers on UNIX systems allow user environments to be configured by scripts (command procedures) that are run before, during, and upon ending each X session. The VSI TCP/IP XDM server, however, takes advantage of the DECwindows STARTLOGIN and LOGINOUT processes to eliminate the need for extra command procedures.

The VSI TCP/IP XDM server does not handle indirect requests.

10.3. XDM Administrative Tasks

The following are common administrative tasks for the XDM server:

- Enabling the XDM server (see Section 10.4).
- Modifying the XDM server configuration (see Section 10.5).
- Controlling the XDM server (see Section 10.6).
- Controlling access to the XDM server (see Section 10.7).
- Managing X11R3 displays (see Section 10.8).

With the exception of controlling the XDM server, each task requires you to modify XDM configuration files and update the XDM server with the new information (as described in Section 10.6).

10.4. Enabling and Starting the XDM Server

When VSI TCP/IP is first installed, the XDM server is disabled, and you must enable it. After the server is enabled, it starts automatically when VSI TCP/IP starts.

1. Issue the following DCL command:

   ```
   $ IP CONFIGURE /SERVER
   ```

2. At the SERVER-CONFIG prompt, enter:

   ```
   SERVER-CONFIG>ENABLE XDM
   ```

3. To verify that the XDM server is enabled, enter:

   ```
   SERVER-CONFIG>SHOW XDM
   ```

   If no asterisk (*) appears to the left of XDM in the output, the service is enabled.

4. To exit the configuration utility, enter:

   ```
   SERVER-CONFIG>EXIT
   ```

5. Restart the master server:

   ```
   $ @IP$:IP$SYSTARTUP.COM
   ```

10.5. Modifying the XDM Server Configuration

XDM is configured through X resources and files specified through X resources. When the XDM server starts or you reload its configuration, it configures itself according to the contents of its master configuration file, `IP$:XDM.CONFIGURATION`, including reading the contents of files specified in the master configuration file.

Resource parameters for VSI TCP/IP XDM are listed in Table 10.2 and Table 10.3.

These resources modify the global behavior of XDM. Because the resource manager uses colons to separate the name of the resource from its value and dots to separate resource name parts, XDM substitutes underscores for both dots and colons when generating the resource name.

Note

If you import an XDM configuration file from a UNIX system, the startup, session, and authorize resources have no effect on the VSI TCP/IP XDM server.

The following is a sample `XDM.CONFIGURATION` file:

```
DisplayManager.logFile:           IP$:xdm-errors.log
DisplayManager.servers:           IP$:xdm.servers
DisplayManager.accessFile:        IP$:xdm.access
DisplayManager.removeDomainname:  false
```

Table 10.2. XDM Server Resources

Configuration File Resource	Description
DisplayManager.accessFile	Specifies a file containing a database of host names that are either allowed direct access to this host or to which queries should be forwarded. For details, see Section 10.7. The equivalent file on UNIX systems is /usr/lib/X11/xdm/Xaccess. Default: `IP$:XDM.ACCESS`.
DisplayManager.debugLevel	Sets the debug output level. Default: 0. You can also set this value while the XDM server is running with the command IP NETCONTROL XDM DEBUG.
DisplayManager.logFile	Specifies the file in which errors are logged when the logType resource is FILE. The equivalent file on UNIX systems is /usr/lib/X11/xdm/xdm-errors. Default: `IP$:XDM.LOG`.
DisplayManager.logType	Specifies where XDM server messages are logged: OPCOM, SYSLOG, FILE, or STDOUT. Default: OPCOM.
DisplayManager.removeDomainName	Specifies whether XDM removes the domain name portion of host names if they are the same as the domain name of the local host. Removing the domain name is useful because name resolvers typically create fully qualified host names when computing display names for XDMCP clients. Default: true.
DisplayManager.requestPort	Specifies the UDP port number to which the XDM server listens for incoming XDMCP requests. Unless you need to debug the system, do not change the value from the default. Default: 177 (the standard for XDMCP).
DisplayManager.servers	Specifies a file containing a list of foreign X servers to manage. If the file specification is prefixed with an @ sign, it contains one listing per line. A foreign X server is an X display that does not support the XDMCP protocol. The equivalent file on UNIX systems is /usr/lib/X11/xdm/Xservers. Default: `IP$:XDM.SERVERS`.

You can use the configuration parameters in the Table 10.3 to adjust the way the XDM server transfers control of the X terminal to a DECwindows process. The default values for these parameters should be suitable for ordinary DECwindows usage.

Under normal operation, the XDM server creates an X connection to the managed display before starting a process to invoke the DECwindows login screen. When the login completes and the DECwindows Session Manager starts, it hands off the initial X connection to the process. The initial X connection closes down automatically when the session ends, causing the X display to reset and either shut down or restart the XDM login chooser (depending on how the X terminal or software is configured).

Table 10.3. Advanced XDM Resources

Configuration Resource	Description
DisplayManager.loginCheckInterval	The number of seconds the XDM server waits between checks to see if the DECwindows login process has begun. Default: 5.
DisplayManager.loginTries	The number of checks the XDM server makes to see if the DECwindows login process has begun. Default: 12.
DisplayManager.sessionCheckInterval	The number of seconds the XDM server waits between checks to see if the DECwindows login process has handed control to the user's Session Manager. Default: 10.
DisplayManager.sessionTries	The number of checks the XDM server makes to see if the user's Session Manager started. Default: 30.
DisplayManager.bypassSessionCheck	If set to true, the check for the start of a Session Manager is bypassed. Default: false.
DisplayManager.sessionManagers	A comma-separated list of image names, without device or directory specifications, that the XDM server should consider to be Session Managers for a DECwindows session. Default: `DECW$SESSION.EXE`.

10.6. Controlling the XDM Server

This section describes the following XDM tasks:

- Checking the status of the XDM server
- Starting the XDM server
- Stopping the XDM server
- Restarting the XDM server
- Reloading the XDM configuration

For most of these tasks, you use the **IP NETCONTROL** utility.

10.6.1. Checking the Status of the XDM Server

To check the status of the VSI TCP/IP XDM server, enter:

```
$ IP NETCONTROL XDM SHOW
```

The following example shows the information generated by this command on a system named WHORFIN that manages three X terminals, BANZAI, BROWN, and MIGUEL. When the terminal is displaying a DECwindows login screen, the following command output changes to show the words "not logged in" instead of a process identifier number.

```
$ IP NETCONTROL XDM SHOW
Connected to NETCONTROL server on "WHORFIN"
< WHORFIN.YOYDYNE.COM
< Network Control 10.5(8) at Wed 8- Apr-2017 11:37AM-EST
< XDM Current Managed Displays:
<  Display BANZAI.FLOWERS.COM:0.0 Type XDMCP Process 20E002A8
<  Display BROWN.FLOWERS.COM:0.0 Type XDMCP Process 20E00289
<  Display MIGUEL.FLOWERS.COM:2005.0 Type XDMCP Process 20E00242
< OK
```

10.6.2. Starting the XDM Server

This section describes how to start the VSI TCP/IP XDM server.

Note

The XDM server is automatically started when VSI TCP/IP starts.

Before starting the VSI TCP/IP XDM server, make sure it is enabled. For details, see Section 10.4.

To start the VSI TCP/IP XDM server, enter:

```
$ IP NETCONTROL XDM START
```

When the XDM server starts, it reads the master configuration file, `IP$:XDM.CONFIGURATION`, to determine which configuration files to read, then reads them. (For information on modifying the master configuration file, see Section 10.5). You can specify other configuration files simply by modifying the contents of the master configuration file and then restarting the XDM server (see Section 10.6.4), or by reloading the XDM configuration files (see Section 10.6.5).

The following example shows the information generated by this command:

```
$ IP NETCONTROL XDM START
< XDM Server Started, process id pid
```

10.6.3. Stopping the XDM Server

To stop the VSI TCP/IP XDM server, enter:

```
$ IP NETCONTROL XDM SHUTDOWN
```

The following example shows the information generated by this command:

```
$ IP NETCONTROL XDM SHUTDOWN
< XDM Server Shutdown
```

10.6.4. Restarting the XDM Server

Restarting the XDM server is a convenient alternative to stopping and starting the XDM server as described in Section 10.6.2 and Section 10.6.3.

When the XDM server restarts, it reads the master configuration file `IP$:XDM.CONFIGURATION` to determine which configuration files to read, then reads them.

To restart the XDM server, enter:

```
$ IP NETCONTROL XDM RESTART
```

The following example shows the information generated by this command:

```
$ IP NETCONTROL XDM RESTART
Connected to NETCONTROL server on "LOCALHOST"
< WHORFIN.FLOWERS.COM Network Control 10.5(8) at Tue 30-Apr-2017 12:47AM-
EST
< XDM Server Started, process id 2040024B
```

For information on modifying the master configuration file, see Section 10.5.

10.6.5. Reloading the XDM Configuration

Changes to XDM server configuration take effect when the XDM server is started or restarted or when the configuration files are reloaded. Reloading the XDM server allows you to reload the XDM server configuration files without restarting the XDM server and interrupting connections between X servers and the XDM server.

When the XDM server reloads its configuration, it first reads the `IP$:XDM.CONFIGURATION` master configuration file to determine which other files to read, then reads them. For information on modifying the master configuration file, see Section 10.5.

To reload the XDM configuration files, enter:

```
$ IP NETCONTROL XDM RELOAD
```

The following example shows the information generated by this command produces:

```
$ IP NETCONTROL XDM RELOAD
< OK: XDM server configuration reloading
```

10.7. Controlling Access to the XDM Server

Access to the VSI TCP/IP XDM server by X11R4 (and later) servers is controlled through the `IP $:XDM.ACCESS` file. This file contains the following information:

- Displays the XDM server will manage
- Displays the XDM server will not manage
- Displays and the XDM servers that will manage them
- Macro definitions

The following sections explain how to edit `XDM.ACCESS` to handle direct, broadcast, and indirect requests.

10.7.1. Handling Direct and Broadcast Requests

To manage X servers that make direct or broadcast requests, add their names to the `XDM.ACCESS` file as follows:

- To manage a specific X server, include a line with the server host name.

- To reject a specific X server, include a line with the server host name preceded by an exclamation point (!).
- To manage any X server that requests management, include a line consisting of an asterisk (*).

For convenience, you can define lists of hosts in macros in `XDM.ACCESS`. To define a macro, include a line at the top of the file in the following format:

```
%macroname host_list
```

You can then use the macro name (including the leading percent sign (%) in the file.

The following sample `XDM.ACCESS` file allows any host to be managed by the XDM server.

```
# Access control file for XDMCP connections
#
* #any host can get a login window
```

10.8. Managing X11R3 Displays

Although X11R3 X servers do not allow users to select the host they are going to log into, they can still be managed by the VSI TCP/IP XDM server. Once an X server is managed by one host, however, the user has limited ability to change to another host.

To configure the VSI TCP/IP XDM server to manage a specific X11R3 display:

1. Add the X11R3 display to the VSI TCP/IP XDM server's `XDM.SERVERS` file (see Section 10.8.1).
2. Configure the X11R3 display to grant access to the VSI TCP/IP host (see Section 10.8.2).
3. Make sure the X11R3 display is not already managed by another XDM server (see Section 10.8.3).
4. Reload the XDM server configuration (see Section 10.8.3).

10.8.1. Specifying X11R3 Displays

The XDM server obtains a list of X displays to manage from the `IP$:XDM.SERVERS` file whenever it starts or you reload its configuration files.

- The XDM server obtains its list of X11R3 servers from the file specified in the DisplayManager.servers resource in the XDM master configuration file, `IP$:XDM.CONFIGURATION`.

Add an entry to `XDM.SERVERS` in the following format:

```
server_name foreign
```

- *`server_name`* is the name of the X11R3 server you want to manage, in the standard X format (*`hostname:server number`*[*`.screen number`*]).
- "foreign" indicates that *`server_name`* does not use XDMCP.

The following sample `XDM.SERVERS` file contains entries for two X11R3 servers, BANZAI:0 and MIGUEL:0.

```
banzai.flowers.com:0 foreign
miguel.flowers.com:0 foreign
```

You can also specify classes of displays that share similar requirements.

10.8.2. Setting Up Host Access on the Display

Before the XDM server can produce a login dialog box on an X11R3 display, it must be granted access to the display via the X host access mechanism.

Note

The VSI TCP/IP XDM server does not support the MIT-MAGIC-COOKIE-1 mechanism.

The XDM server can only present the LOGINOUT dialog on X11R3 servers on which host access restrictions are entirely disabled, or enabled specifically for your VSI TCP/IP XDM server host.

For many X servers, you can disable access restrictions with the **xhost** client, or by modifying configuration files before starting the server. Refer to your X server administration documentation for information on granting access to remote hosts.

10.8.3. Ensuring No Other Host Is Managing the Display

If an X11R3 display is already being managed by another XDM server, either remove the display from the other XDM server's configuration (usually the XSERVERS file) or change the X server's host access list so the other XDM server cannot access the display. Cancel the login dialog and verify that the other XDM server's login dialog does not return.

10.8.4. Reloading the XDM.SERVERS File

Reload the `IP$:XDM.SERVERS` file (or stop and restart the XDM server) after you have:

- Specified which X11R3 servers you want to manage in the `IP$:XDM.SERVERS` file
- Granted access to your XDM server
- Made sure that no other XDM server is managing the X servers

The XDM server reads the `XDM.SERVERS` configuration file when it starts (see Section 10.6.2), restarts (see Section 10.6.4), and when you reload its configuration (see Section 10.6.5).

10.9. X11-Gateway Configuration

The VSI TCP/IP X11-Gateway program provides X (X Window System) connectivity between a DECnet-only host and an IP-only host by a VSI TCP/IP node as an application gateway. The X11-Gateway is bidirectional; it functions as a gateway from a DECnet-only X client to an IP-only X server, or vice versa.

The gateway node requires VSI TCP/IP and DECnet software only. There is no requirement for the gateway to be running any X software. The gateway software can support multiple X Windows connections simultaneously.

10.9.1. X11-Gateway Concepts

Before configuring the X11-Gateway, be sure you understand the following terms:

Term	**Description**
`Client`	The node executing the X application
`Gateway`	The node connected to an IP network and a DECnet network. Information from the *`client`* on one network is passed through the *`gateway`* to the *`server`* on the other network.
`Server`	The node running the X server software. (Typically a host with a mouse, keyboard, and at least one bit-mapped screen.)
`Server number`	A unique number identifying the X11-Gateway to VSI TCP/IP, DECnet, and the X software on the client and server nodes. Each configured gateway is assigned a unique server number by the system manager.

To avoid conflicts with current and future versions of DECwindows software, use numbers beginning with 10 and increment for each new gateway. For example, assign the number 10 to the first X11-Gateway, the number 11 to the second, and so on. As you remove X11-Gateways from the system, you can reuse their server numbers.

10.9.2. Allowing an IP Client Access to a DECnet Server

To configure the X11-Gateway host to allow an IP client access to a DECnet server:

1. Choose an X11-Gateway server number between 10 and 999.

2. Create a TCP port number by adding 6000 to the server number. For example, if the server number is 13, the TCP port number is 6013. TCP port 6000 is used by DECwindows servers. Select port numbers starting at 6010 to avoid conflicts with DECwindows.

3. Add the X11-Gateway service to the list of TCP/IP services using the Server Configuration Utility (SERVER-CONFIG). Information about using this utility and its commands is provided in the *VSI TCP/IP Administrator's Guide: Volume I*.

   The prefix "X11-GATEWAYxxx" is added to the name of the service you install; xxx is the server number you selected in Step 1. For example, for server number 10, the service you add using SERVER-CONFIG is X11-GATEWAY10. SERVER-CONFIG provides the X11-GATEWAY13 service, but the number 13 has no significance; this service is provided only as an example. You can use this service or any number of your choice between 10 and 999. If you chose 13 as the server number in Step 1, you can enable the existing X11-GATEWAY13 service using SERVER-CONFIG. There is no need to add this service, as it has already been added it. The following example adds an X11-Gateway server number of 12.

   ```
   $ IP CONFIGURE/SERVER
   VSI TCP/IP for OpenVMS Server Configuration Utility 10.5(nnn)
   [Reading in configuration from IP$:SERVICES.MASTER_SERVER]
   SERVER-CONFIG>ADD X11-GATEWAY12
   [Adding new configuration entry for service "X11-GATEWAY12"]
   Protocol: [TCP]
   TCP Port number: 6012
   ```

```
Program to run: IP$:X11-GATEWAY.EXE
[Added service X11-GATEWAY12 to configuration]
[Selected service is now X11-GATEWAY12]
SERVER-CONFIG>RESTART
Configuration modified, do you want to save it first ? [YES]
[Writing configuration to
IP$COMMON_ROOT:[IP]SERVICES.MASTER_SERVER]
%RUN-S-PROC_ID, identification of created process is 20E0026B
SERVER-CONFIG>EXIT
```

4. Define the X11-Gateway logical names. The X11-Gateway accepts connections from the IP network and routes the X protocol requests to a specific DECnet X server. Specify the server using the following logical names:

   - `IP$XGATEWAY_TCPIP_`*`server_number`*`_HOSTNAME`

     Specifies the DECnet host name. This logical name must be defined, but do not include the colons from the DECnet host name.

   - `IP$XGATEWAY_TCPIP_`*`server_number`*`_SERVER`

     The X server number of the node where the X client application is displayed. Most hosts run one X server, which is designated as server 0 (zero). This logical name is optional if the DECnet X server number is zero. The X11-Gateway assumes a DECnet X server number of zero if you do not define this logical. You should define the logical if the DECnet X server number is not zero.

   In the following example, the X11-Gateway server number is 12. The X11-Gateway accepts connections from the IP network and gateways them to X server 1 on the DECnet node BRONX. Assign values to these logical names using commands like the following examples:

   ```
   $ DEFINE/SYSTEM/EXEC IP$XGATEWAY_TCPIP_12_HOSTNAME BRONX
   $ DEFINE/SYSTEM/EXEC IP$XGATEWAY_TCPIP_12_SERVER 1
   ```

   Insert these logical name definitions in the system startup procedure so they are invoked after the DECnet and VSI TCP/IP startup procedures execute.

10.9.2.1. Running an IP Client on a DECnet Server

To bring up the X application on the IP-client-to-DECnet-server configuration:

1. On the DECnet server, authorize DECnet connections from user SYSTEM on the gateway node. If the IP$SERVER process on the gateway node has been started under a user name other than SYSTEM, that user should also be authorized. A less secure, but more reliable, method is to authorize the "*" user. See Section 10.9.4.

   To provide connection authorization on ULTRIX or OpenVMS, use the Session Manager.

2. On the IP client, set the display variable to point to the X11-Gateway host. Use the X11-Gateway server number for the display server number.

   To modify the DISPLAY environment on OpenVMS systems, use the **SET DISPLAY** command.

3. On the IP client, start the X application.

   For example:

- The IP X client node is the UNIX node pelham.flowers.com. The X11-Gateway node is amtrak.flowers.com (TCP/IP) and AMTRAK: (DECnet). The X11-Gateway server number is 12. BRONX:: is an OpenVMS DECnet X server.
- On the BRONX:: node, the user authorizes protocol DECNET, node AMTRAK, and user "*" using the Security pull-down menu in the Session Manager.
- On the pelham.flowers.com UNIX node, the user runs setenv to set the DISPLAY environment variable to the value amtrak.flowers.com:12.1. The user can then invoke an X application.
- The X application appears on the BRONX:: node.

10.9.3. Allowing a DECnet Client Access to an IP Server

To configure the X11-Gateway to allow a DECnet client access to an IP server:

1. Choose an X11-Gateway server number as described in Section 10.9.2.
2. Add the X11-Gateway to the list of DECnet objects. The object name for the X11-Gateway has the value X$X`server_number`. For example, for a server number of 17, set the object with this command:

   ```
   $ RUN SYS$SYSTEM:NCP
   NCP>DEFINE OBJECT X$X17 NUMBER 0 FILE IP$:X11-GATEWAY.EXE
   NCP>SET OBJECT X$X17 NUMBER 0 FILE IP$:X11-GATEWAY.EXE
   ```

   If the DECnet default account is disabled, the command should include a valid USERNAME and PASSWORD on the gateway system. In this example the X11-Gateway server number is 17:

   ```
   $ RUN SYS$SYSTEM:NCP
   NCP>DEFINE OBJECT X$X17 NUMBER 0 FILE -
   _ IP$:X11-GATEWAY.EXE USER SYSTEM PASS systempassword
   NCP>SET OBJECT X$X17 NUMBER 0 FILE IP$:X11-GATEWAY.EXE -
   _ USER SYSTEM PASS systempassword
   ```

3. Define the logical names. The X11-Gateway accepts connections from the DECnet network and directs X protocol requests to a specific IP X server. Specify the server is using the following logical names:

   - `IP$XGATEWAY_DECNET_server_number_HOSTNAME`

     Specifies the IP X server host name. You must define this logical name.

   - `IP$XGATEWAY_DECNET_server_number_SERVER`

     Specifies the X server number, which is typically set at 0 (zero) to indicate that a single server is being used. If a second server is in use, set this value to 1, and so on. If this logical name is not defined, the default value is 0. For example, the X11-Gateway server number is 17. The X11-Gateway accepts connections from the DECnet network and directs them to X server number 1 on the IP node englewood-nj.flowers.com. The logical names are then defined as:

     ```
     $ DEFINE/SYSTEM/EXEC IP$XGATEWAY_DECNET_17_HOSTNAME -
     _$ ENGLEWOOD-NJ.FLOWERS.COM
     $ DEFINE/SYSTEM/EXEC IP$XGATEWAY_DECNET_17_SERVER 1
     ```

Insert these logical name definitions into the system startup procedure so the definitions occur after the invocation of the DECnet and VSI TCP/IP startup procedures.

10.9.3.1. Running the DECnet Client on the IP Server

To bring up the X application on the DECnet-client-to-IP-server configuration:

1. On the IP server, authorize IP connections from the gateway node. X-over-IP does not provide user name information. If a user name is required as part of the authorization (for example, on OpenVMS) use a "*" value. Connection authorization is usually accomplished with the **xhost** command on UNIX systems, or with the OpenVMS or ULTRIX Session Manager.

2. On the DECnet client, set the display variable to point to the X11-Gateway host. The X11-Gateway server number should be used for the display server number. On UNIX hosts (including ULTRIX) use the **setenv** command; on OpenVMS systems, use the SET DISPLAY command.

3. On the DECnet client, execute the X Windows application.

   For example:

   - The DECnet X client is an OpenVMS node METRO::. The X11-Gateway node is DENISE:: (DECnet) and `DENISE.FLOWERS.COM (TCP/IP)`. The X11-Gateway server number is 17. The IP X server is the UNIX host `englewood-nj.flowers.com`.

   - On englewood-nj the user authorizes node denise.flowers.com by entering the command:

     ```
     % xhost +denise.flowers.com
     ```

   - On METRO, the user issues the command:

     ```
     $ SET DISPLAY/CREATE/NODE=DENISE/TRANS=DECNET/SERVER=17
     ```

   - The user can then invoke an X application, which appears on the englewood-nj server.

10.9.4. X11-Gateway Security

The X11-Gateway node does not attempt to restrict connections it receives from the network. As a result, any node or user on the client side of the gateway can access the server, essentially allowing a client user to monitor all activity on the X server via the X11-Gateway.

For IP-client-to-DECnet-server connections, you can reduce risk by using the ACCEPT-HOSTS/ ACCEPT-NETS capabilities of the VSI TCP/IP master server on the gateway host. For more information, see *VSI TCP/IP Administrator's Guide: Volume I.*

For DECnet-client-to-IP-server connections, your risk can be reduced by using the NCP utility to limit access to the gateway host.

VSI does not recommend running the X11-Gateway on untrusted networks unless other restrictions have been imposed by the system manager.

10.9.5. X11-Gateway Debugging

The best programs for testing client-to-gateway to server connectivity are based on the Xlib routines (as opposed to widget toolkits).

The ICO program is available on most X implementations (for example, /usr/bin/X11/ico or DECW $EXAMPLES:ICO) and works well for debugging problems. The ICO program opens a window and causes an icosahedron to bounce around the window. When this program works, X works as well. Exit the OpenVMS version of this program by pressing **Ctrl/Y** in the window from which you invoked ICO. You can also use the **IP X11DEBUG** command to debug OpenVMS IP client problems.

10.9.5.1. Selected Error Numbers from ERRNO.H

Table 10.4 lists error values from the `ERRNO.H` file.

Table 10.4. ERRNO.H Error Values

Error	Value	Description
ENETUNREACH	51	The IP network you are trying to contact is currently unreachable.
ECONNRESET	54	The connection was reset by the remote node. This typically occurs when the remote host has rebooted and the local host attempts to transmit on a stale connection.
ETIMEDOUT	60	The connection timed out during the open.
ECONNREFUSED	61	The connection was refused. This occurs when a connection is attempted to a nonexistent server process.
EHOSTUNREACH	65	There is no route to the host you are trying to contact.

10.9.5.2. X11-Gateway Error Messages

The X11-Gateway node transmits NETWORK class operator messages when an error is encountered. You can change the level of information supplied by X11-Gateway messages by defining the logical name `IP$XGATEWAY_DEBUG_LEVEL` in the system table. Set this value to:

- 0 — to receive fatal errors
- 1 — for debugging messages
- 2 — for informational messages

For example, to select debug level 1:

```
$ DEFINE/SYSTEM/EXEC IP$XGATEWAY_DEBUG_LEVEL 1
$ @IP$:IP$SYSTARTUP.COM
```

If the logical name does not exist, the DEBUG level defaults to a value of zero. All error messages from the X11-Gateway are prefixed with "Xgateway:". The errno values can be translated by examining Table 10.4. Status values are OpenVMS error values that you can examine using the command **WRITE SYS$OUTPUT F$MESSAGE** (Status).

Chapter 11. Configuring VSI TCP/IP SNMP Services

This chapter explains how to configure VSI TCP/IP SNMP (Simple Network Management Protocol) agents. SNMP agents are the hosts managed by SNMP.

11.1. Understanding SNMP

SNMP is a protocol from which hosts called *`network management stations`* can obtain and modify information on other network hosts called SNMP agents.

Typical SNMP-based network management systems employ a host that has been configured as an SNMP agent from which the current network configuration of other network nodes can be analyzed and modified. Such management systems often provide graphical interfaces for these tasks.

11.1.1. SNMP Managers, Agents, and Traps

SNMP-managed networks typically include the following network entities:

Network Management Station (NMS)	A node requests information about, or makes changes to, remote nodes. The NMS interface is completely vendor-specific. VSI TCP/IP performs basic SNMP via the IP SET and **IP SHOW** commands using the **/SNMP_HOST** qualifiers (see Section 11.13).
SNMP agent	This software allows the local network configuration to be examined or modified by an NMS. VSI TCP/IP provides an SNMP agent in the form of the SNMP service. The data managed by an SNMP agent is called the Management Information Base (MIB). The VSI TCP/IP SNMP agent supports the current Internet standard MIB known as MIB-II, described in the file RFC1213-MIB. VSI TCP/IP manages MIBs according to the IETF draft `draft_ref_dhcp_server_mib-02.txt`. Both RFC1213-MIB and “draft_ref_...” are found on the IETF website. VSI TCP/IP SNMP agent software is extensible. To have private MIBS served by the VSI TCP/IP SNMP agent, develop a shareable image that exports the APIs in the private MIBs plus the routine need to access the MIB variables. See the *VSI TCP/IP Programmer’s Reference*.
Traps	Traps receive requests such as non-authenticated SNMP requests that are not handled directly by the SNMP agent. Traps are sent only to clients configured to receive traps, as defined in the SNMP agent configuration file (`SNMPD.CONF`). The agent supports all traps defined in the SNMP protocol, except EGP-Neighbor-Loss, Warm-Start, and Enterprise-Specific. Before NMS can obtain or modify network configuration data, the NMS must be authenticated by the agent. The agent compares a

	community string sent by the manager to the local read or write community strings. The agent must authenticate community strings sent by the NMS before authenticating requests: • For read operations, the NMS string must match the agent's read community string. • For write operations, the NMS string must match the agent's write community string.

11.2. Configuring VSI TCP/IP SNMP Services

The following is an overview of how to configure a host as an SNMP agent. Each step is discussed in detail following this overview.

1. Enable the SNMP service (see Section 11.2.1).
2. Configure an SNMP subagent by setting the subagent image (see Section 11.4).
3. Edit the SNMP configuration file (see Section 11.7).
4. Restart the master server.

When the VSI TCP/IP SNMP agent starts, it obtains configuration data from the `IP$:SNMPD.CONF` file. Since the `SNMPD.CONF` file does not exist, you need to edit it using a text editor.

Note

SNMP parameters cannot be modified with the SNMP-CONFIG utility. The **IP CONFIGURE / SNMP** command is not supported.

11.2.1. Enabling the SNMP Service

To enable the SNMP service using the SERVER-CONFIG utility:

1. Start SERVER-CONFIG:

   ```
   $ IP CONFIGURE /SERVERS
   ```

2. Enable the SNMP service:

   ```
   SERVER-CONFIG>ENABLE SNMP
   ```

3. If desired, enable SNMP service on specific OpenVMScluster nodes, or restrict access to the service.
4. Save the modified service configuration.

   ```
   SERVER-CONFIG>SAVE
   [Writing configuration to IP$COMMON_ROOT:[IP]SERVICES.MASTER_SERVER]
   ```

5. Restart the IP$SERVER process:

   ```
   SERVER-CONFIG>RESTART
   ```

```
Exit the utility:
SERVER-CONFIG>EXIT
```

11.3. Private MIB Application Program Interface

In addition to SMUX and AgentX, VSI TCP/IP's SNMP agent supports subagents serving private MIBs through an application programming interface (API). Under this scheme, anyone willing to have their private MIBs served by VSI TCP/IP's SNMP agent should develop a shareable image that exports the APIs in them in addition to the routines they may need for accessing the MIB variables.

The SNMP API routines are described in *VSI TCP/IP Programmer's Reference.*

11.4. Configuring SNMP Subagents (except AgentX)

To configure an SNMP subagent on your host using the SERVER-CONFIG utility:

1. Start SERVER-CONFIG:

   ```
   $ IP CONFIGURE /SERVERS
   ```

2. Select the SNMP service:

   ```
   SERVER-CONFIG>SELECT SNMP
   ```

3. Set the subagent-image:

   ```
   SERVER-CONFIG>SET SUBAGENT-IMAGE
   ```

   You can now delete old entries or add new ones. Enter the name of one subagent per prompt, until finished. When finished, press **Return** at the prompt. *Do not enter the .EXE extension.*

4. Save the modified server configuration and exit.

   ```
   SEVER-CONFIG>SAVE
   [Writing configuration to IP$COMMON_ROOT:[IP]SERVICES.MASTER_SERVER]
   ```

5. Restart the IP$SERVER process:

   ```
   SERVER-CONFIG>RESTART
   ```

6. Exit the utility:

   ```
   SERVER-CONFIG>EXIT
   ```

11.5. SNMP Multiplexing Peers

The SNMP Multiplexing (SMUX) protocol is an SNMP subagent extension protocol. Each subagent or peer registers a MIB subtree with the SNMP Agent. Requests for objects residing in a registered MIB subtree are passed from the SNMP Agent using the SMUX protocol to the subagent. The subagent passes the results of an SNMP query back to the SNMP Agent. The practical limit to the number of peers is 30.

Enabling SMUX (**DEFINE /SYSTEM /EXEC IP$SNMP_SMUX 1**) when there are no SMUX subagents to use it can interfere with walking of the SNMP management base due to the SMUX MIB returning NoSuchName when no subagents exist. SMUX is an historical protocol, and should not be enabled unless there are subagents that will be using it. Specific items in the SNMP management base that appear after the SMUX MIB can still be queried when they are accessed from the start of their management base.

The SNMP server only accepts SMUX connections from peers listed by IP address in the `SNMPD.CONF` file. To enable SMUX support, issue the following command before starting SNMP:

```
$ DEFINE/SYSTEM/EXECUTIVE IP$SNMP_SMUX 1
```

11.5.1. SMUX_PEER *`IP-address`*

The SNMP agent listens on TCP port 199 for peer connections, while the connection to the SNMP client is over UDP port 161, with traps sent over UDP port 162. Multiple peers registering the same subtree are each assigned a priority, and the agent can send multiple variables in a single request. The SMUX protocol is described in RFC 1227.

11.6. SNMP Agent Extensibility (AgentX) Peers

The SNMP agent listens on TCP port 705 for subagent connections. The AgentX framework consists of a single processing entity called the master agent. This master agent, available on the standard transport address, sends and receives SNMP protocol messages in an agent role but has little or no direct access to management information. While some of the AgentX protocol messages appear similar in syntax and semantics to the SNMP, remember that AgentX is not SNMP. Refer to RFCs 2741 and 2742 for complete AgentX information. The SNMP server only accepts AgentX connections from peers listed in the `SNMPD.CONF` file. To enable AgentX support, issue the following command before starting SNMP:

```
$ DEFINE/SYSTEM/EXECUTIVE IP$SNMP_AGENTX 1
```

or

```
$ IP CONFIGURE/NET
NET-CONFIG>SET SNMP-AGENTX TRUE
```

11.6.1. Setting Up VSI TCP/IP to Use Insight Manager

Insight Manager Support has been added to VSI TCP/IP. The Insight Manager (CIM) uses the SNMP extensibility provided by Agent X to allow remote examination and notification of system conditions that may need attention. Remote management agents like CIM allow systems administration personnel to manage more systems while still meeting response time goals by providing access to critical information from a central location. The remote management agent communicates with the SNMP agent on the system being managed, which then sends the request to a program specifically designed to manage a particular component of the system.

Note

Make sure all files extracted from the TCP/IP Services kit have WORLD:RE protection.

1. Add the following to `IP$:SNMPD.CONF`:

   ```
   AGENTX_PEER 127.0.0.1
   community elmginkgo 127.0.0.1 read (and other community strings as
    needed)
   ```

2. Comment out `SMUX_PEER` from `IP$:SNMPD.CONF`

3. Restart SNMP with this command:

   ```
   $ IP NETCONTROL SNMP RELOAD
   ```

   or

   Start VSI TCP/IP

4. Start Insight Manager

5. Run **/PROCESS=HR_MIB SYS$SYSTEM:TCPIP$HR_MIB**

   The Host Resources MIB (RFC 1514) supplied with TCP/IP Services will now work with SNMP.

Note

The new ESNMP client interface in VSI TCP/IP services v10.5 uses Agent X to allow others to provide additional objects for SNMP to manage. Insight Management Agents are written to use the ESNMP client interface, hence the addition of Agent X protocol allows them to be used with VSI TCP/IP. By using the ESNMP library or Agent X directly, writers of TCP/IP services can allow the state of the service to be queried and controlled remotely. This can be useful if the service does not have a user interface, or runs under batch, or as a detached process.

11.7. Configuration File

The SNMP configuration file `SNMPD.CONF` is located in the `IP$CONFIG` directory. The SNMP file defines:

- Values for a subset of MIB management objects
- Clients and communities who can access the SNMP agent
- MIB access privileges for each client and community
- Authentication Failure, Link Up, and Link Down traps' status
- Originating addresses for traps
- SMUX peer details
- Agent X peer details

Note

After editing the configuration to fit your needs, stop and restart the SNMP agent so that the changes can take effect. If you do not edit the configuration file, the SNMP agent uses default values.

11.7.1. File Format

Follow these guidelines when entering data in the SNMP configuration file:

- Allow one line for each item.
- Enter information in any order, in upper- or lowercase.
- Enter variable string information (*`id-string`* and *`contact-name`*) in upper- or lowercase, depending on the operating system. Some SNMP clients on your network (such as those running UNIX) might require information in a specific case.
- Use a pound sign (#) or an exclamation point (!) to denote comments. SNMP ignores all information following these characters.
- Place quotation marks (" ") around strings that contain spaces or that require more than one line in the file, and around the comment characters when used as regular characters.

11.7.2. Values for MIB Objects

To define the values of several MIB objects in the SNMP configuration file, use the corresponding keywords listed in Table 11.1.

Table 11.1. MIB Objects

MIB object name...	Has keyword...
system.sysDescr	SYSDESCR
system.sysContact	SYSCONTACT
system.sysLocation	SYSLOCATION
if.ifTable.ifEntry.ifDescr and if.ifTable.ifEntry.ifSpeed	INTERFACE
system.sysServices	SYSSERVICES

The following paragraphs explain how to define each item.

`SYSDESCR` [*`id-string`*]

The *`id-string`* is the full name of the hardware, operating system, and networking software. For example:

```
SYSDESCR “HPE Integrity rx2800 i4 VSI TCP/IP for OpenVMS V8.4-2L1"
```

If you omit the *`id-string`*, VSI TCP/IP tries to obtain this information from your current system. If the attempt fails, the default is `System description is unknown`. Try again, entering a different *`id-string`*.

`SYSCONTACT` [*`contact-name`*]

The *`contact-name`* specifies the person to contact for the host, and how you can contact this person (such as by mailbox address). For example:

```
SYSCONTACT "John Smith, X 1234, smith@process.com"
```

The default is `System contact is unknown at this time`. Try again, entering a different *`contact-name`*.

`SYSLOCATION` [*`system-location`*]

The *`system-location`* specifies the geographical location of the host. For example:

```
SYSLOCATION "Main Street, Anytown, MA"
```

The default is: `System location is unknown at this time`. Try again, entering a different *`system-location`*.

`INTERFACE` [*`line-id line-speed description`*]

The *`line-id`* specifies the line identification for the IP layer network device. The *`line-speed`* specifies the line speed in bits per second. The *`description`* is the manufacturer's name, product name, and hardware version for the interface. For example:

```
INTERFACE SE0-1 10000000 "DELQA Ethernet Controller Version 1.0"
```

`SYSSERVICES` [*`services-set-number`*]

The services-set-number default is 72. See RFC 1213 for more information about *`services-set-number`* value calculation.

`HOSTID` ip-address

Specifies the IP address to use as the source address for traps sent either from the SNMP Agent or from the TRAP_GEN program. If this is not specified the address of the first interface (often SE0) is used. When this is specified it is checked against the addresses of the interfaces present on the system.

11.7.3. Community Parameters

The SNMP configuration file must contain the following information for each client permitted access to the SNMP agent:

`COMMUNITY` *`community-name internet-address type`*

`community-name`	Specifies the name of the community to which the client belongs.
`internet-address`	Specifies the client's internet address. If you enter 0.0.0.0, any address can use the community. The internet address can be optionally followed by /mask for READ and WRITE.
`type`	Defines the access profile as one of the following: • READ — The client can retrieve data from the MIB on this host. • WRITE — The client can retrieve data from and write data to the MIB on this host. • TRAPS — The client will receive all enabled traps.

The following example shows community parameters defined in the configuration file.

Example 11.1. Community Parameters

```
community northeast 192.168.4.56 READ
community northeast 192.168.220.1 WRITE
community southwest 192.168.23.1 WRITE
community southwest 192.168.23.1 TRAPS
```

- Client 192.168.4.56 in the northeast community has READ access to the MIB, while client 192.168.220.1 in the same community has WRITE access.
- Client 192.168.23.1 belongs to the southwest community. This community has WRITE access to the MIB and can receive all traps.

11.7.4. Template Configuration File

SNMP Services provides a `TEMPLATE_SNMPD.CONF` file in `IP$COMMON:[IP]` that you can use as a basis (see Example 11.2).

Example 11.2. Sample SNMP Configuration File

```
! SNMP Agent (SNMPD) Configuration File (template)
!
! System description: sysdescr <id string>
! Typically the id string would include:
! IA64 cpu model (such as HPE Integrity rx2800 i4)
! OpenVMS and version number
! "VSI TCP/IP for OpenVMS Version 10.5"
!
sysdescr "place system description string here"
!
! System Contact: syscontact contact name
!
syscontact "place name, phone number, and mail address of administrator
here"
!
! System Location: syslocation location
!
syslocation "place system location information here"

! Line Interfaces Information: interface line-id line speed
! description
! Note: You usually need not define these. SNMPD provides good defaults.
!
! line-id is one of LPB-, ETHER-, UNA-, QNA-, BNA-, SVA-, MNA-,ISA-,KFE,
! MXE-, ERA-, EWA-,CEC-, EIA-, CLIP-, ELA-,FDDI-,MFA-,FZA-, FAA-, FEA-,
! FQA-, FPA-, TR-, TRA-,TRE-, TRW-,PRO-, HYP-, DSV-, DSB-,DST-, X25-,SLIP,
! DECnet-, PPP-,PSD- followed by a unit number. Note that the unit number
! may be an encoding of the controller when the device is an ethernet
! adapter.
!
! (A = 0, B= 1, C=2, etc.)
!
! line-speed is an integer in bits per second of the data rate of the
! device
!
! description is a quoted string describing the device.
!
```

```
!interface una-0 10000000 "HEWLETT-PACKARD DELUA Ethernet controller"
!
! Communities:
! community community name internet address
! <READ | WRITE | TRAPS>
!
community readers  192.168.1.2  READ
community netman  192.168.2.3  WRITE
community nettraps 192.168.3.4  TRAPS
!
! To disable authentication traps, remove the "!" from the following
! line.
!no-auth-traps
!
! To disable link status traps, remove the "!" from the following
!line.
!no-link-traps
!
! SMUX Peers:
! AGENTX_PEER ip-address
! SMUX_PEER ip-address
!
AGENTX_PEER 192.168.6.7
SMUX_PEER 192.168.4.5
SMUX_PEER 192.168.5.6
```

AGENTX_PEER *ip-address* — The SNMP server only accepts AGENT X connections from peers listed by IP address in the `SNMPD.CONF` file. Use the following syntax in the file:

```
AGENTX_PEER ip-address
```

The COMMUNITY, SMUX_PEER, and AGENTX_PEER statements in the `SNMPD.CONF` file can take an optional mask after the internet address. The mask should be separated from the internet address with a / (slash). Valid values are from 0 to 32, with 32 being the default. Although the TRAPS community accepts a mask, it is not used currently.

For example:

```
community ournet 192.168.1.10 write  !implied /32
community ourmgr 192.168.1.0/24 read
```

The /24 specifies that only the first 24 bits must match. In this example, IP addresses from 192.168.1.0 to 192.168.1.255 can use the community ourmgr. The mask should be separated from the internet address with a / (slash). Valid values are from 0 to 32, with 32 being the default. A more restrictive IP address may be used within a less restrictive one. For example:

```
community process 192.168.6.0/24 READ
community process 192.168.6.42 WRITE
```

This allows all nodes in 192.168.6 to have READ access with the community name process and 192.168.6.42 to have WRITE access with the community name process.

The following command can be used to display enabled SNMP Agent X subagents in the output of the **SHOW** command:

```
$ IP CONFIGURE /NETWORK
$ SET SNMP-AGENTX TRUE
$ SET SNMP-AGENTX FALSE
```

TRUE enables SNMP Agent X service; FALSE disables SNMP Agent X service. A line displays in the output of the SHOW command if SNMP Agent X subagents are enabled.

11.8. Sending SNMP Traps from VSI TCP/IP

SNMP traps can be sent from VSI TCP/IP in the following manner:

Define the symbol:

```
TRAP_GEN :== $IP$:TRAP_GEN
```

Then type:

```
$ TRAP_GEN enterprise generic_trap specific_trap
[trap_specific_values....]
```

`enterprise` identifies the location in the MIB tree that this trap pertains to.

An example would be:

```
1.3.6.1.4.105.3,
```

which denotes a location in VSI's portion of the MIB tree.

`generic_trap` is an integer representing the generic trap value.

`specific_trap` is an integer representing the specific trap value.

`trap_specific_values` are arbitrary strings separated by spaces that are passed to the agent receiving the trap as octet strings.

The TRAP_GEN program uses the trap community definitions in the `IP$:SNMPD.CONF` file to determine where to send the trap.

Note

For VSI TCP/IP 10.5 there is also a program available that will listen for traps and format them for display. In order to invoke this program, run **IP$:TRAP_LISTEN**. It prompts for an optional file to log information to (default it the terminal) and the port number to listen on (default is 162).

11.9. Disabling Traps

All traps that the SNMP agent supports are initially enabled. You can disable traps by editing the configuration file. These changes take effect the next time you start the agent. The following table shows how to disable traps.

Table 11.2. Disabling Traps

Disable this trap...	By entering...
Authentication Failure	no-auth-traps
Link Up	no-link-traps
Link Down	no-link-traps

Note

SNMP clients can enable or disable the Authentication Failure trap while the SNMP agent is running. These clients must have WRITE community access.

11.10. Generating Traps

To generate an SNMP trap, define the symbol:

```
TRAP_GEN :== $IP$:TRAP_GEN
```

Then type:

```
$ TRAP_GEN ENTERPRISE GENERIC_TRAP SPECIFIC_TRAP [TRAP_SPECIFIC_VALUES....]
```

where:

- ***`enterprise`***: Identifies the location in the MIB tree that this trap pertains to. An example would be: 1.3.6.1.4.105.3, denoting a location in VSI's portion of the MIB tree.
- ***`generic_trap`***: It is an integer representing the generic trap value.
- ***`specific_trap`***: Is an integer representing the specific trap value.
- ***`trap_specific_values`***: Are arbitrary strings separated by spaces that are passed to the agent receiving the trap as octet strings.

The TRAP_GEN program uses the trap community definitions in the `IP$:SNMPD.CONF` file to determine where to send the trap.

To specify a particular IP-address for SNMP traps to originate from put the following line in `IP$:SNMPD.CONF`:

```
HOSTID IP_ADDRESS
```

The IP_ADDRESS specified is checked against those on the system when the line is parsed.

11.11. SNMP Log File

When the SNMP agent starts up, it creates a log file called `IP$:SNMPSERVER.LOG`. This file contains information about the activities of the SNMP agent, such as:

- The time the agent starts up and shuts down.
- When SMUX peers open or close a connection, and register or de-register a MIB tree.
- Any errors found in the SNMP configuration file.
- Any errors that occur when the agent is running.

11.12. Start, Shutdown, or Reload the SNMP Configuration Without Rebooting

To start, shutdown, or reload the SNMP configuration using NETCONTROL:

```
$ IP NETCONTROL SNMP START
$ IP NETCONTROL SNMP SHUTDOWN
$ IP NETCONTROL SNMP RELOAD
```

VSI TCP/IP has a separate SNMP_AGENT process. Once the SNMP service is enabled via the **IP CONFIGURE/SERVERS** command, the SNMP_AGENT process can be accessible via the **NETCONTROL** commands.

11.13. Performing SNMP Functions with VSI TCP/IP

The **IP SET** and **IP SHOW** commands accept the **/SNMP_HOST** qualifier for using remote host information.

Table 11.3 shows qualifiers you can use with the **/SNMP_HOST** qualifier in **IP SHOW** commands to obtain information from remote SNMP agents.

Table 11.3. IP SHOW /SNMP Commands

Qualifier	Description
`/COMMUNITY_NAME=”`*`string`*`”`	Specifies the community name string sent with this command to the remote host. The default is public. **Note** The case of the community name must match what is specified in `SNMPD.CONF`. The name must be specified in quotes unless it is all uppercase.
`/CONNECTIONS[=ALL]`	Displays network connections. If you use the **=ALL** argument, **IP SHOW** also displays sockets on which servers are listening.
`/ARP`	Displays the Ethernet Address Resolution Protocol tables. This qualifier (or **/ROUTE**) must precede all other qualifiers.
`/MIB_VAR[=`*`variable_name`*`]`	Displays the value of the SNMP MIB variable, *`variable_name`*. The value can be any MIB-II variable described in RFC-1213. If you omit *`variable_name`*, all MIB variables are displayed.
`/ROUTE`	Displays routing information for the IP protocol. This qualifier (or **/ARP**) must precede all other qualifiers.
`/STATISTICS[=`*`protocol`*`]`	Causes **IP SHOW** to display either network interface statistics or protocol statistics or both, as defined in MIB-II. If you specify **/STATISTICS** without a value, INTERFACE statistics is displayed.

Chapter 12. Configuring the VSI TCP/IP NFS Server

This chapter describes how to configure and maintain the VSI TCP/IP NFS Server, the VSI TCP/IP software that allows users of OpenVMS computers to export files to a variety of computers.

This chapter refers to the VSI TCP/IP NFS Server and NFS Client software as the *NFS Server* and *NFS Client*, and to the OpenVMS server host as the *server*, and to NFS client hosts as *clients*.

12.1. Understanding the VSI TCP/IP NFS Server

The VSI TCP/IP NFS Server is a high-performance OpenVMS implementation of Sun Microsystems' Network File System (NFS) protocol. It allows client computers running a variety of operating systems to remotely access files on an OpenVMS server. To users on a client system, all mounting and access operations are transparent, and the mounted directories and files appear to reside on the client.

After the VSI TCP/IP NFS Server is installed, the system manager configures the server and its clients to allow network file access. The VSI TCP/IP NFS Server includes configuration utilities for this purpose.

The VSI TCP/IP NFS Server is exceptionally fast due to parallel request processing, a file and directory cache between the file systems and the network, and an optional writeback cache feature. For example, because the VSI TCP/IP NFS Server can process many client requests simultaneously, a single client does not interfere with the requests of others.

12.2. Servers and Clients

A VSI TCP/IP NFS Server system is an OpenVMS system that makes its local files available to the network. A client is a host that accesses these files as if they were its own. Typical clients include:

- Sun Microsystems hosts or similar systems running the UNIX Operating System
- IBM PC computers and PC-compatible systems running the MS-DOS Operating System
- Hewlett-Packard computers running the ULTRIX (Tru64 UNIX) Operating System
- OpenVMS computers running the VSI TCP/IP NFS Client.

The OpenVMS server can make any of its file systems available to the network. A file system is a hierarchy of devices, directories, and/or files stored as a FILES-11 ODS-2 or ODS-5 on-line disk structure. File systems can include bound volumes and shadow sets.

The OpenVMS server *exports* the file systems, making them available to the network. Authorized clients *mount* the exported file systems, making them available to their users as if they were local directories and files.

Each file system is identified by the name of its mount point; that is, the name of the device or directory at the top of its hierarchy. When a client mounts a file system, it connects the mount point to a mount directory in its own file system. Through this connection, all files below the mount point on the server are available to client users as if they were below the client mount directory.

Each client automatically converts mounted directory and file structures, contents, and names to the format required by its own operating system. For example, an OpenVMS file named:

```
USERS:[JOE_NOBODY]LOGIN.COM
```

might appear to a UNIX end user as:

```
/vmsmachine/users/joe_nobody/login.com
```

and to an MS-DOS end user as:

```
E:\users\joe_nobody\login.com
```

Note

The VSI TCP/IP NFS Server can convert all valid UNIX, MS-DOS, or ULTRIX file names to valid OpenVMS names. Similarly, the server can convert those OpenVMS file names back to valid UNIX, MS-DOS, or ULTRIX names.

12.2.1. Security

The VSI TCP/IP NFS Server provides two levels of security:

Access to Individual...	Description
File systems	Can be restricted to specific clients listed in mount restriction lists for those file systems.
Directories and files	These are controlled on a per-user basis. The VSI TCP/IP NFS Server consults a database that maps users on NFS client systems to OpenVMS userids. When the NFS Server receives an NFS file access request, it maps the client user identifier in the request to an OpenVMS userid/UIC and compares the UIC to the owner, protection mask, and any directory or file ACLs. The NFS Server either grants or denies access.

The VSI TCP/IP NFS Server considers default privileges as defined by the user's UAF entry that override OpenVMS protection codes (SYSPRV, BYPASS, READALL, and GRPPRV) when granting access. However, since UNIX clients don't understand OpenVMS privileges, the client may prevent an operation which would otherwise have been allowed. If the UNIX user root (uid 0) is mapped to an OpenVMS user with BYPASS privilege, the user root can access all files.

To get GROUP protection access to a file from UNIX clients, a user must pass both the client and the server protection check. The client check is done using the UNIX GID; the server check is done using the Group portion of the OpenVMS UIC. For GROUP access to be granted, a user must be in the same UIC group on the OpenVMS system and have the same GID on the UNIX system.

Note

All NFS security relies on trusting the client to provide the server with truthful authentication parameters. Nothing in the NFS protocol specification prevents a client from using fraudulent parameters to bypass the security system.

VMS DELETE access does not directly translate to NFS. In NFS, a user with WRITE access to the directory can delete a file. The NFS Server implements DELETE access in the same way as NFS.

With this in mind, it is important for the system manager to review protection settings on exported file systems.

12.2.2. NFS Server Architecture

The VSI TCP/IP NFS Server includes five top-level protocols that run parallel to each other over a stack of lower-level protocols. The top-level protocols are:

- The Network File System protocol (NFS) is an IP-family protocol that provides remote file system access, handling client queries.
- The RPC (Remote Procedure Call) mount protocol (RPCMOUNT) is used by clients to mount file systems and get mount-point information.
- The RPC-protocol port mapper (RPCPORTMAP) performs the mapping between RPC program numbers and UDP and TCP port numbers.
- The RPC quota daemon (RPCQUOTAD) returns disk quota information.
- The RPC status monitor (RPCSTATUS) and the RPC lock manager (RPCLOCKMGR) together coordinate access to sections of files.

Underlying the NFS, RPCLOCKMGR, RPCMOUNT, RPCPORTMAP, RPCQUOTAD, and RPCSTATUS protocols is a stack of protocols:

- The Remote Procedure Call (RPC) protocol allows clients to make procedure calls across the network to the server.
- The external Data Representation (XDR) protocol handles architectural differences between the client and server systems, allowing the NFS protocol to communicate between systems of dissimilar architectures.
- The User Datagram Protocol (UDP), Transmission Control Protocol (TCP), and Internet Protocol (IP) are used for the lowest levels of communication.

Traditionally, NFS has always been run over UDP. The VSI TCP/IP NFS Server also supports communication over TCP, which may provide performance improvements when communicating with NFS clients across slow network links or wide area networks (WANs), which suffer from packet loss and delay.

12.3. NFS Server Configuration Overview

There are three main aspects to configuring the VSI TCP/IP NFS Server system:

1. Enabling the NFS Server on the server host.
2. Configuring the NFS server.
3. Configuring the clients.

These operations are performed normally in the following sequence:

1. Enable the NFS Server, using SERVER-CONFIG.
2. Make sure that each user who will access the server has an OpenVMS user account on the server and an account on the client.

3. Invoke NFS-CONFIG to perform Steps 4 through 6.
4. Provide the NFS Server with a basis for translating between the OpenVMS and client identifiers for each user.
5. Export each file system:
   a. Choose a name for the mount point.
   b. Export the mount point and reload the server to make the change effective.
   c. Mount and test the file system on each client.
   d. If you want to restrict access to the file system to specific clients, create a mount restriction list for the mount point, restart the server, and retest the mount operation from each client.
6. *Only when necessary*, change global parameter settings (followed by a server restart), and retest the configuration. The default parameter settings are sufficient in most cases.

The following sections describe these operations.

12.4. Enabling the VSI TCP/IP NFS Server

Enable the VSI TCP/IP NFS Server by enabling the following services:

- NFS server
- RPCMOUNT mount server
- RPCQUOTAD quota server
- RPCLOCKMGR lock manager
- RPCSTATUS status monitor
- RPCPORTMAP RPC-protocol port mapper

For networks that include IBM PC or PC-compatible clients with PC-NFS software, you should also enable the PC-NFSD server. Use the VSI TCP/IP Server Configuration Utility (SERVER-CONFIG) to enable these services.

The following sample session show how to enable protocols with SERVER-CONFIG:

```
$ IP CONFIGURE/SERVER
VSI TCP/IP Server Configuration Utility
[Reading in configuration from IP$:SERVICES.MASTER_SERVER]
SERVER-CONFIG>ENABLE NFS
SERVER-CONFIG>ENABLE RPCMOUNT
SERVER-CONFIG>ENABLE RPCQUOTAD
SERVER-CONFIG>ENABLE RPCPORTMAP
SERVER-CONFIG>ENABLE RPCLOCKMGR
SERVER-CONFIG>ENABLE RPCSTATUS
SERVER-CONFIG>RESTART
Configuration modified, do you want to save it first ? [YES] YES
[Writing configuration to SYS$COMMON:[IP]SERVICES.MASTER_SERVER]
%RUN-S-PROC_ID, identification of created process is 0000017A
SERVER-CONFIG>EXIT
```

12.5. Reloading the VSI TCP/IP NFS Server Configuration and Restarting the Server

Whenever you change the server configuration, you alter the NFS configuration data files (`NFS_EXPORT.DAT`, `NFS_GROUP.DAT`, `NFS_MNTLST.DAT`, and `NFS_PROXY.DAT`). Most of the remaining procedures described in this chapter change the configuration. Before you can use a new or revised configuration, you must reload the VSI TCP/IP NFS Server, either from within NFS-CONFIG or from DCL.

Reloading the server involves reloading the NFS and RPCMOUNT services:

- Enter the following command from DCL to restart only the VSI TCP/IP NFS Server:

  ```
  $ IP NETCONTROL NFS RESTART
  ```

- Enter the following command from DCL to restart only the RPCMOUNT protocol:

  ```
  $ IP NETCONTROL RPCMOUNT RELOAD
  ```

12.6. Shutting Down the NFS Server

You can edit your **SYSHUTDOWN.COM** procedure to include commands that stop the NFS Server. For example:

```
$ IP NETCONTROL NFS SHUTDOWN
```

12.7. Testing the System Configuration

Test the configuration at these times:

- After your initial configuration, when you have:
  - Specified the mappings between UIDs/GIDs and user names
  - Configured the VSI TCP/IP NFS Server
  - Restarted the NFS Server
  - Configured one or more clients for the NFS Server
- After you modify the configuration by reconfiguring the NFS Server, adding clients, or reconfiguring existing clients

To test a configuration, check all file systems from one client, and at least one file system from every client:

1. Log in as one of the client's users. For example, on a Sun Microsystems host client, you might log in as "joebob" (be sure your system includes a mapping for "joebob's" UID/GID and a user name on the server system).
2. Mount a file system the user can access.
3. Check the mount as described in the next steps.

a. Check the contents of the file system's mount directory. For example, on a Sun host client, use the cd command to change to the mount directory, and the ls -l command to list the names of the directory's files.

b. Verify that files in the mount directory can be read. For example, on a Sun host client, use the cp command to copy a file from directories under the mount point to the /dev/null directory.

c. Verify that files can be written to the OpenVMS server. For example, on a Sun host client, use the following command to copy a file to the current directory:

```
$ cp /vmunix .
```

Note

VSI recommends using the cp utility to test the server because it is better at reporting protection problems than most other UNIX utilities, including cat.

4. Repeat this process until you have mounted and checked all file systems that the client's users wish to access.

5. Log in from each of the other clients and check file system mounts as described in Steps 1 through 4.

12.7.1. Checking for Errors

After exporting file systems and restarting the server, *but before configuring clients*, enter the following command:

```
$ REPLY/ENABLE=NETWORK/TEMP
```

This command causes network event messages to be displayed on your terminal, including error messages from the NFS and RPCMOUNT servers. See the *VSI TCP/IP Messages, Logicals, and DECnet Applications* manual for lists of error messages and the conditions that generate them.

12.8. Idiosyncrasies of ACL Support over NFS

When using ACLs, OpenVMS lets the VSI TCP/IP NFS Server assign different access masks for many different groups of users. When a file's attributes are transmitted to the client, the NFS protocol only lets the server return an access mask for the owner's GID; the protocol does not allow the VSI TCP/IP NFS Server to return multiple GIDs and their associated access masks. Because some NFS clients grant or deny access based on the protections returned with the file's attributes, the VSI TCP/IP NFS Server's responses to attribute requests sometimes *change the owner's GID and associated access mask* to properly represent access for the client user.

One anomaly these dynamic responses produce is that a directory listing on the client (for example, an ls -l command on a UNIX client) shows files accessed through ACLs as being owned by different GIDs at different times, depending on who accessed them most recently.

If the client grants or denies access based on the protection information in the cache, users may experience intermittent access failures when more than one user tries to gain access to the same file via an ACL. This phenomenon happens when the user would normally receive access through the group or through an ACE (access control list entry).

While world access can always be consistently mapped, owner access is only consistently mapped if the ACL does not contain ACEs that cannot be mapped to a GID. For details, see the Section 12.8.2. If the UID/GID translation table is configured correctly, users should never have access to files to which they have no legitimate access on the server. However, they may intermittently be denied access.

12.8.1. How the VSI TCP/IP NFS Server Interprets ACL and UIC Protection

The main difficulty facing VSI TCP/IP NFS Server administrators is how to coordinate NFS use of the UNIX-style UID/GID protection model with OpenVMS ACL support.

Note

Consulting ACLs as part of an NFS Server's access-checking scheme is necessary, but not sufficient, to adequately support the presence of ACLs assigned to files.

Consider the case where an OpenVMS system manager wants to grant access to files based on project groups *without* having to make sure that all client UIDs map to the same OpenVMS group. A single user may be a member of several projects, a concept incompatible with the single-group model used by OpenVMS.

It might seem that the system manager only needs to add rights identifiers to the mapped accounts and then set up ACLs on the appropriate files. The problem with this scheme is that file protections would normally be set to WORLD=NOACCESS, allowing file accesses to be granted only by the ACL. However, because the file protections deny access on a UID basis, any local access checks performed at the client will fail, bypassing the ACLs.

This problem can be solved if the VSI TCP/IP NFS Server makes intelligent use of the NFS GID. The NFS protocol allows a single user to be identified with up to 10 groups (projects), consistent with UNIX. In this model, the NFS client checks the list of GIDs assigned to the local user to see if it matches the group ID associated with the file. If there is a match, the GROUP field of the file protection mask is used to determine accessibility.

The VSI TCP/IP NFS Server takes advantage of this model by selectively modifying the returned group ownership for files based upon applicable ACEs. The VSI TCP/IP NFS Server processes ACLs in the following manner. To determine whether the NFS Server will grant access:

1. The NFS Server obtains rights identifiers for the OpenVMS account associated with the requester's UID.
2. The NFS Server selects the first (if any) ACE assigned to the file (matching one of the rights identifiers held by the OpenVMS account). The protection specified in the ACE is used in place of the protection mask associated with the file.
3. If there are no matching ACEs, the VSI TCP/IP NFS Server performs the standard UIC protection check.

When asked by the NFS client for the protection mask and ownership for a file, the NFS Server does the following:

1. The NFS Server obtains rights identifiers for the OpenVMS account associated with the requester's UID.

2. If one of the ACE identifiers matches the file owner's UIC, the NFS server uses the protection mask in the ACE to calculate the OWNER field of the protection mask returned to the NFS client. Otherwise, the NFS Server uses the OWNER field of the protection mask associated with the file to calculate the OWNER field returned to the NFS client.
3. The NFS Server selects the first (if any) protection ACE assigned to the file (matching one of the rights identifiers held by the requester's OpenVMS account).
4. If the NFS server encounters a matching ACE whose identification is a UIC, and the identifier:
   - Is in the same OpenVMS group as the file owner, the server ignores the ACE.
   - Is not in the same OpenVMS group as the file owner, the server maps the requester's UIC and GID along with the along with the ACE's protection mask as the owner of the file when returning NFS attribute information
5. If the NFS server selects an ACE, the group ownership it returns to the NFS client is taken from the GID associated with the matching identifier.

If no matching ACE is found, the VSI TCP/IP NFS server obtains the GID for the file from the file owner.

To assign GIDs to UICs and identifiers, use the NFS-CONFIG ADD UID command.

Under this scheme, the system manager sets file protections as needed (for example, W:NOACCESS, G:RWE), and creates an ACL to grant access to processes holding a specific rights identifier.

When the NFS client performs local access checking, it compares the list of GIDs (associated with the user) against the file's group ownership, which the NFS Server bases on the ACL information for the file. This scheme prevents the client's caching mechanism from defeating the ACLs associated with the file.

12.8.2. How the VSI TCP/IP NFS Server Handles ACLs

The key to understanding how ACLs affect file access is in the exchange that takes place when an NFS client requests attributes for a file or directory it wants to access. The client sends the server the user's UID/GID pair when it identifies the file it wants to access. The server must respond with the UID/GID pair of the file's owner along with the protections on that file in UNIX format (R/W/E for owner, group, and world/others). To accomplish this, the NFS Server must translate the OpenVMS protection mask, applicable ACEs, and UIC-based file owner into a UNIX-style protection mask and UID/GID-based owner.

If the file being requested has no ACLs associated with it, the NFS Server simply returns the OpenVMS file owner's UID/GID pair which it obtains from the NFS Server's UID translation table and the file's owner, group, and world protections.

If the file has an ACL, the NFS Server scans the ACL for ACEs in a format that does not allow the NFS Server to map group protections. These ACLs must be handled in a special way (see the Section 12.8.3).

If there are no unmappable ACEs, the client's UID is translated, and the ACL is scanned for a match based on the associated UIC. At the same time, the list is also scanned for ACEs that should be mapped to world or owner protections. Based on the scan, the server returns attributes as follows:

- The OWNER protection mask returned is the owner default protection mask logically OR'd with the access mask of the first ACE matching the owner's UIC and associated rights identifiers. This emulates OpenVMS behavior and prevents the owner of the file from being denied access because of an ACE.

- The WORLD protection mask returned is the access mask associated with the first "wildcard" ACE, if one exists. Otherwise, the WORLD protection mask returned is the default WORLD protection.
- The GROUP protection mask returned is the access mask associated with the first ACE matching the requestor's UIC and associated rights identifiers. The GID returned is the GID translation of the rights identifier or UIC that matched this ACE. If no such ACE is found, the GROUP protection mask returned is the default group protection mask and the GID returned is the GID translation of the file's owner.
- The UID returned is the UID translation of the file's owner.

12.8.3. Handling ACLs with Unmappable ACEs

Occasionally, ACE access cannot be mapped to a GID as described in the previous section. This happens when the ACE identifier is specified in the following manner:

```
[*,member]
```

This also happens in cases of multiple identifiers on a single ACE, such as:

ACE	**Description**
A+*B*	*A* and *B* represent rights identifiers.
[*a*,*]+[*b*,*]	*a* and *b* represent UIC groups.
A+[*a*,*]	*A* is a rights identifier and *a* is a UIC group.

If the ACL associated with the file contains any ACEs that cannot be mapped to a GID, file attributes are returned as follows:

- The owner protection mask returned is the access mask associated with the first ACE matching the requestor's UIC and associated rights identifiers. If no such ACE is found, the owner protection mask returned is the default protection mask appropriate for the requestor; that is, the owner's default protection mask if they own the file, the group protection mask if appropriate, and so on.
- The owner UID/GID returned is the UID/GID translation of the requestor.
- The group protection mask returned is NONE.
- The world protection mask returned is NONE.

The VSI TCP/IP NFS Server cannot accurately represent OpenVMS protections in this case. This technique ensures no users are granted access to data to which they would not normally have access in the client's cache on the server. However, on multi-user clients where access is denied based on cached file attributes, this mapping may result in intermittent access failures to other users trying to access the file simultaneously.

Chapter 13. Configuring the VSI TCP/IP NFS Client & Server

13.1. Server Security & Initial Configuration

VSI TCP/IP supports version 3 of the NFS server and client protocols. The NFS server provides several features that maintain the integrity of the OpenVMS filesystem.

First, the server requires that the local system must register any user trying to access OpenVMS files. You do this through the PROXY database when you configure the Server and through later modifications as needed.

Second, you must export an OpenVMS directory for an NFS user to access it. The server does this through the EXPORT database when you configure the Server and through later modifications as needed.

13.2. Mounting Client Directories

NFS clients access OpenVMS files on the NFS server by mounting directories. The MOUNT protocol services the mount requests from clients attempting to mount an NFS export.

Mounting procedures vary by client and may require superuser privileges, or in the case of PC clients, a username and password. Some clients mount a remote directory automatically when they reboot the system (as in the case of fstab). Others mount a remote directory dynamically when they reference the remote file (as with an automount).

Mount procedures require the following information:

- The pathname of the exported directory that matches the pathname in the EXPORT database
- The name of the host running the server that contains the files you want mounted
- A pathname on the client designated as the mount point

Below is a mount command provided by a Solaris 9 Client to the NFS_SERVER running on host IRIS, using the defined export of NFS0:[USER.MNT] . The export is mounted onto the local /mnt partition.

```
# mount IRIS:DKA0:\[USERS.MNT\] /mnt
```

In the example, IRIS is the name of the VSI TCP/IP host. DKA0:[USERS.MNT] is the exported directory. /mnt is the mount point on the Solaris client host.

Check your NFS client documentation before mounting directories.

13.3. File Formats

The NFS protocol does not define standard file and record formats or a way of representing different types, such as text or data files. Each operating system can have a unique file structure and record format.

The Server provides access to all OpenVMS files. However, even though an NFS client can access a file, the client may not be able to correctly interpret the contents of a file because of the differences in record formats.

The UNIX operating system stores a file as a stream of bytes and uses a line feed (LF) character to mark the end of a text file line. PC systems also store a file as a stream of bytes, but use a carriage-return/line-feed (CRLF) character sequence to mark the end of a text file line. PC systems sometimes also use a **Ctrl/Z** character to mark the end of a file.

The OpenVMS operating system, with its Record Management Services (RMS), provides many file organizations and record formats. RMS supports sequential, relative, and indexed file organizations. It also supports FIXED, STREAM, STREAM_CR, STREAM_LF, UNDEFINED, VARIABLE, and variable with fixed size control area (VFC) files.

NFS clients most commonly need to share text files. STREAM is the RMS record format that most closely matches PC text files. STREAM_LF is the RMS record format that most closely matches UNIX text files.

In OpenVMS, you can store standard text files in VARIABLE, STREAM_LF, or VFC record format. Most OpenVMS utilities can process these text files regardless of the record format because the utilities access them through RMS.

The intent of the Server is to provide convenient access to the majority of OpenVMS files. Because many OpenVMS text files are VARIABLE or VFC format, the Server converts these files to STREAM or STREAM_LF format as it reads them.

13.3.1. Reading Files

The Server reads all files (except VARIABLE and VFC) block by block without interpreting or converting them. It reads VARIABLE and VFC files by converting them to STREAM or STREAM_LF, based on a selected option. The file on the NFS server remains unchanged.

The Server's automatic file conversion process can cause a slow reading of VARIABLE and VFC files. For example, in returning the file size, it reads the entire file. Full directory listings can also be slow if the directory contains a number of VARIABLE or VFC record format files. If you need frequent access to these files, consider converting them using the OpenVMS CONVERT utilities.

13.3.2. Writing Files

By default, the Server creates STREAM_LF files, but can also create STREAM files on demand. It writes all files except VARIABLE and VFC block by block without interpreting or converting them. If an NFS client tries to write to or change the size of an existing file not having STREAM, STREAM_LF, STREAM_CR, FIXED, or UNDEFINED format, the Server returns an EINVAL error.

13.4. Troubleshooting

If you are experiencing network communication-related problems on the NFS server, please check the following items:

1. Make sure VSI TCP/IP is running on the OpenVMS system.
2. Confirm the RPC service is running with the following command at the DCL prompt:

   ```
   $ IP SHOW /RPC
   ```

3. Make sure the server is running. If not, start it by entering the following command at the DCL prompt:

   ```
   $ IP NETCONTROL NFS RESTART
   ```

4. To verify general connectivity between the two systems, try using FTP or TELNET. For example, try to open a TELNET connection with the remote host in question. If another product is not available on your system, try using the PING utility

5. Verify the internet addresses the local host and the remote hosts are using. If your local network includes a gateway, also verify the gateway address.

6. If you are experiencing problems performing NFS operations from a NFS client, check the Server's `NFS_SERVER.LOG` file. It may contain messages that can help isolate the problem.

13.5. Managing an Existing NFS Configuration

VSI TCP/IP includes a configuration utility IP CONFIGURE /NFS to manage NFS client and server configurations.

```
$ IP CONFIGURE /NFS
NFS-CONFIG>
```

The configuration utility program has the following commands to manage the configuration:

ADD EXPORT

ADD EXPORT — *NFS server only*. Adds an entry to the EXPORT database that lets the NFS server export the server filesystems to a remote NFS client. Users at the NFS-Client can then mount the server filesystems. Requires write access to the `IP$CONFIG:NFS_EXPORT.DAT` file. The EXPORT database is dynamic. Entries you add to the database become valid immediately. You do not need to restart the server.

Format

ADD EXPORT *nfs-path vms-directory*

Parameters

nfs-path

NFS-style pathname used to reference the exported directory. Typically expressed as a UNIX-style pathname. Enclose in quotation marks (" ").

Although nfs-path can be arbitrary, it usually reflects the actual OpenVMS directory path. The NFS client user must refer to the same nfs-path in naming the mount point.

vms-directory

Directory on the local OpenVMS server that you want to export. The directory must include the device specification, as in the following example:

```
$DISK1:[SALES.RECORDS]
```

When you export a directory, the NFS client user can potentially have access to all files and directories below the export point. The device you export should be a "public" device. The server does not implement volume protection. Also, the server supports Files-11 ODS-2 and ODS-5 structure level disks.

Qualifiers

Note

Many of the following qualifiers are specific to applications running on certain hosts. In these cases, it is critical to use the /HOST qualifier in combination with these qualifiers.

/HOST=(host [,host...])

Only specified host(s) can have access to the exported OpenVMS directory specified as host names or internet addresses. Use the parentheses only if you specify a list of hosts (separated by commas). If you omit /HOST, any host can mount the exported directory.

/CONVERT={STREAM_LF (default) | STREAM_CRLF}
/NOCONVERT

/CONVERT converts files on reads to either `STREAM_LF` (the default) for UNIX systems or `STREAM_CRLF` for Windows systems.

/NOCONVERT disables this conversion and must be specified when using the server together with the VSI TCP/IP NFS Client. For use with VSI TCP/IP's NFS Client.

/EXPLICIT_MOUNT
/NOEXPLICIT_MOUNT (default)

/EXPLICIT_MOUNT prevents users from subsequently mounting subdirectories of the mount point. /NOEXPLICIT_MOUNT allows subdirectory mounts.

/FILENAME={ SRI | ODS5 | PATHWORKS | PATHWORKS_CASE }

Uses the `SRI` International, `ODS5`, or `PATHWORKS` filename mapping schemes. SRI is the default scheme between UNIX and OpenVMS systems.ODS5 uses minimal mapping to get around ODS-5 file naming restrictions. If the disk or system doesn't support ODS-5, it falls through to SRI. PATHWORKS specifies non-case-sensitive filename mapping. `PATHWORKS_CASE` specifies case-sensitive filename mapping.

/HIGHEST_VERSION
/NOHIGHEST_VERSION (default)

/HIGHEST_VERSION returns only the highest version of files in directory requests. / NOHIGHEST_VERSION does not. All file versions still exist in either case.

/NOPRIVILEGED_PORT (default)

/PRIVILEGED_PORT requests that incoming requests originate from privileged ports only. / NOPRIVILEGED_PORT does not.

/PROXY_CHECK
/NOPROXY_CHECK (default)

/PROXY_CHECK specifies that mount requests only originate from users having mappings in the PROXY database. /NOPROXY_CHECK does not

/RFM=*option*

Record format (RFM) of newly created files. The options are STREAMLF, STREAMCR, STREAM, FIXED, and UNDEFINED.

/SERVER_ACCESS
/NOSERVER_ACCESS (default)

/SERVER_ACCESS requests the server to do access checking. /NOSERVER_ACCESS requests that both the server and client do the checking.

/SUPERUSER_MOUNT
/NOSUPERUSER_MOUNT (default)

/SUPERUSER_MOUNT requests that only the superuser can mount a file system. /NOSUPERUSER_MOUNT does not.

/VERSION={ DOT | SEMICOLON (default) | ALL | HIGHEST }

DOT changes the file version display for exported filesystems to file.ext.version (a dot) for UNIX compatibility instead of the usual file.extension;version (a semicolon).

SEMICOLON (default) uses the regular semicolon.

ALL exports files with version numbers intact rather than the default of leaving the highest numbered version unnumbered.

HIGHEST is a synonym for /HIGHEST_VERSION. Do not use DOT with SEMICOLON.

/WRITE (default)
/NOWRITE

/WRITE requests that the client have read-write access to the filesystem. /NOWRITE requests that the client have read access only.

Example

Exports the directory SALES.RECORDS on device $DISK1: as path /vax/records to hosts ORCHID and ROSE. Any subdirectories below SALES.RECORDS are also accessible. However, hosts ORCHID and ROSE cannot have access to or mount directories above SALES.RECORDS or other SALES subdirectories.

```
$ ADD EXPORT "/vax/records" $DISK1:[SALES.RECORDS] /HOST=(ORCHID,ROSE)
```

ADD GROUP

ADD GROUP — *NFS client only*. Adds an entry to the GROUP database that associates an OpenVMS user with an NFS group or list of groups. Requires SYSPRV privilege and write access to the `IP$CONFIG:GROUP.DAT` file. If the GROUP database does not exist, use the CREATE GROUP command first to create an empty one. Use the REMOVE GROUP command to remove a group from the database. The GROUP database is static. Use the RELOAD command when you modify it.

Format

```
ADD GROUP nfs-group vms-identifier
```

Parameters

nfs-group

NFS group number found in the /etc/group file on the server. For example, if the users group appears in the /etc/group file as:

```
users:x:15:
```

Use 15 as the *nfs-group*.

vms-identifier

Associates either an OpenVMS rights identifier or UIC (or wildcarded UIC) with the NFS group. Only associate one *vms-identifier* per NFS group. Use either of the following formats to enter the value:

Format	Description
Name	OpenVMS rights identifier or username
Value	UIC value in [*group*,*member*] or %X*nnnnnnnn* format; you can use wildcard entries such as [200,*].

"Name" and "value" correspond to the columns associated with entries in the OpenVMS rights database. To have access to this database, use the commands:

```
$ SET DEFAULT SYS$SYSTEM
$ RUN AUTHORIZE
UAF>SHOW/IDENTIFIER *
```

For example, the following line may appear in the rights database:

```
Name    Value            Attributes
-----   -----            ----------
USER    [000200,000200]
```

Qualifier

/HOST=(host[,*host*...])

Server host(s) on which the group identification is valid. If omitted, any remote host is valid for the group. /HOST accepts either host names or internet addresses. Use the parentheses with multiple host entries.

Examples

1. Associates NFS group number 15 on server host IRIS with the "value" [200,*], meaning "any user in group 200."

```
$ ADD GROUP /HOST=IRIS
_Group: 15
_Identifier: [200,*]
```

The *nfs-group* number derives from the entry in the /etc/group file on the server for the users group:

```
>cat /etc/group
staff:*:10:
```

```
users:*:15:
```

2. Associates NFS group number 15 with the OpenVMS rights identifier, USERS. As in Example 1, the *`nfs-group`* number derives from the entry in the /etc/group file on the server. Assuming that the USERS rights identifier exists in the rights database, any user granted this identifier would be in the group corresponding to GID 15 in NFS.

   ```
   $ ADD GROUP 15 USERS
   ```

   The resulting ADD GROUP entry would appear in the GROUP database as follows:

   ```
   NFS GROUP Database
   Group   Name    Value       Host(s)
   -----   ----    -----       ------
   15      USERS   %X8001000C
   ```

ADD PROXY

ADD PROXY — *NFS client and server*. Registers an NFS or remote user as an OpenVMS username in the PROXY database. Requires SYSPRV file and write access to the `IP$CONFIG:NFS_PROXY.DAT` file. If you omit the /CLIENT or /SERVER qualifier, or do not define the `IP$NFS_DYNAMIC_PROXY` logical accordingly, you must use the RELOAD PROXY command to reload the database. (For details, see the RELOAD in this chapter.)

Format

`ADD PROXY` *`vms-username`*

Parameter

`vms-username (required)`

OpenVMS username to which you want to map an NFS user ID. The username must appear as in the OpenVMS User Access File (`SYSUAF.DAT`).

Qualifiers

The /HOST, /UID, /GID, or /NFS qualifiers make the PROXY entry more restrictive. When you omit a qualifier, NFS-OpenVMS interprets it as a wildcard. For example, the command ADD PROXY SMITH/UID=210 creates an entry that lets a user with UID=210, but with any GID and from any host, use OpenVMS username SMITH.

/HOST=(host[,*`host`*...])

Host(s) from which the UID/GID identification is valid. Specify at least one host name. If omitted, MANAGE_NFS3 allows any remote host with the matching identification.

/HOST accepts either host names or internet addresses. Use parentheses for multiple hosts.

/UID=*`uid`*

User's ID (UID). If omitted, MANAGE_NFS3 accepts any UID for the *`vms-username`*.

/GID=*`gid`*

User's group ID (GID). If omitted, MANAGE_NFS3 accepts any GID for the *`vms-username`*.

/CLIENT
/NOCLIENT (default)

> /CLIENT notifies the Client to immediately update its loaded PROXY database with an entry for *`vms username`*. /NOCLIENT does not notify the client. This overrides any default action specified using the logical `IP$_NFS_DYNAMIC_PROXY`.

/SERVER
/NOSERVER (default)

> /SERVER notifies the server to immediately update its loaded PROXY database with an entry for *`vms-username`*. /NOSERVER does not notify the Server. This overrides any default action specified using the `IP$NFS_DYNAMIC_PROXY` logical.

Examples

The following examples range from most restrictive to least restrictive:

1. Registers a user with UID=210 and GID=5 at host ROSE to OpenVMS username SMITH for the NFS server only.

   ```
   $ ADD PROXY SMITH /UID=210 /GID=5 /HOST=ROSE /SERVER
   ```

2. Registers a user with UID=210 and GID=5 to OpenVMS username SMITH and dynamically reloads the PROXY database on both the client and server.

   ```
   $ ADD PROXY SMITH /UID=210 /GID=5 /CLIENT /SERVER
   ```

3. Registers any user with GID=5, any UID, and at any host to OpenVMS username JONES.

   ```
   $ ADD PROXY JONES /GID=5
   ```

4. Registers any user from host ORCHID to OpenVMS username JONES.

   ```
   $ ADD PROXY JONES /HOST=ORCHID
   ```

CREATE EXPORT

CREATE EXPORT — *NFS server only*. Creates an empty EXPORT database. Requires write access to the `IP$CONFIG:NFS_EXPORT.DAT` file. NFS server installations create an empty EXPORT database. Use this command to supersede an existing EXPORT database only.

Format

```
CREATE EXPORT
```

Example

Shows the current EXPORT database, overwrites it, and shows that the database is now empty.

```
SHOW EXPORT
NFS EXPORT Database
Path   Directory                Host(s)
----   ---------                -------
/usr   $DISK1:[SALES.RECORDS]   SIGMA
CREATE EXPORT
SHOW EXPORT
```

```
no EXPORT entries found
```

CREATE GROUP

CREATE GROUP — *NFS client only.* Creates an empty GROUP database. Requires write access to the `IP$CONFIG:NFS_GROUP.DAT` file. Client installation creates an empty GROUP database. Only use this command to supersede an existing GROUP database.

Format

```
CREATE GROUP
```

Example

Shows the current GROUP database, overwrites it, and shows that the database is now empty.

```
SHOW GROUP
NFS GROUP Database
Group   Name       Value        Host(s)
-----   ----       -----        -------
15      GROUP      %X8001000B
15      GROUP_16   %X8001000E
CREATE GROUP
SHOW GROUP
   No entries in GROUP database
```

CREATE PROXY

CREATE PROXY — *NFS client and server*. Creates an empty **PROXY** database. Requires write access to the `IP$CONFIG:NFS_PROXY.DAT` file. Client and server installation creates an empty **PROXY** database. Only use this command to supersede an existing **PROXY** database.

Format

```
CREATE PROXY
```

Example

Shows the current **PROXY** database, overwrites it, and shows that the database is now empty.

```
SHOW PROXY
NFS PROXY Database
Username   UID    GID   Host(s)
--------   ---    ---   -------
BART       1116   15
MARGE      1115   15
LISA       1117   16
HOMER      -2     -2
CREATE PROXY
SHOW PROXY
no PROXY entries found
```

EXIT

EXIT — This command exits the NFS_CONFIG Utility.

Format

```
EXIT
```

FIND PROXY

FIND PROXY — *NFS client and server*. Locates and displays a single entry in the PROXY database. Requires read access to the `IP$CONFIG:NFS_PROXY.DAT` file. On the client, use this command to find the UIC assigned a specific user. On the server, use this command to determine which OpenVMS username the server uses when it receives a request from the specified UID, GID, and host name.

Format

```
FIND PROXY
```

Qualifiers

Note

You must specify all threeof the following qualifiers.

/HOST=*`host-name`*

Host on which the user is valid. This qualifier is required.

/UID=*`uid`*

User's ID (UID). This qualifier is required.

/GID=*`gid`*

User's group ID (GID). This qualifier is required.

Example

Locates an OpenVMS username for an NFS user with UID=210, GID=5, at host ROSE.

```
FIND PROXY /UID=210 /GID=5 /HOST=ROSE
NFS PROXY Database
Username   UID   GID   Host(s)
--------   ---   ---   -------
SMITH      210   15    ROSE
```

RELOAD EXPORT

RELOAD EXPORT — *NFS server only*. Implements changes made to the EXPORT database without having to restart the client system. Requires SYSLCK privilege. The **EXPORT** database is normally static. The **RELOAD** command puts the changes into effect. Use this command sparingly. The client can take a significant amount of time to reload the database. The reloading process blocks NFS activity.

Format

```
RELOAD EXPORT
```

Parameter

`nfs-path` (optional)

NFS-style pathname used to reference the exported directory. Typically expressed as a UNIX-style pathname. Enclose in quotation marks (" ").

RELOAD GROUP

RELOAD GROUP — *NFS client only*. Implements changes made to the GROUP database without having to restart the client system. Requires SYSLCK privilege. The GROUP database is normally static. The **RELOAD** command puts the changes into effect. Use this command sparingly. The client can take a significant amount of time to reload the database. The reloading process blocks NFS activity.

Format

```
RELOAD GROUP
```

Parameter

`nfs-path` (optional)

NFS-style pathname used to reference the exported directory. Typically expressed as a UNIX-style pathname. Enclose in quotation marks (" ").

RELOAD PROXY

RELOAD PROXY — *NFS client and server*. Implements changes made to the **PROXY** database without having to restart the client or server. Not necessary if the `IP$NFS_DYNAMIC_PROXY` logical was defined as CLIENT or SERVER or the combination of the two. Requires SYSLCK privilege. When both CLIENT and SERVER are specified the logical should be defined as if it is a search list, not a single value. The **PROXY** database is normally static. The **RELOAD PROXY** command puts the changes into effect. Use this command sparingly. The client can take a significant amount of time to reload the database. The reloading process blocks NFS activity.

Format

`RELOAD PROXY` [*vms-username*] [*vms-username*]

Parameter

vms-username

Reloads only the PROXY database entries for the specified username (or list of usernames separated by commas). This is useful for notifying the client or server of changes to the OpenVMS `SYSUAF.DAT` file, such as changes to the rights list or user privileges.

Qualifiers

Note

If you omit both qualifiers, the **PROXY** database reloads on both the client and server.

/CLIENT
/NOCLIENT

/CLIENT reloads the PROXY database on the client only. /NOCLIENT does not reload the database on the client.

/SERVER
/NOSERVER

/SERVER reloads the PROXY database on the server only. /NOSERVER does not reload the database on the server.

REMOVE EXPORT

REMOVE EXPORT — *NFS server only*. Removes an entry from the **EXPORT** database so that you can remove access to an exported directory for a single host or a list of hosts. Requires write access to the `IP$CONFIG:NFS_EXPORT.DAT` file. The **EXPORT** database is dynamic. Any path that you remove from the database becomes invalid immediately. You do not need to restart the server.

Format

```
REMOVE EXPORT "nfs-path"
```

Parameter

"*nfs-path*"

NFS-style pathname used to reference the exported directory. Typically expressed as a UNIX-style pathname. You must enclose the *nfs-path* in quotation marks (" ").

Qualifier

/HOST=(host[,*host*...])

The following example removes access to an *nfs-path* for a single host or a list of hosts. If omitted, MANAGE_NFS3 removes *nfs-path* for all hosts.

Example

Removes a record from the **EXPORT** database so that NFS host ORCHID can no longer mount an OpenVMS directory on the /vax/records pathname.

```
REMOVE EXPORT "/vax/records" /HOST=ORCHID
```

REMOVE GROUP

REMOVE GROUP — *NFS client only*. Removes a group mapping from the GROUP database. Requires write access to the `IP$CONFIG:NFS_GROUP.DAT` file. The GROUP database is static. The **RELOAD** command puts changes into effect.

Format

```
REMOVE GROUP nfs-group [vms-identifier,...]
```

Parameters

`nfs-group`

NFS group number. If you specify *`nfs-group`* alone, MANAGE_NFS3 removes the entire group from the database.

`vms-identifier`

OpenVMS rights identifier(s) or UIC(s) associated with the NFS group. If you specify one, MANAGE_NFS3 removes only that identifier from the database; MANAGE_NFS3 does not change the remaining entries for that group. See the ADD command for the valid format of *`vms-identifier`* entries.

Qualifier

/HOST=(server[*`server,...`*])

Server host(s) on which the group number is valid. Either host names or internet addresses are valid. This qualifier removes the GROUP entry for the specified host(s) only. Use the parentheses with multiple *`server`* specifications.

Example

Removes a record from the GROUP database so that you can no longer associate group number 15 with a group account on the client.

```
REMOVE GROUP 15
```

REMOVE PROXY

REMOVE PROXY — *NFS client and server*. Removes an entry from the PROXY database. Requires SYSPRV privilege and write access to the `IP$CONFIG:NFS_PROXY.DAT` file. If you omit the /CLIENT or /SERVER qualifier, or do not define the `IP$NFS_DYNAMIC_PROXY` logical accordingly, you must use the RELOAD PROXY command to reload the database. (For details, see the RELOAD command in this chapter.)

Format

`REMOVE PROXY` *`vms-username`*

Parameter

`vms-username`

OpenVMS account you want to remove from the PROXY database. You can use the wildcard * in place of *`vms-username`* as long as you use one of the qualifiers below to be more selective about the update.

Qualifiers

If you omit a /HOST, /GID, or /UID qualifier, the command removes all entries containing the *`vms-username`* account from the database.

/HOST=(server[,*server*...])

Server host(s) on which the user is valid. MANAGE_NFS3 removes the PROXY entry for the specified host(s) only. Use the parentheses with multiple *server* specifications.

/GID=*gid*

User's group ID (GID). MANAGE_NFS3 removes the PROXY entry for the specified GID only.

/UID=*uid*

User's ID (UID). MANAGE_NFS3 removes the PROXY entry for the specified UID only.

/CLIENT
/NOCLIENT (default)

/CLIENT notifies the client to immediately update its loaded PROXY database with an entry for *vms-username*. /NOCLIENT does not notify the client. This overrides any default action specified using the `IP$NFS_DYNAMIC_PROXY` logical.

/SERVER
/NOSERVER (default)

/SERVER notifies the server to immediately update its loaded PROXY database with an entry for *vms-username*. /NOSERVER does not notify the server. This overrides any default action specified using the `IP$NFS_DYNAMIC_PROXY` logical.

Examples

1. Removes authorization for an NFS user at host MARIGOLD with UID=210 and GID=5 to use the OpenVMS username SMITH.

   ```
   REMOVE PROXY SMITH /UID=210 /GID=5 /HOST=MARIGOLD
   ```

2. Removes authorization for all users at host CROCUS to use OpenVMS username JONES.

   ```
   REMOVE PROXY JONES /HOST=CROCUS
   ```

3. Removes authorization for any user at host MARIGOLD to use any OpenVMS username.

   ```
   REMOVE PROXY * /HOST=MARIGOLD
   ```

4. Removes all entries containing the OpenVMS username SMITH.

   ```
   REMOVE PROXY SMITH
   ```

5. Removes authorization for a user with UID=210 and GID=5 to use the OpenVMS username SMITH and dynamically reloads the PROXY database on both the client and server.

   ```
   REMOVE PROXY SMITH /UID=210 /GID=5 /CLIENT /SERVER
   ```

SHOW EXPORT

SHOW EXPORT — Server host(s) from which exports are shown.

Format

`SHOW EXPORT` [*(host*,[host...])]

Parameter

`host`

Name of host. If omitted, local exports are shown. Host accepts either host names or internet addresses. Use the parentheses with multiple host entries.

Qualifiers

/FULL

Displays all information about the exports.

/OUTPUT=*`filespec`*

Uses the specified file instead of the terminal for output.

/PATH=*`path-name`*

Displays information about an export with a particular path name.

/UDP

Shows exports using UDP. Can be used when specifying the host name.

Example

Shows the NFS group number on host IRIS and corresponding OpenVMS group name and value.

```
SHOW GROUP /HOST=IRIS
NFS GROUP Database
Group   Name   Value     Host(s)
-----   ----   -----     -------
15      USER   [200,*]   IRIS
```

SHOW GROUP

SHOW GROUP — *NFS client only*. Displays entries in the client's GROUP database. Requires read access to the `IP$CONFIG:NFS_GROUP.DAT` file.

Format

`SHOW GROUP` [*`nfs-group`*]

Parameter

`nfs-group`

NFS group number for which to show database entries. If omitted, MANAGE_NFS3 displays entries for all groups on the local client.

Qualifiers

/HOST=(*`server`*[,*`server`*...])

Server host(s) on which the group number is valid. MANAGE_NFS3 accepts either host names or internet addresses. Use the parentheses with multiple server specifications.

/OUTPUT=*filespec*

Uses the specified file instead of the terminal for output.

Example

Shows the NFS group number on host IRIS and corresponding OpenVMS group name and value.

```
SHOW GROUP /HOST=IRIS
NFS GROUP Database
Group   Name   Value     Host(s)
-----   ----   -----     -------
15      USER   [200,*]   IRIS
```

SHOW MOUNT

SHOW MOUNT — *NFS client and server*. Displays a list of client hosts that mounted a file system served by a specified NFS server. Returns the mounted directories by the pathnames MANAGE_NFS3 uses to export them, not the directory names as the OpenVMS system knows them.

Format

SHOW MOUNT [*server-host*]

Parameter

server-host

NFS server host from which to get the list of mounted file systems. If omitted, MANAGE_NFS3 uses the local server.

Qualifier

/OUTPUT=*filespec*

Uses the specified file instead of the terminal for output.

Examples

1. Because the user did not specify the server host name, the system displays the full domain name for the local server ZETA. In this example no client hosts have mounted any of the server file system.

   ```
   SHOW MOUNT
   NFS Mount List
   Server: ZETA.example.com
   Path         Host
   ----         ----
   ```

2. Displays the list of client hosts and directories by pathnames for mounted file systems served by the specified server IRIS.

   ```
   SHOW MOUNT IRIS
   NFS Mount List
   Server: IRIS
   Path              Host
   ----              ----
   ```

```
   /sales/records    bart.example.com
   /exported/spool   bart.example.com
```

SHOW PROXY

SHOW PROXY — *NFS client and server*. Displays the contents of the PROXY database. Requires read access to the `IP$CONFIG:NFS_PROXY.DAT` file.

Format

`SHOW PROXY` [vms-username]

Parameter

`vms-username`

OpenVMS account entries you want to display. If omitted, the system displays the contents of the PROXY database determined by the qualifiers listed below.

Qualifiers

/HOST=(*`server`*[,*`server`*...])

Displays the PROXY entries restricted to the specified server host(s) only, or for which there are no host restrictions given. Specify one or more server hosts (if multiple, separate by a comma and use the parentheses).

/GID=*`gid`*

NFS user's group ID (GID). MANAGE_NFS3 displays only entries containing the specified GID.

/UID=*`uid`*

NFS user's ID (UID). The system displays only entries containing the specified UID.

/OUTPUT=*`filespec`*

Uses the specified file instead of the terminal for output.

Example

Displays the PROXY database entries for user SMITH.

```
SHOW PROXY SMITH
NFS PROXY Database
Username   UID   GID   Host(s)
--------   ---   ---   -------
SMITH      100   101
```

SHOW STATISTICS

SHOW STATISTICS — *NFS server only*. Displays statistics information on the NFS server, useful in troubleshooting if problems occur. The server must be running.

Format

`SHOW STATISTICS`

Qualifiers

/RESET

Displays the counter information, then resets the counters. Requires OPER privilege.

/TIMES

Displays the additional average and maximum times (in milliseconds) for certain NFS requests listed.

/OUTPUT=`filespec`

Uses the specified file instead of the terminal for output.

Description

The NFS statistics returned by the command are:

Started	Date and time someone started the server.
Uptime	Total amount of time the server has been running.
Memory in use	Total amount of dynamic memory (in bytes) the NFS server uses. This includes memory allocated for the RPC server routines.
Threads	NFS thread counters give the *total* threads available, the *current* number of threads in use, and the *maximum* number of threads that have been in use at one time. These statistics can give an indication of server load. If the *maximum* number of threads in use at one time is equal to the *total* threads available, you may want to increase the number of threads defined by the parameter NFS_THREADS.
Files	File system counters include the number of *opens* and *closes* performed by the server, the number of files *currently open*, and the *maximum* open files at one time since someone started the server. The number of files *currently open* and the *maximum open* files at one time can be an indication of the load on the server.
NFS	NFS counters return the total NFS procedure calls, and the total calls for each NFS procedure since you started the server. These counters can give an indication of the load on the server. total is the total number of calls bad call is the number of bad calls fail is the number of failed calls null is the number of null calls getattr is the number of get attribute calls setattr is the number of set attribute calls read is the number of reads lookup is the number of lookups

	mkdir is the number of make directory calls write is the number of writes create is the number of creates remove is the number of removes rename is the number of renames rmdir is the number of directory removes readdir is the number of address reads statfs is the number of file system statistics calls link is the number of create link to file calls symlink is the number of create symbolic link calls readlink is the number of read from symbolic link calls other is the number of other calls
RPC	RPC counters provide information on RPC operations. This includes the total number of *receives, transmits, XID hits*, and *duplicate receives*. The *XID hits* counter gives the number of cached replies the NFS Server retransmitted. The *duplicate receives* counter gives the number of times the server received a duplicate request for an operation that was in progress at the time of the request. If either of these counters is excessive you may need to increase the timeout time on the NFS-Client host(s).
RPC Errors	RPC counters also returns the following error conditions: *receive* and *transmit errors, authentication errors, decode errors,* and *RPC program errors*.
MOUNT	MOUNT counters return the total MOUNT procedure calls, the calls for each MOUNT procedure since someone started the server, the total number of directory mounts since someone started the server, and the number of directories currently mounted. total is the number of MOUNT calls bad call is the number of bad MOUNT calls fail is the number of failed MOUNT calls mount is the number of successful mounts unmount is the number of successful dismounts null is the number of null mounts dump is the number of dumps from MOUNT calls mnt export is the number of exported mounts cur mount is the number of current mounts

Example

The command description section describes the output parameters for this example. The /TIME qualifier includes the average and maximum times for the indicated NFS requests.

```
SHOW STATISTICS /TIME
NFS Server Statistics
Started:  1-FEB-2014 07:24:05 Uptime: 14 07:05:53  Memory in use: 1414850
Threads:   total         40 current     0 max         11
Files:     opens         54 closes     54 cur. open    0 max.open     5
NFS:       total       2519 bad call    0 fail         0
  null     6 getattr    149 setattr     6 read       396 lookup    1381
ave:       0 ms           7 ms         82 ms          78 ms          20 ms
max:       0 ms          40 ms        100 ms         180 ms          50 ms
  mkdir    0 write      396 create      6 remove      12 rename      18
ave:       0 ms          38 ms         83 ms          38 ms         117 ms
max:       0 ms         510 ms         90 ms         120 ms         130 ms
  rmdir    0 readdir     51 statfs      1 link         0 symlink      0
ave:       0 ms          32 ms         10 ms           0 ms           0 ms
max:       0 ms         230 ms         10 ms           0 ms           0 ms
  readlink 0 other        0 adfread    97 adfwrite     6
ave:       0 ms           0 ms          7 ms          27 ms
max:       0 ms           0 ms         50 ms          30 ms
RPC:         recv      2520 xmit     2520 xid hits     0 dup recv     0
RPC errors:  recv         0 xmit        0
  authweak 0 authother    0 decode      0 noproc       0 noprog       0
  progvers 2 systemerr    0
MOUNT:       total        1 bad call    0 fail         0
mount      1 unmount      0 null        0 dump         0 mnt export   0
mounts     1 cur. mount   1
```

UNMOUNT ALL

UNMOUNT ALL — *NFS client only*. Removes all the mount list entries for the local client host on the specified NFS server or servers. Useful for notifying the remote server host that the server file systems are no longer mounted on the client in the event that the client system goes down and you need to reboot it. Unmounting is not the same as dismounting. UNMOUNT ALL does not dismount a mounted file system. After using UNMOUNT, you can use SHOW MOUNT (in VSI TCP/IP) or show mount (on a UNIX system server) to verify that the list entry you requested to be unmounted on the specified server(s) is no longer there. The mount list entries are in the /etc/rmtab file on most UNIX systems.

Format

```
UNMOUNT ALL
```

Qualifier

/HOST=(*server*,*server*...)

Server host or hosts. The parentheses are required for multiple servers. If omitted, the client sends a broadcast message to all local network servers to remove the list entry for the local client host.

Examples

1. Sends a broadcast message to all local network servers to remove the mount list entry for the local client host.

   ```
   UNMOUNT ALL
   ```

2. Sends a request to hosts TAU and SIGMA to remove the mount list entry for the local client host.

   ```
   UNMOUNT ALL /HOST=(TAU,SIGMA)
   ```

Note

The following message can occur after an UNMOUNT ALL request sent to a UNIX system server:

```
RPC Client call failed, RPC: Remote system error
```

Ignore this message. However, confirm through a SHOW MOUNT command that the mount list entry was, in fact, removed.

13.6. Mounting an NFS file system on VSI TCP/IP

NFSMOUNT

NFSMOUNT — Mounts an NFS file system on VSI TCP/IP.

Format

`NFSMOUNT` *`server`* `"`*`nfs-path`*`"` `[`*`mountpoint`* `[`*`logical`*`]]`

Parameters

`server`

Name of the remote server, in domain name or IP address format.

`"`*`nfs-path`*`"`

Pathname (enclosed in quotation marks) on the remote server. The pathname must match an exported directory, subdirectory, or file of an exported filesystem on the server. (You can use the SHOW EXPORT command in the MANAGE_NFS3 utility to obtain a list of the exported directories.)

`mountpoint`

NFS device (and, optionally, directory tree) specification for the local mount point. If specified, this parameter must be in the format:

`NFS`*`n`*`:[[`*`dir.dir...`*`.]][`*`filename`*`]`

The value *`n`* can range from 1 to 9999, and *`dir`* is a directory level (up to eight in addition to the [000000] directory). If you omit the *`mountpoint`* specification or specify NFS0:, the client creates an NFS*`n`*:[000000] mount point, and increases *`n`* by one for each subsequent mount.

logical

Optional logical name associated with the volume. The client defines the logical as follows:

If you mount NFS*n*:[000000]	NFS*n*:
If you mount NFS*n*:[*dir.dir*]	NFS*n*:[*dir.dir*.]

The extra dot after the last *dir* in the second definition allows for relative directory specifications. If you perform the following function:

SET DEFAULT *logical*:[*subdir*]

the full default definition becomes:

NFS*n*:[*dir.dir.subdir*]

The client places the logical name in the SYSTEM logical name table unless you specify the /GROUP or /SHARE qualifier. The client deletes the logical name from the SYSTEM table when you dismount the volume. The process must have SYSNAM privilege to mount a system mount point. Without SYSNAM or GRPNAM privilege, the user must specify /SHARE for a JOB mount.

Qualifiers

/ACP_PARAMS= ([BUFFER_LIMIT= *limit-value*] [,DUMP] [,IO_DIRECT=*value*] [,IO_BUFFERED=*value*] [,MAX_WORKSET=*pages*] [,PAGE_FILE=*filespec*] [,PRIORITY=*base-priority*] [,WORKSET=*pages*])

Includes SYSGEN ACP and detached process parameters the system manager can set or modify. The SYSGEN parameters that affect ACPs are dynamic. The client applies the ACP parameters only at the initial start of an ACP and ignores them in subsequent mount requests when the client uses the same ACP.

/ADF=*option*
/NOADF

Controls whether you want to use attributes data files (ADFs). These files appear on a non-VMS server as .$ADF$*filename* files and the server uses them to store OpenVMS file attributes. You cannot directly view these files on the client system. The possible ADF *option* values are:

CREATE (the default and forced if /SERVER_TYPE=VMS_SERVER)	If ADFs exist on the server, the client will use, update, and create them for new files.
UPDATE	If ADFs exist on the server, the client will use and update them, but not create them for new files.
USE	If ADFs exist on the server, the client will use them, but not update them nor create them for new files.

Avoid using UPDATE and USE. The client may create ADFs anyway in certain cases, such as when renaming files. Also, changing OpenVMS attributes for a hard-linked file may result in inconsistent OpenVMS attributes between the linked files.

/AUTOMOUNT [= (INACTIVITY = *inactive-time*)]

Mounts a server filesystem automatically and transparently when you obtain the pathname. INACTIVITY specifies a maximum inactive period for the mount attempt. When the client reaches this period, it unmounts the pathname. Specify the time in delta (see the *VSI TCP/IP*

Administrator's Guide: Volume I for the information on network time). The default is five minutes (:5). Seconds are rounded to the nearest minute.

/BACKGROUND [=(DELAY=`delay-time`, RETRY=`retries`)]

Attempts to mount the filesystem at least once in background mode. If the first mount attempt fails, it informs you and keeps retrying after an optionally specified time delay and number of retries. If omitted, the DELAY defaults to 30 seconds (::30 in delta time). The maximum delay period you can specify is approximately 49 days. The default RETRY time value is 10. If you specify RETRY=0, the client uses 1 instead.

/CACHE_TIMEOUT [= ([DIRECTORY= `t`] [,ATTRIBUTE=`t`] [,READ_DIRECTORY])]

Caching timeout information for the mount point. The following keywords apply:

The DIRECTORY timer	Specifies the amount of time (`t`) the client waits between rereading a directory's status or contents. Specify the time in delta format (see the *VSI TCP/IP Administrator's Guide: Volume I* for the information on network time). The default is 30 seconds (::30 in delta time).
The ATTRIBUTE timer	Specifies the amount of delta time (`t`) the client waits between rereading a file's attributes from the server. The default is 15 seconds (::15 in delta time)
The READ_DIRECTORY keyword	Forces the client to read the contents of the directory requested when the cache timeout occurs, rather than relying on the directory's modified time. By reading the directory contents, the client can be aware of any changes to the number of files within the directory even if the directory's modify time was not updated.

/CONVERT={ STREAM_LF (default) | STREAM_CRLF } /NOCONVERT (forced for VSI TCP/IP's NFS Server)

Controls whether the Client should convert sequential, variable-length, carriage return carriage control (VAR-CR) files to STREAM-LF files for UNIX system servers or STREAM_CRLF for PC system servers. Some OpenVMS applications require that certain files remain VAR-CR. The default is /CONVERT=STREAM_LF unless you use /SERVER_TYPE=VMS_SERVER, in which case VSI TCP/IP forces a /NOCONVERT.

You can only convert files opened using RMS sequential access to STREAM-LF or STREAM_CRLF format when written by the client.

The NFS Client does not perform conversions when "block mode transfers" are performed. COPY and EDT use block mode transfers when copying or creating files. Instead of COPY, use the CONVERT command. Instead of EDT, use the TPU command. Most applications do RMS sequential access when they create files on the export and these will be converted.

/CONFIG=`filsespec`

Mounts one or more remote NFS directories based on information in a configuration file. In this way, you can maintain a regular list of server filesystems that you can automatically mount using one command.

`filsespec`

OpenVMS file containing the configuration information. The contents of the file should include line entries in the format prescribed by the NFSMOUNT command:

Note

The client uses qualifiers specified with the NFSMOUNT /CONFIG command as defaults for mount requests in the configuration file. However, qualifiers included with mount requests in the file override these defaults.

The configuration file must have complete information for a mount on each line (continuation lines are not allowed). The client ignores blank or comment lines. Mount requests in the file can have further configuration file references, although there is limited nesting of these requests.

Examples

1. The following command consults the `CONFIG_NFS.TXT` file for mounting information.

   ```
   $ NFSMOUNT /CONFIG=CONFIG_NFS.TXT
   ```

2. The following command also sets data size and username parameters (which can be overridden by qualifiers in the configuration file).

   ```
   $ NFSMOUNT /CONFIG=CONFIG_NFS.TXT /DATA=512 /USER=BART
   ```

/DATA=[([*read-bytes*],[*write-bytes*])]

> Largest amount of NFS data received (*read-bytes*) or transmitted (*write-bytes*) in a single network operation. The default for both is 8192 bytes, the maximum allowable value appropriate for most servers. The minimum is 512. If you specify only one value, that value applies to both *read* and *write*. However, you can use different values for each.
>
> You do not normally need to use the /DATA qualifier unless a remote server imposes a restriction on data size. Also, if the NFS server requests a smaller transfer size than the one set with this qualifier, the server's requested value will override the one set by /DATA.

/FILEIDS={UNIQUE | NONUNIQUE}

> With UNIQUE (the default), the client uses filenames and 32-bit NFS file IDs when processing the directory information returned by the server, to determine whether cached information is valid.
>
> With NONUNIQUE, the client uses file handles instead of file IDs in retrieving directory information. This can refresh directory entries in the client's cache more quickly, resulting in fewer "no such file" errors. However, this can degrade performance since the client must issue additional RPC requests. /FILEIDS=NONUNIQUE automatically implies a /LOOKUPS, so do not use it together with an explicit /NOLOOKUPS.

/FORCE
/NOFORCE (default)

> Controls whether or not to force an overmount or a mount that can cause filesystem occlusion. This qualifier requires OPER privilege. Overmounting a /SYSTEM mount requires SYSNAM privilege. Overmounting a /GROUP mount requires GRPNAM privilege.

/GID=*gid*

> Default GID if no GID mapping exists for file access. The default value is -2. Requires OPER privileges.

/GROUP

Places the logical name in the group logical name table. If the mount is the first group or system mount on the volume, /GROUP marks the volume as group-mounted and increments the mount count. Requires GRPNAM privilege. Do not use with /SYSTEM.

/LABEL=*volume-label*

ODS-2 volume label used for the remote pathname. You can use this qualifier to provide a unique volume label on a system where there is a conflict. The default is the first 12 characters of the combined *server:mountpoint* parameter. The client accepts only the first 12 characters for all other entries. The client applies the /LABEL qualifier on the first mount of an NFS device only and ignores it with subsequent mounts on that device.

/LOCK
/NOLOCK (default)

Specifies whether the client should use advisory network file locking by way of the Network Lock Manager (NLM) to coordinate access to server files.

/NOLOOKUPS (default)
/LOOKUPS

With /NOLOOKUPS (the default), the client does not look up file handles when building directory caches. However, when accessing an individual file, it does look up its file handle; and with a directory operation, it still looks up the handle for every file in the directory. Do not use an explicit /NOLOOKUPS together with /FILEIDS=NONUNIQUE.

/NOREADDIRPLUS

For NFS this disables the use of the READDIRPLUS command to read directory and file information. The client will fall back to using READDIR if it detects that the server does not support READDIRPLUS, so this is only necessary if there is a problem when using READDIRPLUS. Note that READDIRPLUS is generally more efficient than READDIR.

/OWNER_UIC=*uic*

Specifies the UIC assigned ownership of the volume while you mount it, thereby overriding the ownership recorded on the volume. The client applies the /OWNER_UIC qualifier on the first mount of an NFS device only and ignores it with subsequent mounts on that device.

/PROCESSOR={UNIQUE | SAME:*nfs-device* | FILE:*filespec*}

Requests that NFSMOUNT associate an Ancillary Control Process (ACP) to process the volume, which overrides the default manner in which the client associates ACPs with NFS devices. The qualifier requires OPER privilege. The possible keyword values are:

UNIQUE	Creates a new ACP (additional address space) for the new NFS device. This is useful for mounting large remote filesystems so that you can accommodate more cached information.
SAME:*nfs-device*	Uses the same ACP as the specified device. The *nfs-device* specified cannot be mounted as UNIQUE. Care should be taken when using this as NFS and NFS mount points cannot share an ACP.

FILE:*`filespec`*	Creates a new ACP running the image specified by a particular file. You cannot use wildcards, node names, and directory names in the *`filespec`*. Requires CMKRNL or OPER privilege.

/PROTECTION=*`protection-code`*

Protection code assigned the volume, following the standard syntax rules for specifying protection. If you omit a protection category, the client denies that category of user access. The default is (S:RWED,O:RWED,G:RWED,W:RWED).

The client applies the /PROTECTION qualifier on the first mount of an NFS device only and ignores it with subsequent mounts on that device. /PROTECTION requires OPER privilege.

/RETRIES=*`max-retries`*

Maximum number of times the client retransmits an RPC request. The default is zero (0), where the client retries the request indefinitely.

/SERVER_TYPE=*`server-type`*

Type of server from which the client mounts data. The valid values for *server-type* are:

- UNIX
- VMS_SERVER
- IBM_VM

The default is either UNIX or VMS_SERVER (if the server runs VSI TCP/IP's server).

With /SERVER_TYPE=VMS_SERVER, VSI TCP/IP forces /NOCONVERT and /ADF=CREATE regardless of their specified settings.

/SHARE

Places the logical name in the job logical name table and increments the volume mount count regardless of the number of job mounts. When the job logs out, all job mounts are dismounted, causing the volume mount count to be decremented.

/SHOW

Displays the mounted directories at all mount points or at a particular mount point.

```
$ NFSMOUNT /SHOW [mountpoint | device:]
```

`mountpoint`

Full NFS device name and directory tree for which to show mount information. For example:

```
NFS1:[USER.NOTES]
```

device:

NFS device name part of the *mountpoint* parameter (such as NFS1:). Alternately, you can use a logical name for the mount point. With the /ALL qualifier, the client uses only the device portion of the logical name.

Qualifiers

/ALL

Shows mount information for all servers, or a specified server or NFS device.

/FULL

Displays the full, current operating parameters related to each mount.

See the NFSMOUNT command for descriptions of the qualifiers that correspond to each of the operating parameters.

/QUOTA

Displays quota information for the current user's mount. The qualifier used by itself shows four columns at the top of the display indicating the block usage, soft limit (quota), hard limit, and grace period.

Use /QUOTA with the /FULL qualifier to show four additional columns indicating any possible file quotas. These show as zeros for an OpenVMS system but as actual values for UNIX systems that support file quotas.

Use /QUOTA with the /USER qualifier to request quotas for other than the default user.

/USER=*username*

Use with /QUOTA to show quotas for a specific user. This requires the mount to have been performed using the /SUPERVISOR qualifier, which maps users with SYSPRV, BYPASS, or READALL privileges to the superuser UID. /USER requires SYSPRV or GRPPRV privileges.

Examples

1. This example provides the default command display.

   ```
   $ NFSMOUNT /SHOW
   _NFS1:[000000] automount (inactivity timer 0 00:23:00.00), mounted
   SIGMA.EXAMPLE.COM:/usr
   _NFS2:[000000] mounted
   IRIS.EXAMPLE.COM:/usr/users
   ```

2. This example shows characteristics of all mounts on a specific NFS device.

   ```
   $ NFSMOUNT /SHOW NFS0: /ALL
   _NFS1:[A.B] mounted
   SIGMA.EXAMPLE.COM:/usr
   _NFS2:[A.C] mounted
   SIGMA.EXAMPLE.COM:/work
   ```

3. This example shows the full mount display with all operating parameters for a specific NFS device. Note that you can either enable or disable Writing and Write conversion.

```
$ NFSMOUNT /SHOW NFS1: /FULL
_NFS1:[000000] mounted
MERAK.EXAMPLE.COM:/eng/nfsuser
Transport                  UDP  Writing                Enabled
Read/write size      8192/8192  Write conversion       Disabled
RPC timeout      0 00:00:01.00  ADF usage
 USE,UPDATE,CREATE
RPC retry limit              0  Fileids                Unique,
 Nolookups
Attribute time   0 00:00:15.00  Server type            TCPware, NFS
Directory time   0 00:00:30.00  Advisory Locking       Disabled
Cache Validation   MODIFY TIME  Default user           [USER]
Superuser                   No  Default UID,GID        100,15
```

4. This example shows the additional full block and file quotas for the user's mount.

```
$ NFSMOUNT /SHOW NFS2: /QUOTA /FULL
_NFS2:[000000] mounted
viola:/pctest
Disk Quotas for user [SMITH]: (inactive)
Blocks   Quota   Limit   Grace      Files   Quota   Limit   Grace
117355   500000  600000             0       0          0
Transport                 UDP       Writing               Enabled
Read/write size     8192/8192       Write conversion     Disabled
RPC timeout     0 00:00:01.00       ADF usage   USE,UPDATE,CREATE
RPC retry limit             0       Fileids     Unique, Nolookups
Attribute time  0 00:00:15.00       Server type   VSI TCP/IP, NFS
Directory time  0 00:00:30.00       Advisory Locking     Disabled
Cache Validation  MODIFY TIME       Default user           [USER]
Superuser                  No       Default UID,GID        100,15
```

/SUPERUSER=`uid`
/NOSUPERUSER

Controls whether the client maps users with SYSPRV, BYPASS, or READALL privileges to the superuser UID. The server must allow superuser access. The normal superuser UID is 0.

/SYSTEM

Places the logical name in the system logical name table (the default action). If the mount is the first group or system mount on the volume, this marks the volume as system mounted and increments the volume mount count. Requires SYSNAM privilege. Do no use with /GROUP.

/TIMEOUT=*`timeout-period`*

Minimum timeout period (in OpenVMS delta time) for initial RPC request retransmissions. The default is ::1 (one second).

The *`timeout-period`* value should reflect the estimated typical round trip time for RPC requests. For slower speed links (like NFS traffic over SLIP or WANs), a larger value than the default would be appropriate.

For example, for a maximum read/write size of 8192 (see the /DATA qualifier) over a 19,200-baud SLIP line, the absolute minimum timeout value should be:

$$\frac{10240 \text{ bytes } * 8 \text{ bits per byte}}{19200 \text{ bits per second}} = 4.27 \text{ seconds}$$

The 10240 bytes are 8192 data bytes plus the worst case RPC overhead of 1048 bytes. Since 4.27 seconds is the absolute minimum, a more realistic value for this link would be in the range of 15 to 30 seconds to allow for other traffic.

/TRANSPORT=*`protocol-type`*

Network protocol used to transfer the data. The valid values are TCP and UDP (the default).

/UID=*`uid`*

Default UID, if no UID mapping exists for file access. The default value is -2. Requires OPER privileges.

/USER=*`username`*

Existing OpenVMS account to which the client maps unknown UIDs. The default is the USER account. If the Client does not find the USER account, the DECNET account becomes the default. If the client does not find the DECNET account, [200,200] becomes the default.

/VERSION (default)
/NOVERSION

Use the /NOVERSION qualifier to enforce a limit of one version on a file. This is a way of imposing an NFS file versioning scheme on OpenVMS files. /VERSION, allowing multiple versions, is the default. This qualifier is disabled if connected to a VSI TCP/IP NFS server.

/WRITE (default)
/NOWRITE

Allows that you mount the filesystem either with write access (/WRITE) or read-only (/NOWRITE) on the local machine. If /NOWRITE, file creation, deletion, and other modifications are not allowed.

Examples

1. In this example, the client mounts the /usr filesystem from sigma onto the OpenVMS mount point when it references the pathname. The client keeps the path mounted until the client reaches an inactive period of 10 minutes, after which it unmounts the pathname. Subsequent references cause the client to remount the filesystem.

   ```
   $ NFSMOUNT SIGMA "/usr" NFS0: /AUTOMOUNT=(INACTIVITY=00:10:00)
   ```

2. This example shows an overmount. The second mount specifies a lower level in the server path.

   ```
   $ NFSMOUNT SIGMA "/usr" NFS1:[USERS.MNT]
   %NFSMOUNT-S-MOUNTED, /usr mounted on _NFS1:[USERS.MNT]
   $ NFSMOUNT SIGMA "/usr/users" NFS1:[USERS.MNT] /FORCE
   %NFSMOUNT-S-REMOUNTED, _NFS1:[USERS.MNT] remounted as /usr/users on
    SIGMA
   ```

3. This example shows an occluded mount. The mount point specification is "backed up" one subdirectory on the second mount. Both mounts are visible in an NFSMOUNT/SHOW. However,

if you do a directory listing on NFS2:[USERS.SMITH], the [MNT] directory is no longer visible. To make the directory visible again, dismount NFS2:[USERS.SMITH].

```
$ NFSMOUNT SIGMA "/usr" NFS2:[USERS.SMITH.MNT]
%NFSMOUNT-S-MOUNTED, /usr mounted on _NFS2:[USERS.SMITH.MNT]
$ NFSMOUNT SIGMA "/usr" NFS2:[USERS.SMITH] /FORCE
%NFSMOUNT-S-MOUNTED, /usr mounted on _NFS2:[USERS.SMITH]
-IP-I-OCCLUDED, previous contents of _NFS2:[USERS.SMITH] occluded
```

13.7. Implementation

This section describes the Server restrictions and implementation of the Network File System (NFS) protocol. The material presented here requires a thorough understanding of the protocols. It does not explain or describe the protocols.

13.7.1. Restrictions

The Server has the following OpenVMS-related restrictions:

- The Server supports Files-11 ODS-2 structure level disks, ODS-5 formatted disks, and any CD-ROM format.
- The Server does not implement volume protection. All exported devices should be public devices.
- The Server does not generate security or audit alarms. However, the Server writes access violations to log file `IP$LOG:NFS_SERVER.LOG`.
- When creating files and directories, the Server sets the owner UIC of the file or directory to the UIC derived from the UID/GID in the create request authentication information or to the UID/GID in the set attributes information (if available).

13.7.2. NFS Protocol Procedures

The Server implements the following NFS protocol (version 3) procedures (while continuing to support version 2):

Procedures	**Description**
ACCESS (access)	The server determines the access rights that a user, as identified by the credentials in the request, has with respect to a file system object.
COMMIT CACHED WRITE DATA (commit)	The server forces data to stable storage that was previously written with an asynchronous write call
CREATE FILE (create)	The server creates files using the record format specified in the EXPORT database entry. The client may specify one of 3 methods to create the file: UNCHECKED: File is created without checking for the existence of a duplicate file. GUARDED: Checks for the presence of a duplicate and fails the request if a duplicate exists. EXCLUSIVE: Follows exclusive creation semantics, using a verifier to ensure exclusive creation of the target.
GET ATTRIBUTES (getattr)	Gets a file's attributes. The Server handles certain file attributes in ways that are compatible with the OpenVMS system. These attributes are: File protection—The server maps the OpenVMS file protection mask to the UNIX file protection mask.

Procedures	Description
	Number of links—Although OpenVMS supports hard links, it does not maintain a reference count. Therefore, the server sets this value to 1 for regular files and 2 for directory files. UID/GID—The server maps a file owner's UIC to a UID/GID pair through the PROXY database. Device number—The server returns the device number as -1. Bytes used—The total number of bytes used by the file. Filesystem id—The server returns the filesystem ID as 0. Access, modify, status change times—The OpenVMS system does not maintain the same file times as NFS requires. The server returns the OpenVMS revision (modify) time for all three NFS times. For directory files, the Server returns the access, status change, and modify times as a reasonably recent time, based on the time of the last Serverinitiated directory change, and the NFS_DIRTIME_TIMER parameter. This is a benefit to clients that cache directory entries based on the directory times. OpenVMS bases its time on local time, while UNIX bases its time on Universal time (or Greenwich mean time), and these times may not agree. The offset from Universal time specified when configuring VSI TCP/IP resolves the difference between local and Universal time.
GET DYNAMIC FILESYSTEM INFO (fsstat)	The server provides volatile information about a filesystem, including: • total size and free space (in bytes) • total number of files and free slots • estimate of time between file system modifications
GET STATIC FILESYSTEM INFO (fsinfo)	The server provides nonvolatile information about a filesystem, including: • preferred and maximum read transfer sizes • preferred and maximum write transfer sizes • flags for support of hard links and symbolic links • preferred transfer size of readdir replies • server time granularity • whether or not times can be set in a setattr request
LINK (link)	Creates a hard link to a file. The Server stores the link count in an application access control entry (ACE) on the file.
LOOKUP FILE (lookup)	Looks up a file name. If the file name does not have a file extension, the server first searches for a directory with the specified name. If the

Procedures	Description
	server fails to locate a directory, it searches for the file name without an extension.
MAKE DIRECTORY (mkdir)	Creates a directory. The OpenVMS system does not allow the remote host to create more than eight directory levels from the root of the OpenVMS filesystem. The server ignores access and modifies times in the request.
READ DIRECTORY (readdir)	Reads a directory. The Server returns file names using the filename mapping scheme as specified in the EXPORT database entry. The Server also drops the OpenVMS version number from the file name for the highest version of the file.
READ DIRECTORY PLUS ATTRIBUTES (readdirplus)	In addition to file names, the server returns file handles and attributes in an extended directory list.
READ FROM FILE (read)	Reads from a file. The Server converts VARIABLE and VFC files to STREAM or STREAM_LF format (depending on the option set) as it reads them. The server returns EOF when detected.
REMOVE DIRECTORY (rmdir)	Deletes a directory.
REMOVE FILE (remove)	Deletes a file.
RENAME FILE (rename)	Renames a file. If the destination filename is the same as an existing filename and the destination filename does not have a zero or negative version number, the Server overwrites the existing file.
READ LINK (readlink)	Reads the contents of a symbolic link.
SET ATTRIBUTES (setattr)	Sets file attributes. The Server handles certain file attributes in ways that are compatible with the OpenVMS system. These attributes are: File protection—The Server maps the UNIX file protection mask to the OpenVMS file protection mask, as shown earlier in this chapter. UID/GID—The client changes the file owner's UIC. The PROXY database maps the new UID/GID to an OpenVMS UIC. If the Server cannot locate the new UID/GID in the database, it returns an error and does not change the owner UIC. Size—If the file size is larger than the allocated size, the Server extends the file. If the size is 0, the Server truncates the file and sets the record attributes to sequential STREAM_LF. You cannot change the size of variable length or VFC files (except to zero). Access time—Changing the access time has no effect on the OpenVMS system. Modify time—The modify time updates the OpenVMS revision time.
SYMBOLIC LINK (symlink)	Creates a symbolic link. The server creates the file with an undefined record structure and uses an application ACE on the file to mask it as a symbolic link.
WRITE TO FILE (write)	Writes to a file. The Server does not allow a remote host to write to a directory file, or to VARIABLE and VFC files.

Procedures	Description
	If the server allowed a remote host to write to an existing OpenVMS file that was not a STREAM_LF or fixed-length record format file, the file could become corrupted. The server does not allow a remote host to explicitly change the record format of an OpenVMS file. The server can return the non-standard NFS error ETXTBSY (26) and EINVAL (22). The Server returns ETXTBSY when an OpenVMS user has a file open for exclusive access and an NFS user tries to use the file in a way that is inconsistent with the way the OpenVMS user opened the file. The server returns EINVAL if an NFS user tries to write to or change the size of a VARIABLE or VFC record format file. With version 3, the server supports asynchronous writes (see commit).

Chapter 14. Using the NFS Client

This chapter describes how to configure and maintain the VSI TCP/IP NFS Client, which allows users of OpenVMS client computers to access files on a variety of server computers.

This chapter refers to the VSI TCP/IP NFS Client and NFS Server software as the *NFS Client* and *NFS Server* and the OpenVMS client system and server system as the *client* and *server*.

14.1. Servers and Clients

The VSI TCP/IP NFS Client software allows an OpenVMS system to access file systems made available to the network by many types of server systems, including:

- Sun Microsystems hosts or similar systems running the UNIX operating system
- Hewlett-Packard computers running the ULTRIX (HP UNIX) operating system
- OpenVMS systems running the VSI TCP/IP NFS Server

The client identifies each file system by the name of its mount point on the server, which is the name of the device or directory at the top of the file system hierarchy. When mounting the file system, the client connects the mount point to a mount device in its own hierarchy; for example, NFS2:. Through this connection, all files below the mount point are available to client users as if they resided on the mount device.

The client converts all mounted directory and file structures, contents, and names to the format required by OpenVMS automatically. For example, a UNIX file named:

```
/usr/joe/.login
```

appears to an OpenVMS client user as:

```
DISK$UNIX:[USR.JOE].LOGIN;1
```

The VSI TCP/IP NFS Client can convert most valid UNIX or ULTRIX file names to valid OpenVMS names and vice versa. See the Section 14.1.9 for a complete description of the VSI TCP/IP NFS Client file-naming conventions.

14.1.1. VSI TCP/IP NFS Client Use of User IDs

NFS server systems identify each of their users to the network by a pair of UNIX or UNIX-style user-ID (UID) and group-ID (GID) codes. During an access operation, the client translates back and forth between the user's OpenVMS UIC (user identification code) and UID/GID pair.

For example, a user named Moore has an account on a UNIX server with a UID of 504 and a GID of 10. The UID and GID are mapped on the OpenVMS client to a user name BMOORE with a UIC consisting of the userid 504 and group affiliation 10.

When the NFS client ACP (ancillary control process) starts, it reads the `NFS_EXPORT.DAT, NFS_GROUP.DAT, NFS_MNTLST.DAT,` and `NFS_PROXY.DAT`. file, including the UID translations list. The client uses the list to translate each OpenVMS user name to its UID/GID pair and builds a translation table that maps UID/GID pairs to their corresponding OpenVMS UICs.

As described in the following sections, you must create and maintain the UID translation list that maps each user's OpenVMS user name to a UID/GID pair. For file protections to work properly, mappings must be both unique and consistent in each direction (see the Section 14.1.2 for a description of exceptions to this rule). You cannot map a single UID to multiple OpenVMS user names, nor can you use a single user name for multiple UIDs.

Whenever the UID/GID to OpenVMS UIC mapping is modified, the VSI TCP/IP NFS Client must be reloaded for the changes to take effect. See the Section 14.3 for more information on restarting the NFS Client.

14.1.2. Grouping NFS Client Systems for UID/GID Mappings

If all the systems in your environment share the same UID/GID pairs, you need not create or specify NFS groups. All translations are automatically placed in the default group, which has no group name associated with it.

In the database that translates between UID/GID pairs and OpenVMS user names, each entry is associated with a particular NFS group. An NFS group is a collection of NFS systems sharing a single set of UID/GID pairs. An example of a collection of systems that would be placed in an NFS group would be a UNIX file server and diskless UNIX client systems which share the same /etc/passwd file. Within an NFS group, the mapping between UID/GID pairs and OpenVMS user names must be one-to-one: you *cannot* map a single UID/GID to multiple user names, nor can you use a single user name for multiple UID/GIDs. However, duplicate translations may exist between NFS groups.

When the VSI TCP/IP NFS Client sends an NFS request to a server, it consults the local NFS group database to determine with which group the server is associated. If the server is not specified explicitly in a group, it is assumed to be in the default group. Once the NFS Client has determined the NFS group to which the server belongs, it uses the UID/GID translation list for that group to determine the UID/GID pair to use for a particular local user when accessing files on the server.

14.1.3. Mapping Example

Consider the following example. At Example, Inc., the engineering department has a group of UNIX hosts, the sales department has a collection of PCs, and the marketing department has a mix of PCs and UNIX hosts. Each group also has its own UNIX system acting as an NFS server for the group. Unfortunately, the groups did not coordinate with each other when they assigned user names and UID/GID pairs; and none of the groups are willing to change their current configurations. The accounting department, on the other hand, recently purchased an HPE RX2800 server running OpenVMS with VSI TCP/IP NFS Client. The department wants to use NFS to access personnel data that each department maintains on a local server.

The accounting system manager configures the NFS Client on the OpenVMS system as follows:

1. Using the NFS-CONFIG **ADD NFS-GROUP** command, the system manager creates the three NFS groups ENGINEERING, SALES, and MARKETING, placing the NFS systems in each department in the appropriate NFS group (the default group will be used for systems in the accounting department).

2. Departments create accounts (and hence UID/GID pairs) on their servers that the system manager can map to local OpenVMS user ids.

3. Finally, the system manager uses the NFS Configuration Utility **ADD UID-TRANSLATION** and **ADD NFS-PASSWD-FILE** commands to create mappings between OpenVMS user names and UID/GID pairs for each NFS group.

14.1.4. Effects of Incomplete Mappings

When mappings are incomplete or nonexistent, access operations are denied or severely limited.

If any server files or directories are owned by a UID for which there is no mapping, the client handles them as though they were owned by the OpenVMS user DEFAULT ([200,200]). The client grants access only according to the WORLD file protection setting of these files.

14.1.5. File System Limitations

If your VSI TCP/IP system is running both the VSI TCP/IP NFS Client and NFS Server, you cannot configure the Server to export a file system that has been mounted by the Client. The Server can export *local disks only* when the Server and Client are running on the same system.

Because the NFS Client does not completely emulate the "on disk" structure of the OpenVMS file system, some applications that directly read the file system may not work correctly over the NFS Client.

Note

The NFS Client can only mount file systems from OpenVMS systems that are using the VSI TCP/IP NFS Server.

Finally, you may notice that a file's ID (FID) changes every time the file system is remounted. This happens because the NFS protocol does not allow a local file ID to be stored on the remote host; therefore, each NFS client device (NFS*x*:) has its own idea of what the FID is. The client keeps a cache of local FIDs which it generates by choosing monotonically increasing numbers. Therefore, a file mounted multiple times in a cluster may have a different FID on each node. This might cause trouble with print queues that execute on a node other than the one that submitted the job.

14.1.6. DISKQUOTA Limitations

Although the NFS Client supports NFS server disk quotas, it does not support use of the DCL **SHOW QUOTA** command or the DISKQUOTA utility to examine or manipulate these quotas.

14.1.7. Security and File Protections

The NFS Client supports the standard OWNER, GROUP, and WORLD protections allowing READ, WRITE, EXECUTE, and DELETE access (DELETE access is taken from the WRITE access settings on the server system). Each user has a user account on the OpenVMS client as well as on the server. The NFS Client compares the UIC of the user to the owner and protection mask of the directory or file and then grants or denies access, as indicated earlier in the Section 14.1.1.

OpenVMS Access Control Lists (ACLs) are supported when accessing files on VSI TCP/IP NFS Server OpenVMS systems, as well as most UNIX NFS server systems.

14.1.8. Storing OpenVMS File Attributes on an NFS Server

The NFS protocol assumes an underlying file system in which files are merely streams of bytes with records delimited by linefeed characters (corresponding to the OpenVMS RMS Stream_LF record format). The NFS Client supports storage of non-Stream_LF files on an NFS server through attribute description files and file name-dependent attribute defaults.

When using the NFS Client, if you create a non-Stream_LF OpenVMS file or a file with ACLs associated with it on an NFS server, the NFS Client automatically creates a companion file to hold the attributes. The companion file is a text file in FDL (File Description Language) format.

The client hides the companion file from the user's view; the user sees only a single file with all of the attributes. If you rename or delete the original file from the client, the client automatically renames or deletes the companion file. However, if you rename or delete a file from the server side, you must also rename the companion file. If you do not, file attributes will be lost, the file will revert to stream attributes, and its contents may become unusable.

For example, if you create the remote indexed sequential file foo.bar, the client creates a second remote file .$fdl$foo.bar to hold the attributes.

Ordinary text files (Stream_LF files) are stored in UNIX byte-stream format and do not require companion files. If you use the OpenVMS COPY command to copy a non-Stream_LF file to an NFS client mounted disk, the file will be converted automatically to Stream_LF format. To disable this conversion, use the NOSTREAM_CONVERSION option of the **NFSMOUNT / SEMANTICS=*`qualifier`***.

Note

When communicating with a UNIX NFS server system, this option prevents UNIX users from accessing these unconverted files as Stream_LF text files.

Certain types of files default to a type other than Stream_LF, and the absence of a companion file implies attributes other than Stream_LF. For example, a `*.EXE` file defaults to a fixed length 512-byte record file. The below table lists the default file attributes included with VSI TCP/IP.

File Name	File Type	Default File Attributes
`*.EXE`	Executable	Fixed 512-byte records
`*.OBJ`	Object File	Variable-length records
`*.OLB`	Object Library	Fixed 512-byte records
`*.MAI`	Mail Folder	Indexed file, variable-length records
`*.MLB`	Macro Library	Fixed 512-byte records
`*.HLB`	Help Library	Fixed 512-byte records
`*.TLB`	Text Library	Fixed 512-byte records
`*.STB`	Symbol Table	Variable-length records
`*.DECW$BOOK`	Bookreader Book	Variable-length records
`*.DECW$FONT`	Bookreader Font	Sequential file, FORMAT undefined
`*.DECW_BOOK`	ULTRIX Book	Variable-length records
`*.UID`	DECwindows UID	Fixed 4096-byte records w/ CR

14.1.9. Storing OpenVMS File Names on an NFS Server

The VSI TCP/IP NFS Client uses a special file-naming convention to provide a one-to-one mapping between UNIX and OpenVMS file names. As a result of this convention, there are certain restrictions on names that can be assigned to files accessed using the NFS Client.

The VSI TCP/IP NFS Client attempts to give OpenVMS users access to all files on servers, even when server file names contain characters not permitted by OpenVMS. To accomplish this, the VSI TCP/IP NFS Client performs a mapping between OpenVMS and NFS server file names, using the inverse mapping of the VSI TCP/IP NFS Server. This mapping ensures consistency between other NFS clients accessing and creating files using the VSI TCP/IP NFS Server, and the VSI TCP/IP NFS Client accessing and creating files using other NFS servers. All mapping sequences on the OpenVMS client begin with the escape character "$".

As "$" is the mapping sequence escape character, a real "$" in a file name on the server is mapped to "$$" on the OpenVMS client. For example, the server file name foo$bar.c would map to FOO$ $BAR.Con the OpenVMS client.

A "$" followed by a letter (A to Z) in a file name on the client indicates a case-shift in the file name on the server. For server systems like UNIX, which support case-sensitive file names, a file name can begin in lowercase and alternate between uppercase and lowercase. For example, the server file name aCaseSENSITIVEFilename would map to A$C$ASE$SENSITIVEF$ILENAME on the OpenVMS client. A "$" followed by any digit 4 to 9 indicates a mapping as shown below.

VMS Char.	Server Char.	Hex Value	VMS Char.	Server Char.	Hex Value	VMS Char.	Server Char.	Hex Value
$4A	^A	1	$5A	!	21	$7A	Space	20
$4B	^B	2	$5B	“	22	$7B	;	3B
$4C	^C	3	$5C	#	23	$7C	<	3C
$4D	^D	4	$5E	%	25	$7D	=	3D
$4E	^E	5	$5F	&	26	$7E	>	3E
$4F	^F	6	$5G	‘	27	$7F	?	3F
$4G	^G	7	$5H	(	28			
$4H	^H	8	$5I	)	29	$8A	@	40
$4I	^I	9	$5J	*	2A	$8B	[	5B
$4J	^J	A	$5K	+	2B	$8C	\	5C
$4K	^K	B	$5L	’	2C	$8D	]	5D
$4L	^L	C	$5N	.	2E	$8E	^	5E
$4M	^M	D	$5O	/	2F			
$4N	^N	E	$5Z	:	3A	$9A	‘	60
$4O	^O	F				$9B	{	7B
$4P	^P	10	$6A	^@	00	$9C	|	7C
$4Q	^Q	11	$6B	^[	1B	$9D	}	7D
$4R	^R	12	$6C	^\	1C	$9E	~	7E
$4S	^S	13	$6D	^]	1D	$9F	DEL	7F
$4T	^T	14	$6E	^^	1E			
$4U	^U	16	$6F	^_	1F			
$4V	^V	16						
$4W	^W	17						
$4X	^X	18						
$4Y	^Y	19						

VMS Char.	Server Char.	Hex Value	VMS Char.	Server Char.	Hex Value	VMS Char.	Server Char.	Hex Value
$4Z	^Z	1A						

The digit after the dollar sign and the trailing letter indicates the character in the server file name. In the special case of the "dot" character (.), the first dot in the server file name maps directly to a dot in the client OpenVMS file name. Any following dot characters on the server are mapped to the character sequence $5N on the OpenVMS client. In directory files, any dot character in the server file name maps to $5N on the client. For example, the server file name foo.bar#1.old maps to FOO.BAR $5C1$5NOLD on the OpenVMS client (unless foo.bar#1.old is a directory file, in which case it maps to FOO$5NBAR$5C1$5NOLD.DIR on the OpenVMS client).

The NFS Client also supports OpenVMS file version numbers. If a file created using the NFS Client has a file version number other than 1, the resulting file on the server contains the OpenVMS version number. The highest version of the file is hard-linked to the name without the version number.

Finally, a "$" followed by a three-digit octal number indicates a character in the file name on the server that has the binary value of that three-digit octal number. As all character binary values from 0 to 177 (octal) already have mappings, only characters from 200 to 377 are mapped in this fashion. Thus, the leading digit of the octal number must be either 2 or 3.

14.1.10. NFS Client Architecture

The NFS Client consists of a device driver and an ACP process that receives requests from the OpenVMS Record Management Services (RMS) and translates them into NFS requests. Because the NFS Client is called by RMS, applications using RMS or the standard input/output routines of OpenVMS programming languages do not need to be modified to access files through the NFS Client. The NFS Client presents a $QIO interface identical to the interface documented in the *VSI OpenVMS I/O User's Reference Manual*.

The NFS Client includes two top-level protocols that run parallel to each other above a stack of lower-level protocols:

- The Network File System (NFS) protocol is an IP-family protocol that provides remote file system access, handling client queries.
- The Remote Procedure Call (RPC) mount protocol, RPCMOUNT, is used by the NFSMOUNT and NFSDISMOUNT commands to get mount-point information from the server systems.

Underlying the NFS and RPCMOUNT protocols is a stack of protocols:

- The remote Procedure Call (RPC) protocol allows the client to make procedure calls across the network to servers.
- The external Data Representation (XDR) protocol handles architectural differences between the client and server systems, allowing the NFS protocol to communicate between systems with dissimilar architectures.
- The RPC/NFS Lock Manager protocol allows the NFS Client to support file-locking (exclusive write access).
- User Datagram Protocol (UDP), Transmission Control Protocol (TCP), and Internet Protocol (IP) are used for the lowest levels of communication.

Traditionally, NFS has only run over UDP. The VSI TCP/IP NFS Client also supports communication over TCP. This may provide reliability and performance improvements when communicating with NFS server systems across slow network links or wide area networks (WANs), which suffer from packet loss and delay.

14.2. Mounting and Dismounting File Systems

The final step in performing the first NFS Client configuration is to mount the remote file systems that you want client users to access as if they were local files. You can also modify an existing client configuration by mounting or dismounting file systems.

14.2.1. Mounting a File System

Use the NFSMOUNT command to mount an NFS file system. NFSMOUNT requires CMKRNL, SETPRV, SYSPRV, SYSNAM, ALTPRI, DETACH, ACNT, and SYSLCK privileges. For example:

```
$ NFSMOUNT sun::"/ufs" disk$sun
%NFSMOUNT-I-MOUNTED, /usr NFS mounted on _NFS3:[000000]
$
```

The example command mounts the file system /ufs which is located on the server sun on the local mount device _NFS2:.

The double quotes are necessary in the sample command because of the special meaning of the slash (/) character in OpenVMS. The quotes are not necessary when mounting a file system exported by another OpenVMS system.

14.2.2. Dismounting a File System

When you dismount a file system, you free the resources used by the NFS client. To dismount a file system, use the **NFSDISMOUNT** command:

```
$ nfsdismount mount_device
```

For example:

```
$ nfsdismount nfs2:
```

You can use either the logical name specified in the **NFSMOUNT** command or the actual NFS device name (such as NFS2:) in the *`mount_device`* field of the **NFSDISMOUNT** command.

14.3. Reloading the NFS Client

Before you can use a new or revised set of UID translations, you must first reload the UID mappings into the NFS Client with the DCL command:

```
$ NFSMOUNT /RELOAD
```

You may also update the client's UID mappings using NFS-CONFIG with the following command:

```
NFS-CONFIG>RELOAD
```

For instructions on using the **NFSDISMOUNT** and **NFSMOUNT** commands, see the Section 14.2.

Note

If no file systems are mounted, reloading does not work.

14.4. Mounting File Systems During VSI TCP/IP Startup

When the `IP$SYSTARTUP.COM` script executes during VSI TCP/IP startup, it checks the IP$: directory for the existence of a file named `NFS_MOUNT.COM`. If this file exists, it will be executed in order to mount any remote file system desired by the system manager. The following example illustrates such a file:

```
$ SET NOON
$ Show Queue 'F$GetSYI("NODENAME")'_BATCH/Output=Sys$Manager:Nfs_Mount.Tmp/
All
$ Open/Read File SYS$MANAGER:NFS_Mount.Tmp
$Loop:
$ Read/End=Done File Line
$ If "''F$Element(1," ",F$Edit(Line,"TRIM,COMPRESS"))'" .Eqs. "NFS_MOUNT"
-
    Then Goto Skip
$ Goto Loop
$Done:
$ Submit/User=System/Queue='F$GetSYI("NODENAME")'_BATCH -
     /NoPrint -
     IP$:NFS_MOUNT_BATCH -
     /Name=NFS_MOUNT/Log=Sys$Manager:NFS_Mount.Log
$Skip:
$ Close File
$ Delete Sys$Manager:NFS_Mount.Tmp;*
```

This DCL program submits another DCL command file, `NFS_MOUNT_BATCH.COM`, to the queue node_BATCH to do the work after the system boots.

The following is an example of `NFS_MOUNT_BATCH.COM`:

```
$ Verify = 'f$verify(0)
$ Set Proc/Name="NFS Mounter"
$ Purge NFS_MOUNT.Log
$ SET NOON
$ Errors = 0
$!
$! Attempt to mount the CD player on NFS.EXAMPLE.COM
$!
$ IF .Not. F$GetDVI("DISK$CD","EXISTS") Then -
NFSMOUNT/VMS/TRANS=TCP/SOFT NFS.EXAMPLE.COM::DISK$CD: DISK$CD -
/VOLUME="CD_ROM"
$ If .Not. $Status Then Errors = Errors + 1
$!
$! Check the status and requeue job if necessary.
$!
$ If Errors .Eq. 0 Then Exit
$ Submit/User=System/Queue=SYS$BATCH -
    /NoPrint /Name=NFS_MOUNT -
    IP$:NFS_MOUNT_BATCH -
    /After="''F$CvTime("+01:00","ABSOLUTE")'"
```

14.5. Creating ACPs (Ancillary Control Processes) for NFS Mounts

The NFS Client has an NFS_CLIENT_ACP process that assists the driver by performing some operations that are easier to do in a separate process rather than at the driver level.

Because this ACP process is single-threaded, using a single ACP for all NFS devices has a significant drawback. If you have multiple NFS devices mounted to different computer systems and an operation hangs on one system, all of the NFS devices are affected.

Specifying **NFSMOUNT /PROCESSOR=UNIQUE** creates a separate ACP process for each NFS device. This allows NFS devices to function in parallel so one device does not have to wait for an NFS operation on another device to complete. Multiple ACPs provide for multiple outstanding I/O operations on different devices.

The setting **/PROCESSOR=UNIQUE** creates a separate NFS_CLIENT_*n* process for each mount, *n* is the number of the NFS device (for example, NFS_CLIENT_2, which corresponds to the device NFS2).

The following example illustrates the use of **/PROCESSOR=UNIQUE**, creating four ACP processes (one for each device):

```
$ NFSMOUNT /PROCESSOR=UNIQUE SCROOGE::USERS: SCROOGE$USERS
$ NFSMOUNT /PROCESSOR=UNIQUE PIP::UTIL: PIP$UTIL
$ NFSMOUNT /PROCESSOR=UNIQUE HAVERSHAM::ADMIN: HAVERSHAM$ADMIN
$ NFSMOUNT /PROCESSOR=UNIQUE MARLEY::ENG:MARLEY$ENG
```

A setting of **/PROCESSOR=SAME=*nfs_device*** assigns the mount to the same ACP process as the specified *nfs_device*. For example, **/PROCESSOR=SAME=NFS3** assigns this mount to the NFS_CLIENT_3 ACP process.

Note

The specified device may be either the NFS device name itself (for example, NFS3), or a logical name pointing at the NFS device.

Mounts specified without the **/PROCESSOR** qualifier use the default process NFS_CLIENT_ACP.

VSI recommends that you use the **/PROCESSOR** qualifier to group mounts on the remote server. If the server goes down, access to other servers is not affected. You can use the

/SOFT qualifier to permit NFS operations to time-out instead of hanging indefinitely.

The following example illustrates the use of **/PROCESSOR=SAME**. In this example, all access to the server SCOOBY goes through one ACP process, and all access to PIP goes through another process.

```
$ NFSMOUNT /PROCESSOR=UNIQUE SCROOGE::USERS: SCROOGE$USERS
$ NFSMOUNT /PROCESSOR=SAME=SCROOGE$USERS SCROOGE::DKA100: SCROOGE$DKA100
$ NFSMOUNT /PROCESSOR=UNIQUE PIP::UTIL: PIP$UTIL
$ NFSMOUNT /PROCESSOR=SAME=PIP$UTIL PIP::FOO: PIP$FOO
```

14.6. NFS Clients Using BACKUP

The OpenVMS **BACKUP** utility can write a saveset to an NFS-mounted disk, but the VSI TCP/IP NFS Client does not support specifying files on an NFS-mounted disk as the *input-specifier* in a **BACKUP** command.

BACKUP works with the VSI TCP/IP NFS Client in a *limited* way with the following restrictions:

- **BACKUP** preserves the UIC it finds on a file. If an NFS UNIX file has a UID that does not map to an OpenVMS UIC, the file is backed up as if it belonged to **DEFAULT**. When you restore the file, it will belong to the UNIX user nobody (UID -2, GID -2).
- **BACKUP** does not preserve certain bits of information associated with UNIX (such as the "sticky" or set-UID bits).
- NFS identifies UNIX files using a 32-byte file handle. However, the file handle must be presented to OpenVMS as a 6-byte FID. Because the number of possible 32-byte file handles is much greater than the number of possible 6-byte FIDs, VSI TCP/IP must implement a cached mapping scheme. This approach works well with applications that only care about FID consistency as long as the file is accessed. However, some applications (such as **BACKUP**) expect consistent FIDs for the life of the file.

Note

Using **BACKUP** with NFS-mounted files copies the contents of the files, but does not copy the semantics of files created from foreign operating systems. VSI recommends backing up OpenVMS files to a remote tape via RMT (Remote Magtape Protocol).

Chapter 15. Configuring the Secure Shell (SSH) Servers Versions 1 & 2

This chapter describes how to configure and maintain the VSI TCP/IP Secure Shell (SSH) Server of versions 1 and 2.

This is the server side of the software that allows secure interactive connections to other computers in the manner of rlogin/rshell/telnet. The SSH server has been developed to discriminate between SSH v1 and SSH v2 protocols, so the two protocols can coexist simultaneously on the same system.

15.1. SSH1 and SSH2 Differences

SSH1 and SSH2 are different, and incompatible, protocols. While SSH2 is generally regarded to be more secure than SSH1, both protocols are offered by VSI TCP/IP, and although they are incompatible, they may exist simultaneously on a VSI TCP/IP system. The VSI TCP/IP server front-end identifies what protocol a client desires to use, and will create an appropriate server for that client.

The cryptographic library used by VSI TCP/IP SSH2 (***this <u>does not</u> apply to SSH1 sessions***) is FIPS 140-2 level 2 compliant, as determined by the Computer Security Division of the National Institute of Science and Technology (NIST).

Note

You must install the DEC C 6.0 backport library on all OpenVMS VAX v5.5-2 and v6.0 systems prior to using SSH. This is the `AACRT060.A` file. You can find the ECO on the VSI TCP/IP CD the following directory: VAX55_DECC_RTL.DIR.

Restrictions:

When using SSH1 to connect to a OpenVMS server, if the OpenVMS account is set up with a secondary password, SSH1 does not prompt the user for the secondary password. If the OpenVMS primary password entered is valid, the user is logged in, bypassing the secondary password.

When using SSH1 to execute single commands (in the same manner as RSHELL), some keystrokes like **CTRL/Y** are ignored. In addition, some interactive programs such as HELP may not function as expected. This is a restriction of SSH1. If this behavior poses a problem, log into the remote system using SSH1 in interactive mode to execute the program.

15.2. Understanding the VSI TCP/IP Secure Shell Server

Secure Shell daemon (SSHD) is the daemon program for SSH that listens for connections from clients. The server program replaces rshell and telnet programs. The server/client programs provide secure encrypted communications between two untrusted hosts over an insecure network. A new daemon is created for each incoming connection. These daemons handle key exchange, encryption, authentication, command execution, and data exchange.

15.2.1. Servers and Clients

A VSI TCP/IP SSH server is an OpenVMS system server that acts as a host for executing interactive commands or for conducting an interactive session. The server software consists of two processes (for future reference, SSHD will refer to both SSHD_MASTER and SSHD, unless otherwise specified):

- SSHD_MASTER, recognizes the differences between SSH v1 and SSH v2 and starts the appropriate server. If the request is for SSH v1, then a new SSH v1 server is run; if the request is for SSH v2, then a new SSH v2 server is run.
- SSHD, a copy of which is spawned for each time a new connection attempt is made from a client. SSHD handles all the interaction with the SSH client.

A client is any system that accesses the server. A client program (SSH) is provided with VSI TCP/IP, but any SSH client that uses SSH version 1 protocol may be used to access the server. Examples of such programs are FISSH, VSI TCP/IP SSH, and TCPware SSH on OpenVMS systems; TTSSH, SecureCRT, F-Secure SSH Client, and PuTTY on Windows®-based systems; and other SSH programs on UNIX-based systems.

15.2.2. Security

Each host has a host-specific RSA key (normally 1024 bits) that identifies the host. Additionally, when the SSHD daemon starts, it generates a server RSA key (normally 768 bits). This key is regenerated every hour (the time may be changed in the configuration file) if it has been used, and is never stored on disk. Whenever a client connects to the SSHD daemon:

- SSHD sends its host and server publickeys to the client.
- The client compares the hostkey against its own database to verify that it has not changed.
- The client generates a 256 bit random number. It encrypts this random number using both the hostkey and the server key, and sends the encrypted number to the server.
- The client and the server start to use this random number as a session key which is used to encrypt all further communications in the session.

The rest of the session is encrypted using a conventional cipher. Currently, IDEA (the default), DES, 3DES, Blowfish, and ARCFOUR are supported.

- The client selects the encryption algorithm to use from those offered by the server.
- The server and the client enter an authentication dialog.
- The client tries to authenticate itself using any of the following methods:
    - `.rhosts` authentication
    - `.rhosts` authentication combined with RSA host authentication
    - RSA challenge-response authentication
    - password-based authentication

Note

Rhosts authentication is normally disabled because it is fundamentally insecure, but can be enabled in the server configuration file, if desired.

System security is not improved unless the RLOGIN and RSHELL services are disabled.

When the client authenticates itself successfully, a dialog is entered for preparing the session. At this time the client may request things such as:

- forwarding X11 connections
- forwarding TCP/IP connections
- forwarding the authentication agent connection over the secure channel

Finally, the client either requests an interactive session or execution of a command. The client and the server enter session mode. In this mode, either the client or the server may send data at any time, and such data is forwarded to/from the virtual terminal or command on the server side, and the user terminal in the client side. When the user program terminates and all forwarded X11 and other connections have been closed, the server sends command exit status to the client, and both sides exit.

15.2.3. Break-in and Intrusion Detection

Care must be exercised when configuring the SSH clients and server to minimize problems due to intrusion records created by OpenVMS security auditing. The SSH user should consult the system manager to determine the authentication methods offered by the SSH server. The client should then be configured to not attempt any authentication method that is not offered by the server.

If a client attempts authentication methods not offered by the server, the OpenVMS security auditing system may log several intrusion records for each attempt to create a session to that server. The result being that the user could be locked out and prevented from accessing the server system without intervention from the server's system manager.

The authentication methods to be offered by the server are determined by the configuration keywords RhostsAuthentication, RhostsRSAAuthentication, RSAAuthentication, and PasswordAuthentication. The number of intrusion records to be logged for any attempted SSH session is determined by the StrictIntrusionLogging configuration keyword.

When StrictIntrusionLogging is set to YES (the default), each method that is tried and fails causes an intrusion record to be logged. When Rhosts, RhostsRSA or RSA authentications are attempted and fail, one intrusion record will be logged for each failed method.

When password authentication is attempted, one intrusion record will be logged for each failed password.

Example 1

The server is set up to allow Rhosts, RSA, and password authentication; also, up to three password attempts are allowed. If all methods fail, five intrusion records are logged:

1 for the failed Rhosts

1 for the failed RSA

3 for the failed password attempts, one per attempt

When StrictIntrusionLogging is set to NO, it has the effect of relaxing the number of intrusions logged. Overall failure of all authentication methods simply counts as a single failure, except for password authentication. The following rules apply:

- When password authentication is attempted, one intrusion record is logged for each failed password.
- When any of Rhosts, RhostsRSA, or RSA authentication fails, and password authentication is not attempted, exactly one intrusion record is logged, as opposed to one for each failed method.
- When any of Rhosts, RhostsRSA, or RSA authentication fails, but password authentication is attempted and succeeds, the only intrusion record(s) logged is one for each failed password attempt.

Example 2:

The server is set up to allow Rhosts, RSA, and password authentication; also, up to three password attempts are allowed. If all methods fail, three intrusion records are logged:

0 for the failed Rhosts

0 for the failed RSA

3 for the failed password attempts, one per attempt

Example 3:

The server is set up to allow Rhosts, RSA, and password authentication; also, up to three password attempts are allowed. Rhosts and RSA fail, but password authentication is successful after 1 failed password. Therefore, one intrusion record is logged:

0 for the failed Rhosts

0 for the failed RSA

1 for the failed password attempt

Example 4:

The server is set up to allow Rhosts, RhostsRSA, and RSA authentication, but not password authentication. If all methods fail, one intrusion record is logged.

Example 5:

The server is set up to allow Rhosts, RhostsRSA, and RSA authentication, but not password authentication. Rhosts and RSA authentication both fail, but RhostsRSA succeeds. No intrusion records are logged.

15.3. Configuring SSHD Master for SSH1

SSHD Master is configured using the **IP CONFIGURE/SERVER** command, selecting SSH, and using the following options:

Note

The recommended method to start SSHD Master is to use the **IP NETCONTROL SSH START** command. All of these options are set using **IP CONFIG /SERVER**, and modifying the SSH service.

```
$ IP CONFIGURE /SERVER
VSI TCP/IP Server Configuration Utility
[Reading in configuration from IP$:SERVICES.MASTER_SERVER]
SERVER-CONFIG>SELECT SSH
[The Selected SERVER entry is now SSH]
SERVER-CONFIG>SET PARAM
Delete parameter "enable-ssh1" ? [NO]
Delete parameter "enable-ssh2" ? [NO]
You can now add new parameters for SSH.  An empty line terminates.
Add Parameter:PORT 33000
Add Parameter:
[Service specific parameters for SSH changed]
SERVER-CONFIG>SHOW/FULL
Service "SSH":
         INIT() = Merge_Image
         Program = "IP$:LOADABLE_SSH_CONTROL"
         Priority = 5
         Log for Accepts & Rejects = OPCOM
         Parameters = "enable-ssh1"
                      "enable-ssh2"
                      "port 33000"
SERVER-CONFIG>EXIT
[Writing configuration to IP$COMMON_ROOT:[IP]SERVICES.MASTER_SERVER]
```

15.4. Expired Passwords

The SSH v1 protocol does not provide a method for changing an expired OpenVMS password. When an expired password is encountered by the SSH1 server, it will do one of two things.

1. If the logical name `IP$SSH_ALLOW_EXPIRED_PW` is defined for allowing access for passwords that have exceeded the UAF value for PWDLIFETIME, or if the logical name `IP$SSH_ALLOW_PREEXPIRED_PW` is defined for allowing access for users that have a pre-expired password, the server will allow the user to log in. In the logical name table `LNM$SSH_LOGICALS`, the logical name `IP$SSH_`*`pid`*`_PWDEXP` (where *`pid`* is the process ID for the user process) will be defined. The system manager can look for this logical to be defined, and if so, take action such as executing the DCL **SET PASSWORD** command.

2. If the appropriate logical is not set as described above, the user will be denied access to the system. In that case, the user must log in interactively via another mechanism such as telnet and change the password, or the system manager must reset the password.

When a user is allowed access to the system with an expired password, the LOGIN_FLAGS for the process will reflect this. The values of the LOGIN_FLAGS will be as follows:

- new mail has been received (`JPI$M_NEW_MAIL_AT_LOGIN`)
- the password is about to expire (`JPI$M_PASSWORD_WARNING`)
- the password has expired (`JPI$M_PASSWORD_EXPIRED`)

The DCL lexical function F$GETJPI may be used to examine these flags, as can the $GETJPI(W) system service or LIB$GETJPI RTL function. When an expired password value is detected, the user may then execute a **SET PASSWORD** command in the command procedure run for the account.

For example:

```
$!
$! Login_flags:
$!    1 = new mail messages waiting (JPI$M_NEW_MAIL_AT_LOGIN)
$!    4 = password expired during login (JPI$M_PASSWORD_EXPIRED)
$!    5 = password expires within 5 days (JPI$M_PASSWORD_WARNING)
$!
$ flags = f$getjpi("", "LOGIN_FLAGS")
$ new_flags = (flags/2)*2
$ if new_flags .ne. flags then write sys$output "New mail waiting"
$!
$!Note - new_flags is used below because it has the NEW_MAIL_AT_LOGIN$
$!   bit stripped.  The rest of the possible values are all
$!   discrete; i.e., you can't have combinations of them at the
$!   same time.
$!
$ if new_flags .eq. 4 then write sys$output "Password expired during login"
$ if new_flags .eq. 5 then write sys$output "Password expires within 5
 days"
$!
```

15.5. OPTIONS

bits *n*

Specifies the number of bits in the server key. The default is 768.

ssh1-config-file *filename*

Specifies the name of the configuration file. The default is `IP$:SSHD_CONFIG`.

debug *debug-level*

Turns debugging on using any non-zero debug level.

enable-ssh1

Enables SSH v1 sessions.

host-key-file *filename*

Specifies the file from which the hostkey is read. The default is `IP$:SSH_HOST_KEY`.

keygen-time *n*

Specifies how often the server key is regenerated. The default is 3600 seconds (one hour). The motivation for regenerating the key often is that the key is never stored physically on disk. It is kept in the address space of the server, and after an hour, it becomes impossible to recover the key for decrypting intercepted communications even if the machine is broken into or physically seized. A value of zero indicates that the key will never be regenerated.

listen-address

Specify the IPV4 address on which to listen for connect request. This may be a valid IPV4 address or ANY to listen on all addresses. If not specified, the default is to listen on all addresses.

port *n*

Specifies the port on which the server listens for connections. The default is 22.

quiet_mode

Specifies that nothing is sent to the SSH system log. Normally, the beginning, authentication, and termination of each connection is logged.

verbose

Specifies that verbose message logging will be performed by SSHD MASTER.

15.6. Configuration File

SSHD reads configuration data from `IP$SPECIFIC:[IP.SSH]` (or the file specified with the ssh1-config-file keyword in **IP CONFIGURE/SERVER**). The file contains keyword value pairs, one per line. The following keywords are possible. Keywords are case insensitive.

Keyword	**Value**	**Default**	**Description**
AllowForwardingPort	Port list		Permit forwarding for the specified ports
AllowForwardingTo	Host/port list		Permit forwarding for hosts
AllowGroups	List		Access control by UAF rightslist entries
AllowHosts	Host list		Access control by hostname
AllowShosts	Host list		Access control by hostname
AllowTcpForwarding	Y/N	Y	Enable TCP port forwarding
AllowUsers	User list		Access control by username
DenyForwardingPort	Port list		Forbid forwarding for ports
DenyForwardingTo	Host/port list		Forbid forwarding for hosts
DenyGroups	Rights list		Deny access for UAF rightslist identifiers
DenyHosts	Host list		Deny access for hosts
DenySHosts	Host list		Deny access for hosts
DenyUsers	User list		Access control by username
FascistLogging	Y/N	Y	Verbose logging
Hostkey	Filename	Ssh_host_key.	Hostkey filename
IdleTimeout	Time	0 (infinite)	Set idle timeout
IgnoreRhosts	Y/N	N	Ignore local rhosts
IgnoreRootRhosts	Y/N	Y	Ignore system rhosts

Keyword	Value	Default	Description
KeepAlive	Y/N	Y	Send keepalives
ListenAddress	IP address	0.0.0.0	Listen on given interface
LoginGraceTime	Time	600	Time limit for authentication in seconds
PasswordAuthentication	Y/N	Y	Permit password authentication
PermitEmptyPasswords	Y/N	N	Permit empty (blank) passwords
PermitRootLogin	Y/N	N	SYSTEM can log in
QuietMode	Y/N	N	Quiet mode
RandomSeed	Filename	Random_seed	Random seed file
RhostsAuthentication	Y/N	N	Enable rhosts authentication
RhostsRSAAuthentication	Y/N	Y	Enable rhosts with RSA authentication
RSAAuthentication	Y/N	Y	Enable RSA authentication
StrictIntrusionLogging	Y/N	Y	Determine how intrusion records are created by failed authentication attempts
StrictModes	Y/N	N	Strict checking for directory and file protection
SyslogFacility	Syslog level	"DAEMON"	Syslog log facility
VerboseLogging	Y/N	Y	Verbose logging (also known as FacistLogging)
X11Forwarding	Y/N	Y	Enable X11 forwarding
X11DisplayOffset	#offset	10	Limit X displays for SSH

15.7. Starting the SSH Server for the First Time

Follow these instructions for using SSH for the first time.

1. Use the **IP CONFIGURE /SERVER** command to enable the SSH v1 server.

```
$ IP CONFIGURE/SERVER
VSI TCP/IP Server Configuration Utility
[Reading in configuration from IP$:SERVICES.MASTER_SERVER]
SERVER-CONFIG>SHOW/FULL SSH
Service "SSH": ***DISABLED***
        INIT() = Merge_Image
        Program = "IP$:LOADABLE_SSH_CONTROL"
        Priority = 5
        Parameters = “enable-ssh1”
                     “enable-ssh2”
SERVER-CONFIG>ENABLE SSH
SERVER-CONFIG>EXIT
[Writing configuration to IP$COMMON_ROOT:[IP]SERVICES.MASTER_SERVER]
```

Note

The parameter enable-ssh1 must be set. If it is not set, SSH v1 sessions will not be accepted by the server.

2. Use SSHKEYGEN /SSH1 to generate an ssh1 key and to create the file SSH_HOST_KEY in the IP$: directory.

```
$ IP SSHKEYGEN /SSH1 /HOST
Initializing random number generator...
Generating p:  ...++ (distance 64)
Generating q:  ......................................++ (distance 516)
Computing the keys...
Testing the keys...
Key generation complete.
Key file will be IP$ROOT:[IP]SSH_HOST_KEY.
Your identification has been saved in IP$:SSH_HOST_KEY.
Your publickey is:
1024 37
12103183655766986978653678692919694763882284449699056118642763O8
9072776904462744415966821020109463617644202397294642277946718549
4404442577594868297087171013359743853182442579923801302020844011
5343754909847513973160249324735913146330232410424936751015953611
18716872491123857940537322891584850459319961275605927
SYSTEM@roadrr.acme.com
Your publickey has been saved in IP$ROOT:[IP]SSH_HOST_KEY.pub
```

3. Copy the template configuration file to the IP: directory renaming it to SSHD_CONFIG.;

```
$ COPY IP$SPECIFIC:[IP.SSH].TEMPLATE IP$SPECIFIC:[IP.SSH].;
```

Note

As delivered, the template file provides a reasonably secure SSH environment. However, VSI recommends this file be examined and modified appropriately to reflect the security policies of your organization.

4. Restart VSI TCP/IP. This creates the SSH server process and defines the SSH logical names.

```
$ @IP$:IP$SYSTARTUP.COM RESTART
$ SHOW PROCESS "SSHD Master"
7-APR-2016 09:03:06.42  User: SYSTEM      Process ID:   00000057
                        Node: PANTHR      Process name: "SSHD Master"
Terminal:
User Identifier:    [SYSTEM]
Base priority:      4
Default file spec:  Not available
Number of Kthreads: 1
Devices allocated:  BG1:
                    BG2:
$ SHOW LOGICAL/SYSTEM SSH*
(LNM$SYSTEM_TABLE)
  "SSH_DIR" = "IP$SPECIFIC_ROOT:[IP]"
  "SSH_EXE" = "IP$COMMON_ROOT:[IP]"
  "SSH_LOG" = "IP$SPECIFIC_ROOT:[IP.SSH]"
```

```
    "SSH_TERM_MBX" = "MBA23:
```

15.8. Configuring the SSH1 Server on a OpenVMScluster with a Common System Disk

When configuring the SSH1 server on a OpenVMScluster with a common system disk, you must create the appropriate directories on all cluster nodes other than one on which VSI TCP/IP was originally installed. Note that this does not need to be done for cluster members that do not share a common system disk.

The following procedure should be followed on each cluster node other than the cluster node on which VSI TCP/IP was originally installed:

- Create the necessary directory:

  ```
  $ CREATE/DIR IP$SPECIFIC[IP$SSH]/PROT=(WO:RE,GR:RE)
  ```

- Edit the `IP$SPECIFIC:[IP.SSH]SSHD_CONFIG` file as necessary. This may be copied from another cluster node, or it may be created fresh from the `SSHD_CONFIG.TEMPLATE` file.
- Configure the SSH1 server using **IP CONFIGURE /SERVER**
- Generate the SSH1 hostkeys using **IP SSHKEYGEN/SSH1 /HOST**
- (Re)start SSHD Master using **IP NETCONTROL SSH RESTART**

15.9. Changing SSH1 Configuration File after Enabling SSH1

If you make a change to the SSH1 configuration file after you have enabled SSH1, you must restart SSH for these changes to take effect.

```
$ IP NETCONTROL SSH RESTART
```

Note

When issuing the **RESTART** command for SSH, all active SSH server sessions are terminated. Active client sessions are not affected.

15.10. Connection and Login Process

To create a session, SSHD does the following:

1. SSHD_MASTER process sees the connection attempt. It creates an SSHD v1 or v2 process, depending on the protocol version presented to it by the client. SSHD_MASTER then passes necessary information to the SSHD process, such as the server key and other operating parameters.
2. SSHD process performs validation for the user.

3. Assuming the login is successful, SSHD process creates a pseudoterminal for the user (an _FTAnn: device). This device is owned by the user logging in.

4. SSHD process creates an interactive process on the pseudoterminal, using the username, priority, and privileges of the user logging in. If a command was specified, it is executed and the session is terminated.

5. SSH generates the file SSHD.LOG in the directory `IP$ROOT:[IP.SSH]` for each connection to the SSH server. Many connections result in many log files. Instead of purging the files on a regular basis, use the following DCL command to limit the number of versions:

   ```
   $ SET FILE /VERSION_LIMIT=x IP$ROOT:[IP.SSH]SSHD.LOG
   ```

Note

The value for **/VERSION_LIMIT** must not be smaller than the maximum number of simultaneous SSH sessions anticipated. If the value is smaller, SSH users may be prevented from establishing sessions with the server.

Note

When the SSHD.LOG file version reaches the maximum number of 32767, new log files are not generated. SSH connections still function, however, VSI recommends that you rename the SSHD.LOG to fix the problem. You can change the log file name by defining the logical name IP $SSH_LOG_FILE with one or more of the following tokens:

```
%D date in yyyymmdd format
%N system SCS node name
%C value of childcount
```

For example, defining the following logical name:

```
$ DEFINE /SYSTEM IP$SSH_LOG_FILE "SSH_%N_%D"
```

This results in the following SSH log file names:

```
IP$LOG:SSH_MYNODE_20190513.LOG
```

VSI also recommends that files be removed regularly to allow space for new file creation.

15.10.1. SSH Connections Are Not Logged When SSHD.log Files Reach Maximum Version Number

15.11. FILES

```
IP$:HOSTS.EQUIV
```

Contains host names, one per line. This file is used during .rhosts authentication. Users on those hosts are permitted to log in without a password, provided they have the same username on both machines. The hostname may also be followed by a username. Such users are permitted to log in as any user on the remote machine (except SYSTEM). Additionally, the syntax +@*group* can be used to specify netgroups. Negated entries start with a dash (-). If the client host/user is matched in this file, login is permitted provided the client and server usernames are the same. Successful RSA host authentication is required. This file should be world-readable but writeable only by SYSTEM.

It is never a good idea to use usernames in hosts.equiv. It means the named user(s) can log in as anybody, which includes accounts that own critical programs and directories. Using a username grants the user SYSTEM access. The only valid use for usernames is in negative entries.

Note

This warning also applies to rshell/rlogin.

`IP$:SHOSTS.EQUIV`

Processed as IP$:HOSTS.EQUIV. May be useful in environments that want to run both rshell/rlogin and ssh.

`IP$:SSH_HOST_KEY`

Contains the private part of the hostkey. This file does not exist when VSI TCP/IP is first installed. The SSH server starts only with this file. This file must be created manually using the command:

```
$ IP SSHKEYGEN /SSH1 /HOST
```

This file should be owned by SYSTEM, readable only by SYSTEM, and not accessible to others.

To create a hostkey with a name that is different than what SSHKEYGEN creates, do one of the following:

- Generate with **IP SSHKEYGEN /SSH1 /HOST** and simply rename the file.
- Generate a public/private key pair using SSHKEYGEN without the **/HOST** switch, and copying and renaming the resulting files appropriately.

By default the logical name SSH_DIR points to the `IP$SPECIFIC_ROOT:[IP]` directory.

Refer to the *VSI TCP/IP User's Guide* for more details about SSHKEYGEN.

`IP$:SSH_HOST_KEY.PUB`

Contains the public part of the hostkey. This file should be world-readable but writeable only by SYSTEM. Its contents should match the private part of the key. This file is not used for anything; it is only provided for the convenience of the user so its contents can be copied to known hosts files.

`IP$:SSH_KNOWN_HOSTSSYS$LOGIN:[.SSH]KNOWN_HOSTS`

Checks the publickey of the host. These files are consulted when using rhosts with RSA host authentication. The key must be listed in one of these files to be accepted. (The client uses the same files to verify that the remote host is the one you intended to connect.) These files should be writeable only by SYSTEM (the owner). `IP$:SSH_KNOWN_HOSTS` should be world-readable, and `SYS$LOGIN:[.SSH]KNOWN_HOSTS` can, but need not be, world-readable.

`SSH2:SSH_RANDOM_SEEDSYS$LOGIN:[.SSH]RANDOM_SEED`

Contains a seed for the random number generator. This file should only be accessible by system.

`IP$SPECIFIC:[IP.SSH]`

Contains configuration data for SSHD. This file should be writeable by system only, but it is recommended (though not necessary) that it be world-readable.

`AUTHORIZED_KEYS`

In the user's `SYS$LOGIN[.SSH]` directory

Lists the RSA keys that can be used to log into the user's account. This file must be readable by system. It is recommended that it not be accessible by others. The format of this file is described below.

`SYS$LOGIN:.SHOSTS`

In the user's `SYS$LOGIN:[.SSH]` directory

Permits access using SSH only. For SSH, this file is the same as for .rhosts. However, this file is not used by rlogin and rshell daemon.

`SYS$LOGIN:.RHOSTS`

This file contains host-username pairs, separated by a space, one per line. The given user on the corresponding host is permitted to log in without a password. The same file is used by rlogin and rshell. SSH differs from rlogin and rshell in that it requires RSA host authentication in addition to validating the hostname retrieved from domain name servers. The file must be writeable only by the user. It is recommended that it not be accessible by others. It is possible to use netgroups in the file. Either host or username may be of the form +@*`groupname`* to specify all hosts or all users in the group.

15.12. AUTHORIZED_KEYS File Format

The `SYS$LOGIN:[.SSH]AUTHORIZED_KEYS` file lists the RSA keys that are permitted for RSA authentication. Each line of the file contains one key (empty lines and lines starting with a # are comments and ignored). Each line consists of the following fields, separated by spaces:

Key	Description
bits	Is the length of the key in bits.
comment	Not used for anything (but may be convenient for the user to identify the key).
exponent	Is a component used to identify and make up the key.
modulus	Is a component used to identify and make up the key.
options	Optional; its presence is determined by whether the line starts with a number or not (the option field never starts with a number.)

Note

Lines in this file are usually several hundred characters long (because of the size of the RSA key modulus). You do not want to type them in; instead, copy the `IDENTITY.PUB` file and edit it. The options (if present) consists of comma-separated option specifications. No spaces are permitted, except within double quotes. Option names are case insensitive.

The following RSA key file AUTHORIZED_KEYS option specifications are supported:

`Allowforwardingport="`*`port list`*`"`

Can be followed by any number of port numbers, separated by spaces. Remote forwarding is allowed for those ports whose number matches one of the patterns.

You can use * as a wildcard entry for all ports.

You can use these formats ">x", "<x", and "x_y" to specify greater than, less than, or inclusive port range. By default, all port forwardings are allowed.

The quotes (" ") are required. The <> show a list. Do not use the < > in the specification. For example:

`allowforwardingport `*`"2,52,2043"`*

`Allowforwardingto="`*`hostname and port list`*`"`

Can be followed by any number of hostname and port number patterns, separated by spaces. A port number pattern is separated from a hostname pattern by a colon. For example: *`hostname:port`*

Forwardings from the client are allowed to those hosts and port pairs whose name and port number match one of the patterns.

You can use '*' and '?' as wildcards in the patterns for host names. Normal name servers are used to map the client's host into a fully-qualified host name. If the name cannot be mapped, its IP address is used as the hostname.

You can use '*' as a wildcard entry for all ports.

You can use these formats '>x', '<x', and 'x_y' to specify greater than, less than, or inclusive port range. By default, all port forwardings are allowed.

`command="`*`command`*`"`

Specifies the command to be executed whenever this key is used for authentication. The user-supplied command (if any) is ignored. You may include a quote in the command by surrounding it with a backslash (\). Use this option to restrict certain RSA keys to perform just a specific operation. An example might be a key that permits remote backups but nothing else. Notice that the client may specify TCP/IP and/or X11 forwardings unless they are prohibited explicitly.

`Denyforwardingport="`*`port list`*`"`

Can be followed by any number of port numbers, separated by spaces. Remote forwardings are disallowed for those ports whose number matches one of the patterns.

You can use '*' as a wildcard entry for all ports.

You can use these formats '>x', '<x', and 'x_x' to specify greater than, less than, or inclusive port range.

`Denyforwardingto="`*`hostname port list`*`"`

Can be followed by any number of hostname and port number patterns, separated by spaces. A port number pattern is separated from a hostname by a colon. For example: hostname:port number pattern

Forwardings from the client are disallowed to those hosts and port pairs whose name and port number match one of the patterns.

You can use ‘*’ and ‘?’ as wildcards in the patterns for host names. Normal name servers are used to map the client’s host into a fully-qualified host name. If the name cannot be mapped, its IP address is used as a host name.

You can use ‘*’ as a wildcard entry for all ports.

You can use these formats ‘>x’, ‘<x’, and ‘x_x’ to specify greater than, less than, or inclusive port range.

`from=”`*`pattern-list`*`”`

In addition to RSA authentication, specifies that the fully-qualified name of the remote host must be present in the comma-separated list of patterns. You can use ‘*’ and ‘?’ as wildcards.

The list may contain patterns negated by prefixing them with ‘!’; if the fully-qualified host name matches a negated pattern, the key is not accepted.

This option increases security. RSA authentication by itself does not trust the network or name servers (but the key). However, if somebody steals the key, the key permits login from anywhere in the world. This option makes using a stolen key more difficult because the name servers and/or routers would have to be comprised in addition to just the key.

`idle-timeout=`*`time`*

Sets the idle timeout limit to a time in seconds (s or nothing after the number), in minutes (m), in hours (h), in days (d), or in weeks (w). If the connection has been idle (all channels) for that time, the process is terminated and the connection is closed.

`no-agent-forwarding`

Forbids authentication agent forwarding when used for authentication.

`no-port-forwarding`

Forbids TCP/IP forwarding when used for authentication. Any port forward requests by the client will return an error. For example, this might be used in connection with the command option.

`no-X11-forwarding`

Forbids X11 forwarding when used for authentication. Any X11 forward requests by the client will return an error.

Example 15.1. RSA Key File Examples

```
1024 33 12121...312314325 ylo@foo.bar
from="*.emptybits.com,!sluf.psccos.com"

1024 35 23...2334 ylo@niksula
command="dir *.txt",no-port-forwarding

1024 33 23...2323 xxxxx.acme.com
allowforwardingport="localhost:80"

1024 35 23...2334 www@localhost
```

15.12.1. SSH Port Forwarding and OpenVMS Captive Users

SSH implements a user group, known internally by SSH, which designates the users of captive accounts. This group, `IP$SSH_CAPTIVE_USERS`, gives the system administrator a method by which to specify captive users in various aspects of SSH configuration without requiring definition and management of OpenVMS rights identifiers. Additionally, the supplied SSH configuration template, `SSHD2_CONFIG.TEMPLATE`, disables port forwarding for captive users by default.

To enable the SSH port forwarding feature for captive users, remove the line `“DenyTcpForwardingForGroups”` from the `SSHD2_CONFIG.CONF`, which can be found in `SYS$SPECIFIC:[IP.CONFIG.SSH2]`.

15.13. SSH_KNOWN_HOSTS File Format

The `IP$:SSH_KNOWN_HOSTS` and `SYS$LOGIN:[.SSH]KNOWN_HOSTS` files contain host publickeys for all known hosts. The global file should be prepared by the administrator (optional), and the per-user file is maintained automatically; whenever the user connects an unknown host its key is added to the per-user file. Each line in these files contains the following fields: hostnames, bits, exponent, modulus, comment. The fields are separated by spaces.

Hostnames is a comma-separated list of patterns (* and ? act as wildcards). Each pattern is matched against the fully-qualified host names (when authenticating a client) or against the user-supplied name (when authenticating a server). A pattern may be preceded by '!' to indicate negation; if the hostname matches a negated pattern, it is not accepted (by that line) even if it matched another pattern on the line.

Bits, exponent, and modulus are taken directly from the hostkey. They can be obtained from `IP $:SSH_HOST_KEY.PUB`. The optional comment field continues to the end of the line, and is not used. Lines starting with # and empty lines are ignored as comments. When performing host authentication, authentication is accepted if any matching line has the proper key.

It is permissible (but not recommended) to have several lines or different hostkeys for the same names. This happens when short forms of host names from different domains are put in the file. It is possible that the files contain conflicting information. Authentication is accepted if valid information can be found from either file.

Note

The lines in these files are hundreds of characters long. Instead of typing in the hostkeys, generate them by a script or by copying `IP$:SSH_HOST_KEY.PUB` and adding the host names at the front.

Example

```
bos,bos.example.com,...,10.0.0.41
1024  37  159...93 bos.example.com
```

15.14. SSH Logicals

These logicals are used with the SSH server in the system logical name table.

```
$ SHOW LOGICAL/SYSTEM *SSH*
```

`SSH_DIR`

Points to the directory where the SSH1 configuration, master server log file, and hostkey files are kept. Normally, this is `IP$SPECIFIC_ROOT:[CONFIG]`. It is defined in `START_SSH.COM`.

`SSH_EXE`

Points to the directory where SSH executables are kept. Normally, this is `IP$COMMON_ROOT:[IP]`. It is defined in `START_SSH.COM`.

`SSH_LOG`

Points to the directory where the log files are kept. Normally, this is `IP$SPECIFIC_ROOT:[IP.LOG]`. It is defined in `START_SSH.COM`.

`SSH_TERM_MBX`

Mailbox used by SSHD_MASTER to receive termination messages from SSHD daemon processes. **Do not change this logical name.** This is created by the SSHD_MASTER process.

`IP$SSH_ACC_REJ_LOG_FILE`

If the user has set a log file to log connection accept and reject messages, this logical will be defined and will provide the name of the log file. This logical is set by using the **SET LOG-FILE** keyword in **IP CONFIGURE /SERVER**, and should not be modified directly by the user.

`IP$SSH_ALLOW_EXPIRED_PW`

Allows logging in to an account when the account's password has expired due to pwdlifetime elapsing. This applies to all users and circumvents normal OpenVMS expired-password checking, and therefore should be used with caution. An entry is made into the `SSH_LOG:SSHD.LOG` file when access is allowed using this logical name.

When access is allowed by way of this logical, the logical name table `LNM$SSH_LOGICALS` contains a logical name constructed as `IP$SSH_`*`pid`*`_PWDEXP` (where *`pid`* is the PID for the process). The system manager can use this to execute, for example, the DCL **SET PASSWORD** command in the site `SYLOGIN.COM` file.

`IP$SSH_ALLOW_PREEXPIRED_PW`

Allows logging in to an account when the password has been pre-expired. This applies to all users and circumvents normal OpenVMS expired-password checking, and therefore should be used with caution. An entry is made into the `SSH_LOG:SSHD.LOG` file when access is allowed using this logical name.

When access is allowed by way of this logical, the logical name table `LNM$SSH_LOGICALS` contains a logical name constructed as `IP$SSH_`*`pid`*`_PWDEXP` (where *`pid`* is the PID for the process). The system manager can use this to execute, for example, the DCL **SET PASSWORD** command in the site `SYLOGIN.COM` file.

`IP$SSH_DISPLAY_SYS$ANNOUNCE`

The SSH v1 protocol does not allow for the display of SYS$ANNOUNCE prior to logging in. If this logical is set, the contents of SYS$ANNOUNCE is displayed immediately after successful authentication and prior to the display of the contents of SYS$WELCOME.

`IP$SSH_ENABLE_SSH1_CONNECTIONS`

Set by the VSI TCP/IP master server process to enable SSH V1 sessions.

`IP$SSH_KEYGEN_MIN_PW_LEN`

Defines the minimum passphrase length when one is to be set in SSHKEYGEN. If not defined, defaults to zero.

`IP$SSH_LOG_ACCEPTS`

When set, causes the server to log successful connection requests as either an OPCOM message or a line in a log file. Specified by the **SET LOG-ACCEPT** command in **IP CONFIGURE / SERVER**. Note that the server does not use the information set in the ACCEPT-HOSTS keyword in **CONFIGURE /SERVER**. Rather, it uses the AllowHosts and DenyHosts keywords in the SSH server configuration file. Also, a successful connection request doesn't equate to a successful authentication request. This logical should not be modified directly by the user.

`IP$SSH_LOG_MBX`

Points to the OpenVMS mailbox used to log connection accept and reject messages. This must not be modified by the user.

`IP$SSH_LOG_REJECTS`

When set, causes the server to log rejected connection requests as either an OPCOM message or a line in a log file. Specified by the **SET LOG-REJECT** command in **IP CONFIGURE /SERVER**. Note that the server does not use the information set in the REJECT-HOSTS keyword in **CONFIGURE / SERVER**. Rather, it uses the AllowHosts and DenyHosts keywords in the SSH server configuration file. This logical should not be modified directly by the user.

`IP$SSH_MAX_SESSIONS`

Set this to the maximum number of concurrent SSH sessions you want to allow on the server system. If `IP$SSH_MAX_SESSIONS` is not defined, the default is 1000. Setting `IP$SSH_MAX_SESSIONS` to zero (0) causes an error. The value must be between 1 and 1000. The suggested place to set this is in `START_SSH.COM`. You must restart SSH for these changes to take effect.

`IP$SSH_PARAMETERS_`*`n`*

These values are set by VSI TCP/IP and must not be modified by the user.

`IP$SSH_USE_SYSGEN_LGI`

If defined, causes SSHD to use the OpenVMS SYSGEN value of LGI_PWD_TMO to set the login grace time, overriding anything specified in the command line or the configuration file.

15.15. Configuring the Secure Shell (SSH) 2 Server

This section describes how to configure and maintain the VSI TCP/IP Secure Shell (SSH) server v2.

15.15.1. Servers and Clients

A VSI TCP/IP SSH server is an OpenVMS system that acts as a host for executing interactive commands or for conducting an interactive session. The server software consists of two pieces of software (for future reference, "SSHD" will refer to both SSHD_MASTER and SSHD, unless otherwise specified):

- SSHD_MASTER, recognizes the differences between SSH v1 and SSH v2 and starts the appropriate server. If the request is for SSH v1, then the existing SSH v1 server is run; if the request is for SSH v2, then the SSH v2 server is run.
- SSHD, a copy of which is spawned for each connection instance. SSHD handles all the interaction with the SSH client.

A client is any system that accesses the server. A client program (SSH) is provided with VSI TCP/IP, but any SSH client that uses SSH version 2 protocol may be used to access the server. Examples of such programs are VSI TCP/IP SSH, TCPware SSH, puTTY, SecureCRT, and other SSH programs on UNIX-based systems.

Each host has a key using DSA encryption and is usually 1024 bits long (although, the user may create a different-sized key, if desired). The same key may be used on multiple machines. For example, each machine in a OpenVMScluster could use the same key.

When a client connects to the SSHD daemon:

- The client and server together, using the Diffie-Hellman key-exchange method, determine a 256-bit random number to use as the "session key". This key is used to encrypt all further communications in the session.
- Note that this key may be renegotiated between the client and the server on a periodic basis by including the RekeyIntervalSeconds keyword in the server configuration file (SSH2_DIR:SSHD2_CONFIG). This is desirable because during long sessions, the more data that is exchanged using the same encryption key, the more likely it is that an attacker who is watching the encrypted traffic could deduce the session key.
- The server informs the client which encryption methods it supports. See the description of the CIPHERS configuration keyword for the encryption methods supported.
- The client selects the encryption algorithm from those offered by the server.
- The client and the server then enter a user authentication dialog. The server informs the client which authentication methods it supports, and the client then attempts to authenticate the user by using some or all of the authentication methods.

The following authentication algorithms are supported:

- public-key (DSA keys)
- host-based
- password keyboard-interactive
- Kerberos V5 (password, kerberos-tgt, kerberos-1, kerberos-tgt-1, kerberos-2, kerberos-tgt-2)
- Certificate

System security is not improved unless the RLOGIN and RSHELL services are disabled.

If the client authenticates itself successfully, a dialog is entered for preparing the session. At this time the client may request things like:

- forwarding X11 connections
- forwarding TCP/IP connections
- forwarding the authentication agent connection over the secure channel

Finally, the client either requests an interactive session or execution of a command. The client and the server enter session mode. In this mode, either the client or the server may send data at any time, and such data is forwarded to/from the virtual terminal or command on the server side, and the user terminal in the client side. When the user program terminates and all forwarded X11 and other connections have been closed, the server sends command exit status to the client, and both sides exit.

15.15.2. Expired Password Handling

The SSH2 server supports expired password changing for interactive accounts without the CAPTIVE or RESTRICTED flags set and, via the DCL **SET PASSWORD** command. When an expired password is detected, the server will behave as if a **SET PASSWORD** command was specified by the user as a remotely-executed command (e.g., $ ssh foo set password), and the user will be logged out after changing the password. The user may then log in again using the changed password.

For CAPTIVE or RESTRICTED accounts, or for those accounts where LGICMD is set in the UAF record, the scenario is different. In these cases, the server can't directly execute **SET PASSWORD** command, because the command procedure specified in the LGICMD field of the UAF record will override the SSH server attempting to do a **SET PASSWORD** command. For these types of accounts, the system manager and/or user can use the value of the LOGIN_FLAGS for the process (normal interactive sessions may also examine these flags). For SSH logins, these flags will reflect:

- new mail has been received (`JPI$M_NEW_MAIL_AT_LOGIN`)
- the password is about to expire (`JPI$M_PASSWORD_WARNING`)
- the password has expired (`JPI$M_PASSWORD_EXPIRED`)

The DCL lexical function F$GETJPI may be used to examine these flags, as can the $GETJPI(W) system service or LIB$GETJPI RTL function. When an expired password value is detected, the user may then execute a **SET PASSWORD** command in the command procedure run for the account.

For example:

```
$!
$! Login_flags:
$!    1 = new mail messages waiting (JPI$M_NEW_MAIL_AT_LOGIN)
$!    4 = password expired during login (JPI$M_PASSWORD_EXPIRED)
$!    5 = password expires within 5 days (JPI$M_PASSWORD_WARNING)
$!
$ flags = f$getjpi("", "LOGIN_FLAGS")
$ new_flags = (flags/2)*2
$ if new_flags .ne. flags then write sys$output "New mail waiting"
$!
$! Note - new_flags is used below because it has the NEW_MAIL_AT_LOGIN$
$!   bit stripped.  The rest of the possible values are all
$!   discrete; i.e., you can't have combinations of them at the
$!   same time.
$!
$ if new_flags .eq. 4 then write sys$output "Password expired during login"
$ if new_flags .eq. 5 then write sys$output "Password expires within 5
 days"
$!
```

When an account in the SYSUAF has an expired password and the system syslogin.com or user's login.com has a **SET TERM** command, a warning message will be displayed prior to prompting to change the password as shown in the following example:

```
Your password has expired; you must set a new password to log in
% SET-W-NOTSET, error modifying DKA0:
-SET-E-INVDEV, device is invalid for requested operation
```

Old password:

The way to suppress these warning messages would be to check for the appropriate login flag, ignoring any **SET TERM** commands. For example:

```
$ flags = $getjpi(“”, “LOGIN_FLAGS”)
$ new_flags = (flags/2)*2
$ if new_flags.eq.4 then goto skip_the_inquiry
```

15.15.3. Break-In and Intrusion Detection

Care must be exercised when configuring the SSH clients and server to minimize problems due to intrusion records created by OpenVMS security auditing. The SSH user should consult the system manager to determine the authentication methods offered by the SSH server. The client should then be configured to not attempt any authentication method that is not offered by the server.

If a client attempts authentication methods not offered by the server, the OpenVMS security auditing system may log several intrusion records for each attempt to create a session to that server. The result being that the user could be locked out and prevented from accessing the server system without intervention from the server's system manager.

The authentication methods to be offered by the server are determined by the configuration keywords AllowedAuthentications and RequiredAuthentications. The number of intrusion records to be logged for any attempted SSH session is determined by the StrictIntrusionLogging configuration keyword.

When StrictIntrusionLogging is set to YES (the default), each method that is tried and fails causes an intrusion record to be logged. The following rules apply:

- When HostBased or PublicKey authentications are attempted and fail, one intrusion record is logged for each failed method.
- When password authentication is attempted, one intrusion record is logged for each failed password.

Example 1:

The server is set up to allow HostBased and password authentication; also, up to three password attempts are allowed. If all methods fail, four intrusion records are logged:

1 for the failed HostBased

3 for the failed password attempts, one per attempt

When StrictIntrusionLogging is set to NO, it has the effect of relaxing the number of intrusions logged. Overall failure of all authentication methods simply counts as a single failure, except for password authentication. The following rules apply:

- When password authentication is attempted, one intrusion record is logged for each failed password.
- When any of HostBased or PublicKey authentication fails, and password authentication is not attempted, exactly one intrusion record is logged, as opposed to one for each failed method.
- When any of HostBased or PublicKey authentication fails, but password authentication is attempted and succeeds, the only intrusion record(s) logged is one for each failed password attempt.

Example 2:

The server is set up to allow HostBased and password authentication; also, up to three password attempts are allowed. If all methods fail, three intrusion records are logged:

0 for the failed HostBased

3 for the failed password attempts, one per attempt

Example 3:

The server is set up to allow HostBased and password authentication; also, up to three password attempts are allowed. HostBased and RSA fail, but password authentication is successful after 1 failed password. Therefore, one intrusion record is logged:

0 for the failed HostBased

1 for the failed password attempt

Example 4:

The server is set up to allow HostBased and PublicKey authentication, but not password authentication. If all methods fail, one intrusion record is logged.

Example 5:

The server is set up to allow HostBased and PublicKey authentication, but not password authentication. HostBased authentication fails, but PublicKey succeeds. No intrusion records are logged.

15.15.4. Configuring SSHD Master

SSHD Master is configured using the **IP CONFIGURE /SERVER** command, selecting SSH, and using the following parameters:

Note

The only supported methods to start SSHD Master are to restart the VSI TCP/IP server if SSH is not currently running, or to use the **IP NETCONTROL SSH START** command. All of these options are set using **IP CONFIGURE /SERVER**, and modifying the SSH service.

bits *n*

Specifies the number of bits in the server key. The default is 768.

debug *debug-level*

Disables or enables debugging levels. Values are between 0 and 50. Zero (0) disables debugging and higher values turn on successively more debugging.

ipv4-disable

Disables the server from listening on an IPv4 socket.

ipv6-disable

Disables the server from listening on an IPv6 socket.

enable-ssh2

Enables SSH V2 sessions.

listen-address

Specify the IPV4 or IPV6 address on which to listen for connect request. This may be a valid IPV4 or IPV6 address, or ANY to listen on all addresses. If not specified, the default is to listen on all IPV4 and IPV6 addresses.

port *n*

Specifies the port on which the server listens for connections. The default is 22.

quiet_mode

Specifies that nothing is sent to the system log. Normally, the beginning, authentication, and termination of each connection is logged.

ssh2-config-file *filename*

Specifies the name of the configuration file. The default is `SSH2_DIR:SSHD2_CONFIG`.

verbose

Specifies that verbose message logging will be performed by SSHD Master.

15.15.5. SSHD2 Configuration File

SSHD reads configuration data from its configuration file. By default, this file is `SSH2_DIR:SSHD2_CONFIG`. The file contains keyword value pairs, one per line. Lines starting with # and empty lines are interpreted as comments. The following keywords are possible. Keywords are case insensitive.

Keyword	Value	Default	Description
AllowedAuthentications	List	Publickey, Password	Permitted techniques. Valid values are: Keyboard-interactive, password, public-key, hostbased, kerberos-1, kerberos-tgt-1, kerberos-2, kerberos-tgt-2 Along withRequiredAuthentications, the system administrator can force the users to complete several authentications before they are considered authenticated.
AllowedPasswordAuthentications	List	kerberos, local	Specifies the different password authentication schemes that are allowed. Only kerberos and local are acceptable.
AllowGroups	List		Access control by UAF rights list entries
AllowHosts	Host list		Access control by hostname
AllowShosts	Host list		Access control by hostname
AllowTcpForwarding	Y/N	Y	Enable TCP port forwarding
AllowTcpForwardingForUsers	User list		Per-User forwarding
AllowTcpForwardingForGroups	Rights list		Per-Rights list ID forwarding
AllowUsers	User list		Access control by username
AllowX11Forwarding	Y/N	Y	Enable X11 forwarding
AuthInteractiveFailureTimeout	Seconds	2	Delay, in seconds, that the server delays after a failed attempt to log in using keyboard-interactive and password authentication.
AuthKbdInt.NumOptional	Number	0	Specifies how many optional submethods

Keyword	Value	Default	Description
			must be passed before the authentication is considered a success. (Note that all reported submethods must always be passed.) See AuthKbdInt.Optional for specifying optional submethods, and AuthKbdInt.Required for required submethods. The default is 0, although if no required submethods are specified, the client must always pass at least one optional submethod.
AuthKbdint.Optional	List	None	Specifies the optional submethods keyboard-interactive will use. Currently only the submethod password is defined. AuthKbdInt.NumOptional specifies how many optional submethods must be passed. The keyboard-interactive authentication method is considered a success when the specified amount of optional submethods and all required submethods are passed.
AuthKbdInt.Required			Specifies the required submethods that must be passed before the keyboard-interactive authentication method can succeed.
AuthKbdInt.Retries	Number	3	Specifies how many times the user can retry keyboard-interactive.
AuthorizationFile	Filename	Authorization	Authorization file for publickey authentication.
AuthPublicKey.MaxSize	Number	0	Specifies the maximum size of a publickey that can be used to log in.

Keyword	Value	Default	Description
			Value of 0 disables the check.
AuthPublicKey.MinSize	Number	0	Specifies the minimum size of a publickey that can be used to log in. Value of 0.
Cert.RSA.Compat.HashScheme	md5 or sha	md5	Previous clients and servers may use hashes in RSA certificates incoherently (sometimes SHA-1 and sometimes MD5). This specifies the hash used when a signature is sent to old versions during the initial key exchanges.
BannerMessageFile	Filename	SYS $ANNOUNCE	Message sent to the client before authentication begins.
CheckMail	Y/N	Y	Display information about new mail messages when logging in
Ciphers	Cipher list		Encryption ciphers offered
DenyGroups	Rights list		Deny access for UAF rightslist identifiers
DenyHosts	Host list		Deny access for hosts
DenySHosts	Host list		Deny access for hosts
DenyTcpForwardingForUsers	User list		Forbid forwarding for listed users
DenyTcpForwardingForGroups	Rights list		Forbid forwarding for listed rightslist names
DenyUsers	User list		Access control by username
FascistLogging	Y/N	Y	Verbose logging
ForwardACL	Pattern	None	With this option, you can have more fine-grained control over what the client is allowed to forward, and to where. See Section 15.15.5.4
ForwardAgent	Y/N	Y	Enable agent forwarding
HostCA	Certificate	None	Specifies the CA certificate (in binary or PEM (base64) format) to be used when

Keyword	Value	Default	Description
			authenticating remote hosts. The certificate received from the host must be issued by the specified CA and must contain a correct alternate name of type DNS (FQDN). If no CA certificates are specified in the configuration file, the protocol tries to do key exchange with ordinary publickeys. Otherwise, certificates are preferred. Multiple CAs are permitted.
HostCANoCRLs	Certificate	None	Similar to HostCA, but disables CRL checking for the given CA certificate.
HostCertificateFile	Filename	None	This keyword works very much like PublicHostKeyFile, except that the file is assumed to contain an X.509 certificate in binary format. The keyword must be paired with a corresponding HostKeyFileoption. If multiple certificates with the same publickey type (DSS or RSA) are specified, only the first one is used.
HostbasedAuthForceClient HostnameDNSMatch	Y/N	N	Host name given by client.
Hostkeyfile	Filename	Hostkey	Hostkey filename
HostSpecificConfig	Pattern	None	Specifies a subconfiguration file for this server, based on the hostname of the client system.
IdentityFile	Filename	Identification	Identity filename
IdleTimeout	Time	0 = none	Set idle timeout (in seconds)

Keyword	**Value**	**Default**	**Description**
IgnoreRhosts	Y/N	N	Do not use rhosts and shosts for hostbased authentication for all users
IgnoreRootRhosts	Y/N	Y	Don't use rhosts and shosts files for authentication of SYSTEM
KeepAlive	Y/N	Y	Send keepalives
LdapServers	Server URL	None	Specified asldap:// server.domain-name:389 CRLs are automatically retrieved from the CRL distribution point defined in the certificate to be checked if the point exists. Otherwise, the comma-separated server list given by option LdapServersis used. If intermediate CA certificates are needed in certificate validity checking, this option must be used or retrieving the certificates will fail.
ListenAddress	IP address	0.0.0.0	Listen on given interface
Macs	Algorithm		Select MAC (Message Authentication Code) algorithm
MapFile	Filename	None	This keyword specifies a mapping file for the preceding Pkikeyword. Multiple mapping files are permitted per one Pkikeyword. The mapping file format is described below.
MaxBroadcastsPerSecond	#broadcasts	0	Listen for UDP broadcasts
NoDelay	Y/N	N	Enable Nagel Algorithm
PasswordAuthentication	Y/N	Y	Permit password authentication
PasswordGuesses	#guesses	3	Limit number of password tries to specified number

Keyword	Value	Default	Description
PermitEmptyPasswords	Y/N	N	Permit empty (blank) passwords
PermitRootLogin	Y/N	N	SYSTEM can log in
Pki	Filename	None	This keyword enables user authentication using certificates. CA-certificate must be an X.509 certificate in binary format. This keyword must be followed by one or more MapFile keywords. The validity of a received certificate is checked separately using each of the defined Pkikeywords in turn until they are exhausted (in which case the authentication fails), or a positive result is achieved. If the certificate is valid, the mapping files are examined to determine whether the certificate allows the user to log in. A correct signature generated by a matching private key is always required.
PkiDisableCrls	Y/N	Y	This keyword disables CRL checking for the Pkikeyword, if argument is Y.
PrintMotd	Y/N	Y	Display SYS $WELCOME when logging in
PublicHostKeyFile	Filename	Hostkey.pub	Hostkey file location
QuietMode	Y/N	N	Quiet mode
RandomSeedFile	Filename	Random_seed	Random seed file
RekeyIntervalSeconds	# seconds	0	Frequency of rekeying
RequiredAuthentication	Authentication list		Authentications client must support
RequireReverseMapping	Y/N	N	Remote IP address must map to hostname
ResolveClientHostName	Y/N	Y	Controls whether the server will try to resolve the client IP address

Keyword	Value	Default	Description
			at all, or not. This is useful when you know that the DNS cannot be reached, and the query would cause additional delay in logging in. Note that if you set this to N, you should not set RequireReverseMappingto Y.
RSAAuthentication	Y/N	Y	Enable RSA authentication
SendKeyGuess	Y/N	Y	This parameter controls whether the server will try to guess connection parameters during key exchange, or not. Some clients do not support key exchange guesses and may fail when they are present.
SftpSysLogFacility	log facility	None	Defines the log facility the SFTP server will use
StrictIntrusionLogging	Y/N	Y	Determine how intrusion records are created by failed authentication attempts.
StrictModes	Y/N	N	Strict checking for directory and file protection.
SyslogFacility	Facility	AUTH	Defines what log facility to be used when logging server messages.
Terminal.AllowUsers	pattern	All users	List users that are allowed terminal (interactive) access to the server.
Terminal.DenyUsers	pattern	None	List users that are denied terminal (interactive) access to the server.
Terminal.AllowGroups	pattern	All groups	Similar to Terminal.AllowUsers but matches groups instead of usernames.
Terminal.DenyGroups	pattern	None	Similar to Terminal.DenyUsers but matches groups instead of usernames

Keyword	Value	Default	Description
UserConfigDirectory	Directory	SYS$LOGIN:	Location of user SSH2 directories
UserKnownHosts	Y/N	Y	Respect user [.ssh2] known hosts keys
UserSpecificConfig	Pattern	None	Specifies a subconfiguration file for this server, based on user logging in.
VerboseMode	Y/N	N	Verbose mode

The keywords **/MAC** and **/CIPHER** have discrete values, plus there are values that actually denote a grouping of 2 or more of the discrete values. Each of these values may be put in the configuration file (SSH2_DIR:SSHD2_CONFIG).

MACs	Discrete values: hmac-sha1, hmac-sha256, hmac-md5, hmac-ripemd160, none
	Group ANYMAC consists of: hmac-sha1, hmac-sha256, hmac-md5, hmac-ripemd160
	Group ANY consists of: hmac-sha1, hmac-sha256, hmac-md5, hmac-ripemd160, none
	Group ANYSTD consists of: hmac-sha1, hmac-md5, none
	Group ANYSTDMAC consists of: hmac-sha1, hmac-md5
Ciphers	Discrete values: 3des, aes, blowfish, aes128-ctr, aes128-cbc, aes192-ctr, aes192-cbc, aes256-ctr, aes256-cbc, 3des-ctr, 3des-cbc, blowfish-ctr, blowfish-cbc, des-cbc, rc2-cbc, none
	Group ANYSTDCIPHER consists of: aes128-ctr, aes128-cbc, aes192-ctr, aes192-cbc, aes256-ctr, aes256-cbc, 3des-ctr, 3des-cbc, blowfish-ctr, blowfish-cbc
	Group ANY consists of: aes128-ctr, aes128-cbc, aes192-ctr, aes192-cbc, aes256-ctr, aes256-cbc, 3des-ctr, 3des-cbc, blowfish-ctr, blowfish-cbc, des-cbc, rc2-cbc, none
	Group ANYCIPHER consists of: aes128-cbc, 3des-cbc, twofish128-cbc, cast128-cbc, twofish-cbc, blowfish-cbc, aes192-cbc, aes256-cbc, twofish192-cbc, twofish256-cbc, arcfour, des-cbc, rc2-cbc
	Group ANYSTD consists of:

	aes128-cbc, 3des-cbc, twofish128-cbc, cast128-cbc, twofish-cbc, blowfish-cbc, aes192-cbc, aes256-cbc, twofish192-cbc, twofish256-cbc, arcfour, none

A discrete value or a group identifier may be used with MACS and CIPHERS. For example, in the configuration file, the following examples could be used:

Ciphers ANYCIPHER

Ciphers 3des,aes128-cbc

MACs ANYMAC

MACs hmac-sha1

Aliases may be used for some standard ciphers:

Alias	**Value**
aes	aes128-cbc
3des	3des-cbc
blowfish	blowfish-cbc

15.15.5.1. HostSpecificConfig Notes:

The global server file (SSH2_DIR:SSHD2_CONFIG) now can use the keyword HostSpecificConfig to allow the specification of a configuration file based on the client system. These lines are specified as:

HostSpecificConfig *hostname subconfig-file*

*hostname*will be used to match the client host, as specified under option AllowHosts. The file *subconfig-file* will then be read, and configuration data amended accordingly. The file is read before any actual protocol transactions begin, and you can specify most of the options allowed in the main configuration file. You can specify more than one subconfiguration file, in which case the patterns are matched and the files read in the order specified. Later defined values of configuration options will either override or amend the previous value, depending on which option it is. The effect of redefining an option is described in the documentation for that option. For example, setting Ciphers in the subconfiguration file will override the old value, but setting AllowUsers will amend the value.

The *subconfig-file* will be assumed by default to exist in the SSH2_DIR directory. However, this may be overridden by specifying a complete directory/file specification. For example:

HostSpecificConfig bos.example.com dka0:[sshconfigs]bosconfig.dat

HostSpecificConfig clt.example.com cltconfig.dat

In the first instance, an incoming connection from bos.example.com will use the subconfig file dka0:[sshconfigs]bosconfig.dat. In the second example, an incoming connection from clt.example.com will use ssh2_dir:cltconfig.dat.

Unlike ssh2_config, the subconfig files may have configuration blocks, or stanzas, in them. They are used per-host. The subconfiguration heading is interpreted identically to what is described above (i.e, with UserSpecificConfig, the pattern is of the format “hostname”.)

Note

If the subconfig file cannot be found or cannot be parsed successfully for any reason, access to the system will be denied for the system to which the subconfig file applies.

15.15.5.2. UserSpecificConfig Notes:

The global server file (SSH2_DIR:SSHD2_CONFIG) can use the keyword UserSpecificConfig to allow the specification of a configuration file based on the username of the user who's logging into the server. These keywords are of the form:

UserSpecificConfig user[%group] [@host] subconfig-file

*pattern*will be used to match the username, as specified under the option AllowUsers. The file *subconfig-file* will then be read, and configuration data amended accordingly. The file is read before any actual protocol transactions begin, and you can specify most of the options allowed in the main configuration file. You can specify more than one subconfiguration file, in which case the patterns are matched and the files read in the order specified. Later defined values of configuration options will either override or amend the previous value, depending on which option it is. The effect of redefining an option is described in the documentation for that option. For example, setting Ciphers in the subconfiguration file will override the old value, but setting AllowUsers will amend the value.

Unlike sshd2_config, the subconfig files may have configuration blocks, or stanzas, in them. They are used per user. The subconfiguration heading is interpreted identically to what is described above (i.e., with UserSpecificConfig, the pattern is of the format user[%group] [@host].

The subconfig-file will be assumed by default to exist in the SSH2_DIR directory. However, this may be overridden by specifying a complete directory/file specification. For example:

UserSpecificConfig dilbert dka0:[sshconfigs]dilbert.dat

UserSpecificConfig boss@lima.beans.com pointyhair.dat

In the first instance, an incoming connection for user alice will use the subconfig file dka0:[sshconfigs]alice.dat. In the second example, an incoming connection from user bob at system lima.beans.com will use ssh2_dir:bob.dat.

Note

If the subconfig file cannot be found or cannot be parsed successfully for any reason, access to the system will be denied for the user to which the subconfig file applies.

15.15.5.3. KEYBOARD-INTERACTIVE Notes:

At this point, KEYBOARD-INTERACTIVE mode is simply another form of password authentication. The user won't notice anything different with this mode. In the future, VSI may implement items such as system passwords, secondary passwords, and true OpenVMS-style password changing using this authentication method. As other clients support the use of the KEYBOARD-INTERACTIVE authentication method for doing password authentication (without using any external callouts from the mechanism such as SecureID cards), the server should support those clients.

15.15.5.4. ForwardACL Notes

With this option, you can have more fine-grained control over what the client is allowed to forward, and to where. Format for this option is:

`[allow|deny] [local|remote]` *`user-pat forward-pat`* `[`*`originator-pat`*`]`

`user-pat` will be used to match the client-user, as specified under the option UserSpecificConfig. *`forward-pat`* is a pattern of format *`host-id`*[%*`port`*]. This has different interpretations, depending on whether the ACL is specified for local or remote forwards. For local forwards, the *`host-id`* will match with the target host of the forwarding, as specified under the option AllowHosts. *`port`* will match with the target port. Also, if the client sent a host name, the IP address will be looked up from the DNS, which will be used to match the pattern. For remote forwardings, where the forward target is not known (the client handles that end of the connection); this will be used to match with the listen address specified by the user (and as such is not as usable as with local forwards). *`port`* will match the port the server is supposed to be listening to with this forward. With local forwards, *`originator-pat`* will match with the originator address that the client has reported. Remember, if you do not administer the client machine, users on that machine may use a modified copy of ssh that can be used to lie about the originator address. Also, with NATs (Network Address Translation), the originator address will not be meaningful (it will probably be an internal network address). Therefore, you should not rely on the originator address with local forwards, unless you know exactly what you are doing. With remote forwards, *`originator-pat`* will match with the IP address of the host connecting to the forwarded port. This will be valid information, as it is the server that is checking that information.

If you specify any allow directives, all fowards in that class (local or remote) not specifically allowed will be denied (note that local and remote forwards are separate in this respect, e.g., if you have one “allow remote” definition, local forwards are still allowed, pending other restrictions). If a forward matches with both allow and deny directives, the forwarding will be denied. Also, if you have specified any of the options [Allow.Deny]TcpForwardingForUsers.Groups] or AllowTcpForwarding, and the forwarding for the user is disabled with those, an allow directive will not re-enable the forwarding for the user. Forwarding is enabled by default.

15.15.5.5. MappingFileFormat

When certificates are used in user authentication, one or more mapping files determine whether the user can log to an account with a certificate. The mapping file must contain one or more lines in the following format:

account-id keyword arguments

Keyword must be one of the following: Email, EmailRegex, Subject, SerialAndIssuer, or SubjectRegex.

Arguments are different for each keyword. The following list describes each variation:

Email

arguments: an email address in standard format. If the certificate contains the email address as an alternate name, it is good for logging in as user account-id.

Subject

arguments: a subject name in DN notation (LDAP style). If the name matches the one in the certificate, the certificate is good for logging in as user account-id.

SerialAndIssuer

arguments: a number and an issuer name in DN notation (LDAP style), separated by whitespace. If the issuer name and serial number match those in the certificate, the certificate is good for logging in as user account-id.

EmailRegex

arguments: a regular expression (*egrep syntax*). If it matches an altername (of type email-address) in the certificate, the certificate is good for logging in as user account-id. As a special feature, if account-id contains a string %subst%, it is replaced by the first parenthesized substring of the regular expression before comparing it with the account the user is trying to log into.

SubjectRegex

Works identically to EmailRegex, except it matches the regular expression to the canonical subject name in the received certificate.

Empty lines and lines beginning with # are ignored.

EXAMPLE: MAPPINGFILE

guest email guest@domain.org

guest subject C=Fl,O=Company Ltd., CN-Guest User

guest SerialAndUser 123 C=Fl, O=Foo\Ltd., CN=Test CA

%subst% EmailRegex ([a-z]+)@domain.\org

%subst% Subjectregex ^C=Fl,O=Company,CN=([a-z]+)$

The example EmailRegexpermits in users with email addresses with domain domain.org and usernames that contain only letters, each user to the account that corresponds to the username part of the email address.

The example SubjectRegex lets in all users with fields C=Fl and O=Company in the subject name if their CN field contains only letters and is the account name they are trying to log into.

Note the ^ and $ at the beginning and end of the regular expression; they are required to prevent the regular expression from matching less than the whole string (subject name).

Note also that all characters interpreted by the regular expression parser as special characters must be escaped with a backslash if they are a part of the subject name. This also means that the backslash in the SerialAndIssuer example would have to be escaped with another backslash if the same subject name was used in a SubjectRegexrule.

15.15.6. Starting the SSH Server for the First Time

Follow these instructions for using SSH for the first time.

1. Use the **IP CONFIGURE /SERVER** command to enable the SSH v2 server.

```
$ IP CONFIGURE/SERVER
VSI TCP/IP Server Configuration Utility
[Reading in configuration from IP$:SERVICES.MASTER_SERVER]
SERVER-CONFIG>SHOW/FULL SSH
Service "SSH": ***DISABLED***
         INIT() = Merge_Image
         Program = "IP$:LOADABLE_SSH_CONTROL"
         Priority = 5
         Parameters = “enable-ssh1”
                      “enable-ssh2”
```

```
SERVER-CONFIG>ENABLE SSH
SERVER-CONFIG>EXIT
[Writing configuration to IP$COMMON_ROOT:[IP]SERVICES.MASTER_SERVER]
```

Note

The parameter enable-ssh2 must be set. If it is not set, SSH v2 sessions will not be accepted by the server.

2. Use SSHKEYGEN /SSH2 to generate an ssh2 key and to create the server keys in the SSH2_HOSTKEY_DIR directory:

   ```
   $ DEFINE IP$SSH2_HOSTKEY_DIR -
   _$ IP$SPECIFIC_ROOT:[IP.SSH2.HOSTKEYS]
   $ IP SSHKEYGEN /SSH2 /HOST
   Generating 1024-bit dsa key pair
     8 .oOo.oOoo.oO
   Key generated.
   1024-bit dsa, lillies@sfo.example.com, Mon Aug 04 2015 09:19:47
   Private key saved to ip$ssh2_hostkey_dir:hostkey.
   publickey saved to ip$ssh2_hostkey_dir:hostkey.pub
   ```

3. Copy the template server configuration file to the SSH2_DIR: directory renaming it SSHD2_CONFIG.:

   ```
   $ COPY IP$CONFIG:SSHD2_CONFIG.TEMPLATE IP$SPECIFIC_ROOT:
   [IP.SSH2]SSHD2_CONFIG
   ```

4. Copy the template client configuration file to the SSH2_DIR: directory renaming it SSH2_CONFIG.:

   ```
   $ COPY IP$SPECIFIC_ROOT:[IP.SSH2]SSH2_CONFIG.TEMPLATE -
   _$ IP$SPECIFIC_ROOT:[IP.SSH2]SSH2_CONFIG.
   ```

Note

As delivered, the template files provide a reasonably secure SSH environment. However, VSI recommends these files be examined and modified appropriately to reflect the security policies of your organization.

5. Restart VSI TCP/IP. This creates the SSH server process and defines the SSH logical names.

   ```
   $ @IP$:IP$SYSTARTUP.COM RESTART
   $ SHOW SYSTEM/PROCESS="IP$SSH_SERVER"
   OpenVMS V8.4-2L1  on node HNALOR   18-DEC-2017 13:09:33.58   Uptime  2
    19:01:55
     Pid    Process Name    State  Pri      I/O       CPU       Page flts
    Pages
   0000082E IP$SSH_SERVER   LEF      7      761   0 00:00:00.23      1344
     529
   $ SHOW LOGICAL/SYSTEM *SSH*
     "IP$SSH2_HOSTKEY_DIR" =
              "IP$SPECIFIC_ROOT:[IP.SSH2.HOSTKEYS]"
     "IP$SSH2_KNOWNHOSTS_DIR" =
              "IP$SPECIFIC_ROOT:[IP.SSH2.KNOWNHOSTS]"
     "IP$SSH_ENABLE_SSH2_CONNECTIONS" = "1"
   ```

```
    "SSH2_DIR" = "IP$SPECIFIC_ROOT:[IP.SSH2]"
    "SSH_DIR" = "IP$SPECIFIC_ROOT:[IP]"
    "SSH_EXE" = "IP$COMMON_ROOT:[IP]"
    "SSH_LOG" = "IP$SPECIFIC_ROOT:[IP.SSH]"
    "SSH_TERM_MBX" = "MBA36:"
```

15.15.7. Configuring the SSH2 Server on a OpenVMScluster with a Common System Disk

When configuring the SSH2 server on a OpenVMScluster with a common system disk, you must create the appropriate directories on all cluster nodes other than the one on which VSI TCP/IP was originally installed. Note that this does not need to be done for cluster members that do not share a common system disk.

The following procedure should be followed on each cluster node other than the cluster node on which VSI TCP/IP was originally installed:

- Create the necessary directories:

  ```
  $ CREATE/DIR IP$SPECIFIC_ROOT:[IP.SSH2]/PROT=(WO:RE,GR:RE)
  $ CREATE/DIR IP$SPECIFIC_ROOT:[IP.SSH2.KNOWNHOSTS]/PROT=(WO:R,GR:R)
  $ CREATE/DIR IP$SPECIFIC_ROOT:[IP.SSH2.HOSTKEYS]/PROT=(WO:R,GR:R)
  $ CREATE/DIR IP$SPECIFIC_ROOT:[IP.SSH]/PROT=(WO:RE,GR:RE)
  ```

- Edit the `IP$SPECIFIC_ROOT:[IP.SSH2]SSHD2_CONFIG` file as necessary. This may be copied from another cluster node, or it may be created fresh from the `SSHD2_CONFIG.TEMPLATE` file.
- Edit the `IP$SPECIFIC_ROOT:[IP.SSH2]SSH2_CONFIG` file as necessary. This may be copied from another cluster node, or it may be created fresh from the `SSH2_CONFIG.TEMPLATE` file.
- Configure the SSH2 server using **IP CONFIGURE/SERVER**
- Generate the SSH2 hostkeys using **IP SSHKEYGEN/SSH2/HOST**
- (Re)start SSHD Master using **IP NETCONTROL SSH RESTART**

15.15.8. Changing SSHD2 Configuration File After Enabling SSH2

If you make a change to the SSH server configuration file after you have enable SSH, you must restart SSH for these changes to take effect for subsequent new connections.

```
$ IP NETCONTROL SSH RESTART
```

Note

When issuing the **RESTART** command for SSH, all active SSH server sessions are terminated. Active client sessions are not affected.

15.15.9. Connection and Login Process

To create a session, SSHD does the following:

1. SSHD_MASTER sees the connection attempt. It creates an SSHD process, passing the operating parameters to it. SSHD performs validation for the user.
2. Assuming the login is successful, SSHD creates a pseudo terminal for the user (an FTAnn: device). This device is owned by the user attempting to log in.
3. SSHD creates an interactive process on the pseudo terminal, using the username, priority, and privileges of the user who is attempting to log in. If a command was specified, it is executed and the session is terminated.
4. SSH generates the file `SSHD.LOG` for each connection to the SSH server. Many connections result in many log files. Instead of purging the files on a regular basis, use the following DCL command to limit the number of versions:

   ```
   $ SET FILE /VERSION_LIMIT=x IP$LOG:SSHD.LOG
   ```

Note

The value for **/VERSION_LIMIT** must not be smaller than the maximum number of simultaneous SSH sessions anticipated. If the value is smaller, SSH users may be prevented from establishing sessions with the server.

15.15.9.1. FILES

`IP$:HOSTS.EQUIV`

Contains host names, one per line. This file is used during .rhosts authentication. Users on those hosts are permitted to log in without a password, provided they have the same username on both machines. The hostname may also be followed by a username. Such users are permitted to log in as any user on the remote machine (except SYSTEM). Additionally, the syntax +@group can be used to specify netgroups. Negated entries start with dash (-). If the client host/user is matched in this file, login is permitted provided the client and server usernames are the same. Successful RSA host authentication is required. This file should be world-readable but writable only by SYSTEM.

It is never a good idea to use usernames in hosts.equiv. It means the named user(s) can log in as anybody, which includes accounts that own critical programs and directories. Using a username grants the user SYSTEM access. The only valid use for usernames is in negative entries.

Note

This warning also applies to rshell/rlogin.

`IP$:SHOSTS.EQUIV`

Processed as `IP$:HOSTS.EQUIV`. May be useful in environments that want to run both rshell/rlogin and ssh.

`IP$SSH2_HOSTKEY_DIR:HOSTKEY`

Contains the private part of the hostkey. This file does not exist when VSI TCP/IP is installed. The SSH server starts only with this file. This file must be created manually using the command:

```
$ IP SSHKEYGEN /SSH2 /HOST.
```

This file should be owned by SYSTEM, readable only by SYSTEM, and not accessible to others.

To create a hostkey with a name that is different than what **SSHKEYGEN** creates, do one of the following:

- Generate with **SSHKEYGEN /SSH2 /HOST** and simply rename the file(s).
- Generate without the **/HOST** switch and then name the file(s) whatever you want.

By default the logical name SSH2_DIR points to the `IP$SPECIFIC_ROOT:[CONFIG.SSH2]` directory.

Refer to the *VSI TCP/IP User's Guide*, for more details about **SSHKEYGEN**.

`IP$SSH2_HOSTKEY_DIR:HOSTKEY.PUB`

Contains the public part of the hostkey. This file should be world-readable but writable only by SYSTEM. Its contents should match the private part. This file is not used for anything; it is only provided for the convenience of the user so its contents can be copied to known hosts files.

`SSH2:SSH_RANDOM_SEED`
`SYS$LOGIN:[.SSH]RANDOM_SEED`

Contains a seed for the random number generator. This file should only be accessible by system.

`SSH2_DIR:SSHD2_CONFIG`

Contains configuration data for the v2 SSHD server. This file should be writable by system only, but it is recommended (though not necessary) that it be world-readable.

`SYS$LOGIN:[.SSH2].SHOSTS`

Permits access using SSH2 only. For SSH2, this file is the same as for .rhosts. However, this file is not used by rlogin and rshell daemon.

`SYS$LOGIN:.RHOSTS`

This file contains host-username pairs, separated by a space, one per line. The given user on the corresponding host is permitted to log in without a password. The same file is used by rlogin and rshell. SSH2 differs from rlogin and rshell in that it requires RSA host authentication in addition to validating the hostname retrieved from domain name servers (unless compiled with the -with-rhosts configuration option). The file must be writable only by the user. It is recommended that it not be accessible by others. It is possible to use netgroups in the file. Either host or username may be of the form +@groupname to specify all hosts or all users in the group.

`SYS$LOGIN:[.SSH2]AUTHORIZATION`

This file contains information on how the server verifies the identity of a user.

`SYS$LOGIN:[.SSH2.KNOWNHOSTS]`*`xxxxyyyy`*`.pub`

These are the public hostkeys of hosts that a user wants to log in from using "hostbased" authentication (equivalent to the SSH1's RhostsRSAAuthentication). Also, a user must set up his/her individual `.SHOSTS` or `.RHOSTS` file. If the username is the same in both hosts, it is adequate to put the public hostkey in `SSH2_DIR:KNOWNHOSTS` and add the host's name to the system-wide `SHOSTS.EQUIV` or `RHOSTS.EQUIV` file.

`xxxx` is the hostname (FQDN) and *`yyyy`* denotes the publickey algorithm of the key (ssh-dss or ssh-rsa).

For example bos.example.com's hostkey algorithm is ssh-dss. The hostkey would then be `bos_example_com_ssh-dss.pub` in the `[.SSH2.KNOWNHOSTS]` directory.

15.15.10. SSH2 AUTHORIZATION File Format

The Authorization file contains information on how the server verifies the identity of a user. This file has the same general syntax as the SSH2 configuration files. The following keywords may be used:

Keyword	Description
KEY	The filename of a publickey in the `[.SSH2]` directory in the user's `SYS$LOGIN` directory. This key is used for identification when contacting the host. If there are multiple KEY lines, all are acceptable for login.
COMMAND	This keyword, if used, must follow the KEY keyword above. This is used to specify a "forced command" that executes on the server side instead of anything else when the user is authenticated. This option might be useful for restricting certain publickeys to perform certain operations.

15.15.11. SSH2 Logicals

These logicals are used with the SSH server in the system logical name table.

```
$ SHOW LOGICAL/SYSTEM *SSH*
```

`SSH_DIR`

Points to the directory where the master server log file is kept. Normally, this is `IP$SPECIFIC_ROOT:[IP]`. It is defined in `START_SSH.COM`.

`SSH_EXE`

Points to the directory where SSH executables are kept. Normally, this is `IP$COMMON_ROOT:[IP]`. It is defined in `START_SSH.COM`.

`SSH_LOG`

Points to the directory where the log files are kept. Normally, this is `IP$SPECIFIC_ROOT:[IP.SSH]`. It is defined in `START_SSH.COM`.

`IP$LOG_MBX`

Points to the OpenVMS mailbox used to log connection accept and reject messages. This must not be modified by the user.

`IP$SSH_ACC_REJ_LOG_FILE`

If the user has set a log file to log connection accept and reject messages, this logical will be defined and will provide the name of the log file. This logical is set by using the SET LOG-FILE keyword in **IP CONFIGURE/SERVER**, and should not be modified directly by the user.

`IP$SSH_LOG_ACCEPTS`

When set, causes the server to log successful connection requests as either an OPCOM message or a line in a log file. Specified by the **SET LOG-ACCEPT** command in **IP CONFIGURE / SERVER**. Note that the server does not use the information set in the ACCEPT-HOSTS keyword in **CONFIGURE /SERVER**. Rather, it uses the AllowHosts and DenyHosts keywords in the SSH server configuration file. Also, a successful connection request doesn't equate to a successful authentication request. This logical should not be modified directly by the user.

`IP$SSH_LOG_REJECTS`

When set, causes the server to log rejected connection requests as either an OPCOM message or a line in a log file. Specified by the **SET LOG-REJECT** command in **IP CONFIGURE/SERVER**. Note that the server does not use the information set in the REJECT-HOSTS keyword in **CONFIGURE/SERVER**. Rather, it uses the AllowHosts and DenyHosts keywords in the SSH server configuration file. This logical should not be modified directly by the user.

`IP$SSH_MAX_SESSIONS`

Set this to the maximum number of concurrent SSH sessions you want to allow on the server system. If `IP$SSH_MAX_SESSIONS` is not defined, the default is 1000. Setting `IP$SSH_MAX_SESSIONS` to zero (0) will cause an error. The value must be between 1 and 1000. The suggested place to set this is in `START_SSH.COM`. SSH must be restarted to use the new value if it is changed.

`SSH_TERM_MBX`

Mailbox used by `SSHD_MASTER` to receive termination messages from SSHD daemon processes. Do not change this logical name. This is created by the `SSHD_MASTER` process.

`IP$SSH_KEYGEN_MIN_PW_LEN`

Defines the minimum passphrase length when one is to be set in **SSHKEYGEN**. If not defined, defaults to zero.

`IP$SSH_PARAMETERS_n`

These values are set by VSI TCP/IP and must not be modified by the user.

`IP$SSH_USE_SYSGEN_LGI`

If defined, causes SSHD to use the OpenVMS SYSGEN value of LGI_PWD_TMO to set the login grace time, overriding anything specified in the command line or the configuration file.

`IP$SSH_ENABLE_SSH2_CONNECTIONS`

Enables SSHD Master to accept SSH V2 sessions.

`IP$SSH2_HOSTKEY_DIR`

Directory containing the hostkeys for the SSH V2 server. Normally set to `IP$SPECIFIC_ROOT:[IP.SSH2.HOSTKEYS]`.

`IP$SSH2_KNOWNHOSTS_DIR`

Directory containing the publickeys for known systems. Normally set to `IP$SPECIFIC_ROOT:[IP.SSH2.KNOWNHOSTS]`.

`SSH2_DIR`

Contains all SSH V2-specific files, such as configuration files. Normally set to `IP$SPECIFIC_ROOT:[IP.SSH2]`.

SSH daemon Files

These files are used by or created by SSH when you log into a daemon. These files are not to be altered in any way.

`SSH_LOG:SSHD.LOG`

This log file is created by each SSHD daemon.

`SSHD_MASTER.LOG`

This log file is created by SSHD_MASTER.

`SSH_START.COM`

This files is used to start SSH.

Chapter 16. Configuring IPSEC and SETKEY

This chapter describes how to configure the IP Security (IPSEC) and SETKEY protocols. IPSEC provides per-packet authenticity/confidentiality guarantees between peers that communicate using IPSEC.

IPSEC provides security for transmission of sensitive information over unprotected networks such as the Internet. IPSEC acts at the network layer, protecting and authenticating IP packets between participating IPSEC devices.

16.1. About the IP Security (IPSEC) Protocol

IPSEC consists of a couple of separate protocols that are listed below:

Authentication Header (AH): Provides authenticity guarantee for packets, by attaching strong crypto checksums to packets. Unlike other protocols, AH covers the whole packet, from the IP header to the end of the packet.

If you receive a packet with AH and the checksum operation was successful, you can be sure about two things *if you and the peer share a secret key, and no other party knows the key*:

- The packet was originated by the expected peer. The packet was not generated by impersonator.
- The packet was not modified in transit.

Encapsulating Security Payload (ESP): Provides confidentiality guarantee for packets, by encrypting packets with encryption algorithms. If you receive a packet with ESP and have successfully decrypted it, you can be sure that the packet was not wiretapped in the middle, *provided that you and the peer share a secret key, and no other party knows the key*.

IP payload compression (IPcomp): ESP provides encryption service to the packets. However, encryption tends to give negative impact to compression on the wire (such as ppp compression). IPcomp provides a way to compress packets before encryption by ESP. (Of course, you can use IPcomp alone if you wish to).

Internet Key Exchange (IKE): An alternate form of keying is called Internet Key Exchange.

Note

Security of IPSEC protocols depends on the secrecy of secret keys. If secret keys are compromised, IPSEC protocols can no longer be secure. Take precautions about permission modes of configuration files, key database files, or whatever may lead to information leakage.

16.2. Security Associations and Security Policies

Both ESP and AH rely on security associations. A *security association* (SA) consists of a source, destination and an instruction. The collection of all SA is maintained by the network kernel for a system is termed the *security association database* (SAD). A sample authentication SA may look like this:

```
add 10.0.0.11 10.0.0.216 ah 15700 -A hmac-md5 "1234567890123456";
```

This says “traffic going from 10.0.0.11 to 10.0.0.216 that needs an AH can be signed using HMAC-MD5 using secret 1234567890123456”. This instruction is labelled with SPI (“Security Parameter Index”) id “15700” (SPI’s will be covered later). SAs are symmetrical; both sides of a conversation share exactly the same SA, it is not mirrored on the other side. Note that there is no ‘autoreverse’ rule - this SA only describes a possible authentication from 10.0.0.11 to 10.0.0.216. For two-way traffic, two SAs are needed.

Following is a sample SA:

```
add 10.0.0.11 10.0.0.216 esp 15701 -E 3des-cbc “123456789012123456789012”;
```

This says “traffic going from 10.0.0.11 to 10.0.0.216 that needs encryption can be enciphered using 3des-cbc with key 123456789012123456789012”. The SPI id is “15701”.

SAs describe possible instructions, but do not in fact describe policy as to when these need to be used. In fact, there could be an arbitrary number of nearly identical SAs with only differing SPI ids. To do actual cryptography, we need to describe a *security policy* (SP). This policy can include things such as “use ipsec if available” or “drop traffic unless we have ipsec”. The collection of SPs for a system is termed the *security policy database* (SPD).

A typical simple Security Policy (SP) looks like this:

```
spdadd 10.0.0.216 10.0.0.11 any -P out ipsec esp/transport//require ah/
transport//require;
```

If entered on host 10.0.0.216, this means that all traffic going out to 10.0.0.11 must be encrypted and be wrapped in an AH authenticating header. Note that this does not describe which SA is to be used, that is left as an exercise for the kernel to determine.

In other words, a Security Policy specifies *what* we want; a Security Association describes *how* we want it.

Outgoing packets are labelled with the SA SPI (“the how”) which the kernel used for encryption and authentication so the remote can look up the corresponding verification and decryption instruction.

16.3. IPSEC Configuration File

The configuration file for IPSEC is loaded by the SETKEY program. The default name and location of this file is `IP$:IPSEC.CONF`, but other filenames may be used when specifying the -f switch for SETKEY.

The configuration file describes all the Security Associations (SA) and Security Policies (SP) for the system. Lines starting with the “#” character are treated as comment lines. Configuration descriptions can be spread over multiple lines in the file. Each description *must* end with a semicolon.

16.3.1. Configuration File Options

add [-4n] src dst protocol spi [extensions] algorithm ...;

Add an SAD entry. This can fail with multiple reasons, including when the key length does not match the specified algorithm.

get [-4n] src dst protocol spi;

Show an SAD entry.

delete [-4n] src dst protocol;

Remove all SAD entries that match the specification.

deleteall [-4n] src dst protocol;

Removes all SAD entries that match the specification.

flush [protocol];

Clears all SAD entries matched by the options. -F on the command line achieves the same functionality.

dump [protocol];

Dumps all SAD entries matched by the options. -D on the command line achieves the same functionality.

spdadd [-4n] src_range dst_range upperspec policy;

Adds an SPD entry.

spdflush ;

Clears all SPD entries. -FP on the command line achieves the same functionality.

spddump ;

Dumps all SPD entries. -DP on the command line achieves the same functionality.

16.3.2. Configuration File Operation Arguments

Arguments for the configuration file operations are as follows:

src/dst

Source/destination of the secure communication is specified as IPv4/v6 addresses. SETKEY can resolve a FQDN into numeric addresses. If the FQDN resolves into multiple addresses, SETKEY will install multiple SAD/SPD entries into the kernel by trying all possible combinations. -4 and -n restricts the address resolution of FQDN in certain ways. -4 restricts results into IPv4/v6 addresses only. -n avoids FQDN resolution and requires addresses to be numeric addresses.

Protocol (where protocol is one of the following):

- ***esp*** ESP based on RFC 2406
- ***esp-old*** ESP based on RFC 1827
- ***ah*** AH based on RFC 2402

- ***ah-old*** AH based on RFC 1826
- ***ipcomp*** IPComp

spi

Security Parameter Index (SPI) for the SAD and the SPD. ***spi*** must be a decimal number or a hexadecimal number prefixed by ‘0x’.

SPI values between 0 and 255 are reserved for future use by IANA.

16.3.2.1. Extensions

May take the following values:

-m mode

Specify a security protocol mode for use. Mode is one of the following: ***transport***,***tunnel*** or ***any***. The default value is ***any***.

-r size

Specify window size in bytes for replay prevention. ***size*** must be a decimal number in 32-bit word. If ***size*** is zero or not specified, replay checks do not take place.

-u id

Specify the identifier of the policy entry in SPD. See ***policy***.

-f pad_option

Defines the content of the ESP padding, and must be one of the following:

- ***zero-pad*** All of the padding are zero.
- ***random-pad*** A series of randomized values are set.
- ***seq-pad*** A series of sequential increasing numbers started from 1 are set.

-f nocyclic-seq

Do not allow cyclic sequence number.

-ls time in seconds

Specify the soft lifetime duration of the SA.

-lh time in seconds

Specify the hard lifetime duration of the SA.

16.3.2.2. Algorithm

-E *ealgo* key

Specify an encryption algorithm for ESP.

-E *ealgo* key -A *aalgo*key

Specify an encryption algorithm ***ealgo***, as well as a payload authentication algorithm ***aalgo***, for ESP.

-A *aalgo* key

Specify an authentication algorithm for AH.

-C *calgo* [-R]

Specify a compression algorithm for IPComp. If -R is specified, the spi field value will be used as the IPComp CPI (compression parameter index). If -R is not specified, the kernel will use well-known CPI, and the spi field will be used only as an index for kernel internal usage.

key must be a double-quoted character string, or a series of hexadecimal digits preceded by “0x”.

Possible values for ***ealgo***, ***aalgo***, and ***calgo*** are specified in Table 16.1 and Table 16.2.

src_range

dst_range

These are selections of the secure communication specified as an IPv4/v6 address or an IPv4/v6 address range, and it may also include a TCP/UDP port specification. The addresses may take one of the following forms (an example of each is shown):

```
Address                    192.168.0.15
Address/prefixlen          192.168.0.15/24
Address[port]              192.168.0.15[1234]
Address/prefixlen[port]    192.168.0.15/24[1234]
```

Note that ***prefixlen*** and ***port*** must be decimal numbers. The square brackets around ***port*** are necessary. For FQDN resolution, the rules applicable to ***src*** and ***dst*** apply here as well.

upperspec

Upper-layer protocol to be used. ***icmp6***, ***ip4***, and ***any*** can be specified, where any indicates “any protocol.” The protocol number may also be used.

policy

Policy is one of the following three formats:

- -P *direction discard protocol/mode/src-dst/level[...]*
- -P *direction none protocol/mode/src-dst/level[...]*
- -P *direction ipsec protocol/mode/src-dst/level[...]*
- ***direction*** must be ***out*** or ***in***.
- ***discard*** means the packet matching indexes will be discarded.
- ***none*** means that IPsec operations will not take place on the packet.
- ***ipsec*** means that IPsec operations will take place on the packet.
- ***protocol/mode/src-dst/level*** specifies the rule as to how the packet will be processed:

- ***protocol*** must be one of ***ah***, ***esp*** or ***ipcomp***.
- ***mode*** must be transport. Note that only transport is valid for the implementation of VSI TCP/IP, as tunneling is not supported.
- Both ***src*** and ***dst*** can be omitted.

level must be one of the following:

- ***default*** means the kernel consults the system-wide default against the protocol specified.
- ***use*** means that the kernel uses a SA if one is available.
- ***require*** means SA is required whenever the kernel sends a packet matched with the policy.
- ***unique*** is the same as require; in addition, it allows the policy to bind with the unique outbound SA.

16.4. Configuration Encryption Algorithms

The following table shows the supported algorithms that can be used as ***aalgo*** in a **-A** ***aalgo*** protocol parameter:

Table 16.1. Authentication Algorithms

Algorithm	**Key Length in Bits**
hmac-md5	128
hmac-sha1	160
keyed-md5	128
keyed-sha1	160
null	0 to 2048

The following table shows the supported algorithms that can be used as *ealgo* in a -E *ealgo* protocol parameter:

Table 16.2. Encryption Algorithms

Algorithm	**Key Length in Bits**
blowfish-cbc	40 to 448
cast128-cbc	40 to 128
des-cbc	64
3des-cbc	192
des-deriv	64
rijndael-cbc	128/192/256

Only ***deflate*** may be used as ***calgo*** in a **-C** ***ealgo*** protocol parameter.

16.5. Simple Configuration Example

What follows is a very simple configuration for talking from host 10.0.0.216 to 10.0.0.11 using encryption and authentication.

On host 10.0.0.216:

```
add 10.0.0.216 10.0.0.11 ah 24500 -A hmac-md5 “1234567890123456”;
add 10.0.0.216 10.0.0.11 esp 24501 -E 3des-cbc
 “123456789012123456789012 22”;
spdadd 10.0.0.216 10.0.0.11 any -P out ipsec esp/transport//require ah/
transport//require;

On host 10.0.0.11, the same Security Associations, no Security Policy:

add 10.0.0.216 10.0.0.11 ah 24500 -A hmac-md5 “1234567890123456”;
add 10.0.0.216 10.0.0.11 esp 24501 -E 3des-cbc
 “123456789012123456789012 22”;
```

With the above configuration in place, $ IP ping 10.0.0.11 from 10.0.0.216 looks like this using tcpdump:

```
22:37:52
 10.0.0.216>10.0.0.11:AH(spi=0x00005fb4,seq=0xa):ESP(spi=0x00005fb5,seq=0xa)
(DF)
23:37:52 10.0.0.11>10.0.0.216:icmp:echo reply
```

Note how the ping back from 10.0.0.11 is plainly visible. The forward ping cannot be read by tcpdump (as the packet is encrypted), but it does show the Security Parameter Index of AH and ESP, which tells 10.0.0.11 how to verify the authenticity of our packet and how to decrypt it.

A problem with the previous example is that it specifies a policy on how 10.0.0.216 should treat packets going to 10.0.0.11, and that it specifies how 10.0.0.11 should treat those packets. However, it *does not* instruct 10.0.0.11 to discard unauthenticated or unencrypted traffic. Hence, anybody could insert spoofed and completely unencrypted data and 10.0.0.11 will accept it. To remedy the above, we need an incoming Security Policy on 10.0.0.11, as follows:

```
spdadd 10.0.0.216 10.0.0.11 any -P in ipsec esp/transport//require ah/
transport//require;
```

This instructs 10.0.0.11 that any traffic coming in to it from 10.0.0.216 is required to have valid ESP and AH.

To complete this configuration, the return traffic needs to be encrypted and authenticated as well. Therefore, the following configurations will be required:

On 10.0.0.216:

```
flush;
spdflush;
# AH
add 10.0.0.11 10.0.0.216 ah 15700 -A hmac-md5 “1234567890123456”;
add 10.0.0.11 10.0.0.11 ah 24500 -A hmac-md5 “1234567890123456”;
# ESP
add 10.0.0.11 10.0.0.216 esp 15701 -E 3des-cbc “123456789012345678901 2”;
add 10.0.0.11 10.0.0.216 esp 24501 -E 3des-cbc “123456789012345678901 2”;
spdadd 10.0.0.216 10.0.0.11 any -P out ipsec esp/transport//require ah/
transport//require;
spdadd 10.0.0.11 10.0.0.216 any -P in ipsec esp/transport//require ah/
transport//require;
```

And on 10.0.0.11:

```
flush;
spdflush;
# AH
add 10.0.0.11 10.0.0.216 ah 15700 -A hmac-md5 “1234567890123456”;
add 10.0.0.11 10.0.0.11 ah 24500 -A hmac-md5 “1234567890123456”;
# ESP
add 10.0.0.11 10.0.0.216 esp 15701 -E 3des-cbc “123456789012345678901 2”;
add 10.0.0.11 10.0.0.216 esp 24501 -E 3des-cbc “123456789012345678901 2”;
spdadd 10.0.0.216 10.0.0.11 any -P out ipsec esp/transport//require ah/
transport//require;
spdadd 10.0.0.11 10.0.0.216 any -P in ipsec esp/transport//require ah/
transport//require;
```

Note that in this example, identical keys were used for both directions of traffic. However, it is not required to use identical keys for both directions.

16.6. The SETKEY Program

The configuration file for IPSEC is loaded by the SETKEY program. A foreign command must be defined to invoke SETKEY:

```
$ setkey :== $IP$:setkey
$ setkey [-v] -f filename
$ setkey [“-aPlv”] “-D”
$ setkey [“-Pv”] “-F”
$ setkey [-h] -x
```

The possible command options for SETKEY are:

-D	Dump the SAD entries. If with -P, the SPD entries are dumped. If with -a, the dead SAD entries will be displayed as well.
-f *filename*	Read the configuration commands from the specified file.
-F	Flush the SAD entries. If with -P, the SPD entries are flushed.
-a	A dead SAD entry means that it has been expired but remains in the system because it is referenced by some SPD entries.
-h	Add hexadecimal dump on -x mode.
-l	Loop forever with short output on -D.
-v	Be verbose. The program will dump messages exchanged on PF_KEY socket, including messages sent from other processes to the kernel.
-x	Loop forever and dump all the messages transmitted to PF_KEY socket.

16.6.1. SETKEY Usage Examples

Example 1: Parse and load the policies in the file `IP$:IPSEC.CONF` into the kernel (note that the output from parsing can be quite verbose, so part of the output has been deleted from the middle this example to keep it to a reasonable size):

```
$ setkey “-f” IP$:ipsec.conf
Starting parse
Entering state 0
Reducing via rule 1 (line 126), -> commands
```

```
state stack now 0
Entering state 1
Reading a token: Next token is 261 (ADD)
Shifting token 261 (ADD), Entering state 2
Reducing via rule 57 (line 537), -> ipaddropts
state stack now 0 1 2
entering state 23
Reading a token: Next token is 292 (STRING)
Shifting token 292 (STRING), Entering state 36
Reducing via rule 61 (line 568), STRING -> ipaddr
state stack now 0 1 2 23
Entering state 39
...
Entering state 19
Reducing via rule 9 (line 141),spdadd_command -> command
state stack now 0 1
Entering state 12
Reducing via rule 2 (line 127), commands command -> commands
state stack now 0
Entering state 1
Reading a token: Now at end of input.
Shifting token 0 ($), Entering state 137
Now at end of input.
```

Example 2: Dump out the policies in the kernel:

```
$ setkey “-PD”
192.168.154.10/24[any] 192.168.228.100/24[any]any
 out ipsec
 esp/transport//use
 ah/transport//use
 spid=1 seq=0 pid=149
 refcnt=1
```

Example 3: Dump out the SAD entries in the kernel:

```
$ setkey “-D”
192.168.228.100 192.168.154.10
 ah mode=any spi=1000(0x00002711)reqid=0(0x00000000)
 A:hmac -sha1 6d6f6761 6d6f6761 6d6f6761 6d6f6761
 replay=0 flags=0x00000040 state=mature seq=3 pid=149
 created: Apr 22 15:52:49 2017 current: Apr 22 15:54:30 2017
 diff: 101(s)       hard:0(s)       soft:0(s)
 last:              hard:0(s)       soft:0(s)
 current:0(bytes)   hard:0(bytes)   soft:0(bytes)
 allocated:0        hard:0 soft:0
 refcnt=1
192.168.154.10.192 168.228.100
ah mode=any spi=9877(0x00002695)reqid=0(0x00000000)
 A:hmac -sha1 686f6765 686f6765 686f6765 686f6765
 replay=0 flags=0x00000040 state=mature seq=2 pid=149
 created: Apr 22 15:52:49 2017 current: Apr 22 15:54:30 2017
 diff: 101(s)       hard:0(s)       soft:0(s)
 last:              hard:0(s)       soft:0(s)
 current:0(bytes)   hard:0(bytes)   soft:0(bytes)
 allocated:0        hard:0          soft:0
 refcnt=1
192.168.228.100.192 168.154.10
```

```
ah mode=transport spi=10000(0x00002710)reqid=0(0x00000000)
 E:3des-cbc deadbeef deadbeef deadbeef deadbeef deadbeef
 replay=0 flags=0x00000040 state=mature seq=1 pid=149
 created: Apr 22 15:52:49 2017 current: Apr 22 15:54:30 2017
 diff: 101(s)       hard:0(s)       soft:0(s)
 last:              hard:0(s)       soft:0(s)
 current:0(bytes)   hard:0(bytes)   soft:0(bytes)
 allocated:0        hard:0          soft:0
 refcnt=1
192.168.154.10 192.168.228.100
ah mode=transport spi=9876(0x00002694)reqid=0(0x00000000)
 E:3des-cbc 686f6765 686f6765 686f6765 686f6765 686f6765
 replay=0 flags=0x00000040 state=mature seq=0 pid=149
 created: Apr 22 15:52:49 2017 current: Apr 22 15:54:30 2017
 diff: 101(s)       hard:0(s)      soft:0(s)
 last:              hard:0(s)      soft:0(s)
 current:0(bytes)   hard:0(bytes)  soft:0(bytes)
 allocated:0        hard:0         soft:0
 refcnt=1
```

Example 4: Dump the messages out on the PF_KEY socket.

```
$ setkey “-hx”
14:38:47.009961
00000000:02 0b 00 00 06 00 00 00 00 00 00 00 00 00 00 00
0000000010:02 0b 00 01 02 00 00 00 00 00 00 95 00 00 00
sadb_msg { version=2 type=1 1 errno=0 satype=1
  len=2 reserved=0 seq=0 pid=149
14:38:47 057809
00000000:02 0b 00 01 02 00 00 00 00 00 00 00 95 00 00 00
```

Example 5: Flush all of the entries from the kernel.

```
$ setkey “-F”
$ setkey “-D”
No SAD entries.
```

16.7. IPSEC Configuration File Examples

16.7.1. Configuration Example: Host-to-Host Encryption

If you want to run host-to-host (transport mode) encryption with manually configured secret keys, the following configuration should be sufficient:

```
# IP$:ipsec.conf
#
# packet will look like this: IPv4 ESP payload
# the node is on 10.1.1.1, peer is on 20.1.1.1
add 10.1.1.1 10.2.1.1 esp 9876 -E 3des-cbc “hogehogehogehogehogehoge”;
add 10.2.1.1 10.1.1.1 esp 10000 -E 3des-cbc
0xdeadbeefdeadbeefdeadbeefdeadbeefdeadbeefdeadbeef;
spdadd 10.1.1.1 10.2.1.1 any -P out ipsec esp/transport//use;
```

From 10.1.1.1 to 10.2.1.1, use the 3DES-CBC algorithm with SPI 9876, with secret key “hogehogehogehhogehogehoge”.The traffic will be identified by SPI 9876.

From 10.2.1.1 to 10.1.1.1, use 3DES-CBC algorithm with SPI 10000, with secret key `0xdeadbeefdeadbeefdeadbeefdeadbeef`.

The last line configures per-packet IPSEC policy for the node. Using this configuration, the transmit node (10.1.1.1) used to send packets to the peer (20.1.1.1), is encrypted whenever a secret key is configured in to the kernel. The configuration does not prohibit unencrypted packets from 20.1.1.1 to reach 10.1.1.1. To reject unencrypted packets, the following line would be added to the configuration file:

```
spdadd 10.2.1.1 10.1.1.1 any -P in ipsec esp/transport//require;
```

On the peer's node (10.2.1.1), the configuration will look similar to what is shown in the following example. Note that the addresses need to be swapped on the "spdadd" line, but "add" lines do not need to be swapped.

```
# IP$:ipsec.conf
#
# packet will look like this: IPv4 ESP payload
# the node is on 10.2.1.1, peer is on 10.1.1.1
add 10.1.1.1 10.2.1.1 esp 9876 -E 3des-cbc “hogehogehogehogehogehoge”;
add 10.2.1.1 10.1.1.1 esp 10000 -E 3des-cbc
0xdeadbeefdeadbeefdeadbeefdeadbeefdeadbeefdeadbeef
spdadd 10.2.1.1 10.1.1.1 any -P out ipsec esp/transport//use;
```

The previous example uses human-readable secret keys. However, using human-readable secret keys is discouraged by the IPSEC policy specification, since they are more likely to be compromised than binary keys. Binary keys should be used for normal operations.

Key length is determined by algorithms. See Table 16.1 and Table 16.2 for the required key lengths. For 3des-cbc, the secret key *must* be 192 bits (24 bytes). If a shorter or longer key is specified, SETKEY will return an error when parsing the line.

The following is an example of rijndael-cbc (also known as AES) using 128-bit keys.

```
# IP$:ipsec.conf
#
# the packet will look like this: IPv4 ESP payload
# the node is on 10.1.1.1, peer is on 10.2.1.1
# rijndael -cbc with 128bit key
add 10.1.1.1 10.2.1.1 esp 9876 -E rijndael -cbc “hogehogehogehoge”;
add 10.2.1.1 10.1.1.1 esp 10000 -E rijndael -cbc
0xdeadbeefdeadbeefdeadbeefdeadbeefdeadbeefdeadbeef;
spdadd 10.1.1.1 10.2.1.1 any -P out ipsec esp/transport//use;
```

16.7.2. Configuration Example: Host-to-Host Authentication

The following example shows a sample configuration for host-to-host authentication:

```
# IP$:ipsec.conf
#
# the packet will look like this: IPv4 ESP payload
# the node is on 10.1.1.1, peer is on 10.2.1.1
# rijndael -cbc with 128bit key
add 10.1.1.1 10.2.1.1 esp 9876 -E rijndael -cbc “hogehogehogehoge”;
add 10.2.1.1 10.1.1.1 esp 10000 -E rijndael -cbc
```

```
0xdeadbeefdeadbeefdeadbeefdeadbeefdeadbeefdeadbeef;
spdadd 10.1.1.1 10.2.1.1 any -P out ipsec esp/transport//use;
```

16.7.3. Configuration Example: Host-to-Host Encryption+Authentication

The following example shows sample keys that are configured for both AH and ESP.

Note

It is recommended that you apply AH after ESP.

```
# IP$:ipsec.conf
#
# packet will look like this: IPv4 AH ESP payload
# the node is on 10.1.1.1, peer is on 10.2.1.1
add 10.1.1.1 10.2.1.1 esp 9876 -E 3des-cbc “hogehogehogehogehogehoge”;
add 10.2.1.1 10.1.1.1 esp 10000 -E 3des-cbc
0xdeadbeefdeadbeefdeadbeefdeadbeefdeadbeefdeadbeef;
add 10.1.1.1 10.2.1.1 ah 9877 -A hmac-md5 “mogamogamogamoga”;
spdadd 10.1.1.1 10.2.1.1 any -P out ipsec esp/transport//use
ah/transport//use;
```

16.8. Conformance to Standards and Interoperability

The VSI TCP/IP IPSEC implementation conforms to the following IPSEC standards: RFC 2401, RFC 2402, RFC 2404, RFC 2406, RFC 4835, RFC 4868. The IKEv2 (Racoon2) implementation provides support for RFC 4306, RFC 4307 and RFC 4718.

16.9. Racoon Internet Key Exchange Daemon

The RACOON service performs the task of securely creating security associations for participating systems. When a security policy senses the need of a security association, RACOON is notified and securely communicates with an Internet Key Exchange daemon on the other system to establish the security association. Security Policies must still be configured manually with the SETKEY program.

The RACOON service can be controlled through the following **IP NETCONTROL** commands:

- DEBUG – set the debug level for a currently running Racoon IKE daemon.
- ESTABLISH remote-IP-address [local-IP-address] – initiate key exchange protocol communication between the remote IP-address and the local IP-address. If local-IP-address is not specified then the value of `IP$HOST_NAME` is used. This does not install Security Associations, but does the initial negotiation necessary to allow Security Associations to be established when necessary. It is not necessary to manually establish the negotiation information – RACOON will do it automatically when necessary.
- FLUSH – flush all existing keys
- NOOP – No Operation

- SHOW – show existing key associations
- SHUTDOWN – Shutdown the Racoon IKE Daemon. All established keys are flushed as part of this process.
- START – Start the Racoon IKE Daemon
- STOP – Same as SHUTDOWN
- VERSION – Display control interface version

The configuration file (`IP$:RACOON.CONF`) can be configured to handle all systems in general, or specific systems.

RACOON negotiates security associations for itself (ISAKMP SA, or phase 1 SA) and for kernel IPsec (IPsec SA, or phase 2 SA). The file consists of a sequence of directives and statements. Each directive is composed by a tag, and statements are enclosed by '{' and '}'. Lines beginning with '#' are comments.

16.9.1. Meta Syntax

Keywords and special characters that the parser expects exactly are displayed using **this** font. Parameters are specified with *this* font. Square brackets '[' and ']' are used to show optional keywords and parameters. Note that you have to pay attention when this manual is describing port numbers. The port number is always enclosed by '[' and ']'. In this case, the port number is not an optional keyword. If it is possible to omit port number, the expression becomes [[port]]. The vertical bar ('|') is used to indicate a choice between optional parameters. Parentheses '(' and ')' are used to group keywords and parameters when necessary. Major parameters are listed below.

Number means a hexadecimal or a decimal number. The former must be prefixed with `0x'.

string path file means any string enclosed in '"' (double quote).

Address means IPv4 address.

Port means a TCP/UDP port number. The port number is always enclosed by '[' and ']'.

Timeunit is one of following: **sec**, **secs**, **second**, **seconds**, **min**, **mins**, **minute**, **minutes**, **hour**, **hours**.

16.9.2. Path Specification

path include *path*;

specifies a path to include a file. See Section 16.9.3.

path pre_shared_key *file*;

specifies a file containing pre-shared key(s) for various ID(s). See Section 16.9.10.

path certificate *path*;

racoon will search this directory if a certificate or certificate request is received.

path backupsa *file*;

specifies a file to be stored a SA information which is negotiated by racoon. racoon will install SA(s) from the file with a boot option -B. The file is increasing because racoon simply adds a SA to the file at the moment. You should maintain the file manually.

16.9.3. File Inclusion

include *file*

other configuration files can be included.

16.9.4. Timer Specification

timer { *statements* }

specifies various timer values.

counter *number*;

the maximum number of retries to send. The default is 5.

interval *number timeunit*;

the interval to resend, in seconds. The default time is 10 seconds.

persend *number*;

the number of packets per send. The default is 1.

phase1 *number timeunit*;

the maximum time it should take to complete phase 1. The default time is 15 seconds.

phase2 *number timeuni*;

the maximum time it should take to complete phase 2. The default time is 10 seconds.

16.9.5. Listening Port Specification

listen { *statemen*ts }

If no listen directive is specified, racoon will listen on all of the available interface addresses. The following is the list of valid statements:

isakmp *address* [[*port*]];

If this is specified, racoon will only listen on *address*. The default port is 500, which is specified by IANA. You can provide more than one address definition.

strict_address;

require that all addresses for ISAKMP must be bound. This statement will be ignored if you do not specify any addresses.

16.9.6. Remote Nodes Specifications

remote (*address* | **anonymous**) [[*port*]] { *statements* } specifies the parameters for IKE phase 1 for each remote node. The default port is 500. If **anonymous i**s specified, the statements apply to all peers which do not match any other remote directive.

The following are valid statements:

exchange_mode (**main** | **aggressive** | **base**);

defines the exchange mode for phase 1 when racoon is the initiator. In addition, it means the acceptable exchange mode when racoon is responder. More than one mode can be specified by separating them with a comma. All of the modes are acceptable. The first exchange mode is what racoon uses when it is the initiator.

doi ipsec_doi;

means to use IPSEC-DOI as specified RFC 2407. You can omit this statement.

situation identity_only;

means to use SIT_IDENTITY_ONLY as specified RFC 2407. You can omit this statement.

my_identifier *idtype ...*;

specifies the identifier sent to the remote host and the type to use in the phase 1 negotiation. *address*, *fqdn*, *user_fqdn*, *keyid* and *asn1dn* can be used as an idtype. they are used like:

my_identifier address [*address*];

the type is the IP address. This is the default type if you do not specify an identifier to use.

my_identifier user_fqdn *string*;

the type is a USER_FQDN (user fully-qualified domain name).

my_identifier fqdn *string*;

the type is a FQDN (fully-qualified domain name).

my_identifier keyid *file*;

the type is a KEY_ID. my_identifier asn1dn [string]; the type is an ASN.1 distinguished name. If string is omitted, racoon will get DN from Subject field in the certificate.

peers_identifier *idtype ...*;

specifies the peer's identifier to be received. If it is not defined then racoon will not verify the peer's identifier in ID payload transmitted from the peer. If it is defined, the behavior of the verification depends on the flag of verify_identifier. The usage of idtype is same to my_identifier.

verify_identifier (**on** | **off**);

If you want to verify the peer's identifier, set this to on. In this case, if the value defined by peers_identifier is not same to the peer's identifier in the ID payload, the negotiation will failed. The default is off.

certificate_type *certspec*;

specifies a certificate specification. *certspec* is one of followings:

x509*certfile privkeyfile*;

certfile means a file name of certificate. *privkeyfile* means a file name of secret key.

peers_certfile (**dnssec** | *certfile*);

If **dnssec** is defined, racoon will ignore the CERT pay- load from the peer, and try to get the peer's certificate from DNS instead. If *certfile* is defined, racoon will ignore the CERT payload from the peer, and will use this certificate as the peer's certificate.

send_cert (**on** | **off**);

If you do not want to send a certificate for some reason, set this to off. The default is **on**.

send_cr (**on** | **off**);

If you do not want to send a certificate request for some reason, set this to off. The default is **on**.

verify_cert (**on** | **off**);

If you do not want to verify the peer's certificate for some reason, set this to **off**. The default is **on**.

lifetime time *number timeunit*;

define a lifetime of a certain time which will be proposed in the phase 1 negotiations. Any proposal will be accepted, and the attribute(s) will be not proposed to the peer if you do not specify it (them). They can be individually specified in each proposal.

initial_contact (**on** | **off**);

enable this to send an INITIAL-CONTACT message. The default value is **on**. This message is useful only when the implementation of the responder choices an old SA when there are multiple SAs which are different established time, and the initiator reboots. If racoon did not use the message, the responder would use an old SA even when a new SA was established. The KAME stack has the switch in the system wide value, net.key.preferred_oldsa. When the value is zero, the stack always use a new SA.

passive (**on** | **off**);

If you do not want to initiate the negotiation, set this to on. The default value is off. It is useful for a server.

proposal_check *level;*

specifies the action of lifetime length and PFS of the phase 2 selection on the responder side. The default level is **strict**. If the level is:

- **obey** the responder will obey the initiator anytime.
- **Strict** If the responder's length is longer than the initiator's one, the responder uses the initiator's one. Otherwise, it rejects the proposal. If PFS is not required by the responder, the responder will obey the proposal. If PFS is required by both sides and if the responder's group is not equal to the initiator's one, then the responder will reject the proposal.

- **Claim** If the responder's length is longer than the initiator's one, the responder will use the initiator's one. If the responder's length is shorter than the initiator's one, the responder uses its own length AND sends a RESPONDER-LIFETIME notify message to an initiator in the case of lifetime. About PFS, this directive is same as **strict**.
- **exact** If the initiator's length is not equal to the responder's one, the responder will reject the proposal. If PFS is required by both sides and if the responder's group is not equal to the initiator's one, then the responder will reject the proposal.

support_mip6 (**on** | **off**);

If this value is set on then both values of ID payloads in phase 2 exchange are always used as the addresses of end-point of IPsec-SAs. The default is **off**.

generate_policy (on | **off**);

This directive is for the responder. Therefore you should set passive on in order that racoon only becomes a responder. If the responder does not have any policy in SPD during phase 2 negotiation, and the directive is set on, then racoon will choice the first proposal in the SA payload from the initiator, and generate policy entries from the proposal. It is useful to negotiate with the client which is allocated IP address dynamically. Note that inappropriate policy might be installed by the initiator because the responder just installs policies as the initiator proposes. Therefore, that other communication might fail if such policies installed. This directive is ignored in the initiator case. The default value is **off**.

nonce_size *number*;

define the byte size of nonce value. Racoon can send any value although RFC2409 specifies that the value MUST be between 8 and 256 bytes. The default size is 16 bytes.

proposal { *sub-substatements* }

encryption_algorithm *algorithm*;

specify the encryption algorithm used for the phase 1 negotiation. This directive must be defined. algorithm is one of following: **des**, **3des**, **blowfish**, **cast128** for Oakley. For other transforms, this statement should not be used.

hash_algorithm *algorithm*;

define the hash algorithm used for the phase 1 negotiation. This directive must be defined. algorithm is one of following: **md5**, **sha1** for Oakley.

authentication_method *type*;

defines the authentication method used for the phase 1 negotiation. This directive must be defined. type is one of: *pre_shared_key*, *rsasig*, *gssapi_krb*.

dh_group *group*;

define the group used for the Diffie-Hellman exponentiations. This directive must be defined. group is one of following: **modp768**, **modp1024**, **modp1536**. Alternatively, you can define 1, 2, or 5 as the DH group number. When you want to use aggressive mode, you must define same DH group in each proposal.

lifetime time *number timeunit*;

define lifetime of the phase 1 SA proposal. Refer to the description of **lifetime** directive immediately defined in remote directive.

gssapi_id *string*;

define the GSS-API endpoint name, to be included as an attribute in the SA, if the **gssapi_krb** authentication method is used. If this is not defined, the default value of `ike/hostname' is used, where hostname is the FQDN of the interface being used.

16.9.7. Policy Specifications

The policy directive is obsolete, policies are now in the SPD. racoon will obey the policy configured into the kernel by setkey, and will construct phase 2 proposals by combining **sainfo** specifications in **racoon.conf**, and policies in the kernel.

16.9.8. Sainfo Specifications

sainfo (*source_id destination_id* | **anonymous**) { *statements* }

defines the parameters of the IKE phase 2 (IPsec-SA establishment). *source_id* and *destination_id* are constructed like:

address *address* [/ *prefix*] [[*port*]] *ul_proto*

or

idtype string

It means exactly the content of ID payload. This is not like a filter rule. For example, if you define 3ffe:501:4819::/48 as *source_id*. 3ffe:501:4819:1000:/64 will not match.

pfs_group *group*;

define the group of Diffie-Hellman exponentiations. If you do not require PFS then you can omit this directive. Any proposal will be accepted if you do not specify one. group is one of following: **modp768**, **modp1024**, **modp1536**. Alternatively, you can define 1, 2, or 5 as the DH group number.

lifetime time *number timeunit*;

define the lifetime of amount of time which are to be used IPsec-SA. Any proposal will be accepted, and no attribute(s) will be proposed to the peer if you do not specify it(them). See Section 16.9.6 [256] on the **proposal_check** directive.

racoon does not have the list of security protocols to be negotiated. The list of security protocols are passed by SPD in the kernel. Therefore, you have to define all of the potential algorithms in the phase 2 proposals even if there is an algorithm which will not be used. These algorithms are define by using the following three directives, and they are lined with single comma as the separator. For algorithms that can take variable-length keys, algorithm names can be followed by a key length, like ``blowfish 448''. racoon will compute the actual phase 2 proposals by computing the permutation of the specified algorithms, and then combining them with the security protocol specified by the SPD. For example, if **des**, **3des**, **hmac_md5**, and **hmac_sha1** are specified as algorithms, we have four combinations for use with ESP, and two for AH. Then, based on the SPD settings, racoon will construct the actual proposals. If the SPD entry asks for ESP only, there will be 4 proposals. If it asks for both AH and ESP, there will be 8 proposals. Note that the kernel may not support the algorithm you have specified.

encryption_algorithm *algorithms*;

des, **3des**, **des_iv64**, **des_iv32**, **rc5, rc4**, **idea**, **3idea**, **cast128**, **blowfish**, **null_enc**, **twofish**, **rijndael** (used with ESP)

authentication_algorithm*algorithms*;

des, **3des**, **des_iv64**, **des_iv32**, **hmac_md5**, **hmac_sha1**, **non_auth** (used with ESP authentication and AH)

compression_algorithm *algorithms*;

deflate (used with IPComp)

Logging level

log *level*;

define logging level. Level is one of following: **notify**, **debug** and **debug2**. The default is notify. If you put too high logging level on slower machines, IKE negotiation can fail due to timing constraint changes.

Specifying the way to pad

padding { *statements* }

specified padding format. The following are valid statements:

randomize (**on** | **off**);

enable using a randomized value for padding. The default is **on**.

r**andomize_length** (**on** | **off**);

the pad length is random. The default is **off**.

maximum_length *number*;

define a maximum padding length. If **randomize_length** is **off**, this is ignored. The default is 20 bytes.

exclusive_tail (**on** | **off**);

means to put the number of pad bytes minus one into last part of the padding. The default is **on**.

strict_check (**on** | **off**);

means to be constrained the peer to set the number of pad bytes. The default is **off**.

Special directives

complex_bundle (**on** | **off**);

defines the interpretation of proposal in the case of SA bundle. Normally "IP AH ESP IP payload" is proposed as "AH tunnel and ESP tunnel". The interpretation is more common to other IKE implementations, however, it allows very limited set of combinations for proposals. With the option enabled, it will be pro- posed as "AH transport and ESP tunnel". The default value is **off**.

Pre-shared key *File*

Pre-shared key file defines a pair of the identifier and the shared secret key which are used at Pre-shared key authentication method in phase 1. The pair in each lines are separated by some number of blanks and/or tab characters like hosts (5). Key can be included any blanks because all of the words after 2nd column are interpreted as a secret key. Lines start with `#' are ignored. Keys which start with `0x' are hexadecimal strings. Note that the file must be owned by the user ID running racoon (usually the privileged user), and must not be accessible by others.

16.9.9. Example RACOON configuration file:

```
#
# Basic Racoon configuration file
#
path pre_shared_key "IP$:psk.txt" ;
remote anonymous
{
  exchange_mode aggressive, main, base ;
  lifetime time 24 hour ;
  proposal {
   encryption_algorithm blowfish 448;
   hash_algorithm md5; #sha1;
   authentication_method pre_shared_key ;
   dh_group 5 ;
  }
}
sainfo anonymous
{
  pfs_group 2;
  lifetime time 12 hour ;
  encryption_algorithm blowfish 448;
  authentication_algorithm hmac_sha1, hmac_md5 ;
  compression_algorithm deflate ;
}
```

16.9.10. Example pre-shared key file:

```
192.168.1.2 deadbeef
192.168.1.3 deadbeef
192.168.1.4 deadbeef
192.168.1.5 face0ff0
```

16.10. Restrictions

The following restrictions exist regarding the use of IPSEC in VSI TCP/IP. These restrictions may be lifted in future releases of VSI TCP/IP.

- Security may not be set on a socket-by-socket basis via the use of setsockopt().
- Only transport mode is supported to both AH and ESP. Tunnel mode (primarily used for VPN's) is not supported in any mode (AH or ESP).
- IPcomp is not currently implemented.

16.11. IPSec key management with Racoon2

Racoon2 is the IPSEC key management package that replaces Racoon. The VSI TCP/IP implementation is based upon racoon2-20090327c from the WIDE project. Racoon2 is available on VSI OpenVMS TCP/IP and offers IKEv1 in main mode and IKEv2. Racoon2 consists of three images:

- SPMD – the Security Policy Management Daemon, which installs security policies and acts on various requests from IKED
- SPMDCTL – the program to send SPMD control messages while it is running
- IKED – the IPSEC Key Exchange Daemon, which authenticates the identification of a remote system and establishes security associations. IKED supports the following specifications:
    - Internet Key Exchange (IKEv2) Protocol
        - RFC 4306, Internet Key Exchange (IKEv2) Protocol
        - RFC 4307, Cryptographic Algorithms for Use in the Internet Key Exchange Version 2 (IKEv2)
        - RFC 4718, IKEv2 Clarifications and Implementation Guidelines
    - The Internet Key Exchange (IKE)
        - RFC 2409, The Internet Key Exchange (IKE)
        - RFC 3947, Negotiation of NAT-Traversal in the IKE
        - RFC 3948, UDP Encapsulation of IPsec ESP Packets

All images assume a default configuration file of `IP$:RACOON2.CONF`.

Additional information on Racoon2 is available from http://www.racoon2.wide.ad.jp/.

Note

KINKD is not provided because there are a number of Kerberos routines that it needs that are not visible in the packages provided.

16.11.1. SPMD

SPMD manages IPsec Security Policy for racoon2, the key management daemon (IKED). It can also serve as a local DNS proxy server if you set on in racoon2 configuration file.

usage:

```
spmd [-dhV] [-f config] [-D level]
-V show version number
-f specify config file
-d increase debug level(max ddd)
-D specify debug level(1-3)
-h show this help
```

Note

When using command line options, make sure to enclose them in double quotes to preserve case.

SPMD must be started first. Users who are familiar with SPMD on Unix platforms will notice that SPMD on VSI TCP/IP only runs in "foreground mode".

16.11.2. Name resolution

- SPMD can provide name resolution for IKED on systems that do not run a name server. This can be either as a name server proxy, or based completely upon the `HOSTS.LOCAL` file. The logical `IP$SVCORDER` can be used to define the order of name resolution and can be defined to any combination of **local** and **bind**. The default value is **bind**.

```
$ define/system IP$SVCORDER "local, bind"
```

16.11.3. SPMDCTL

SPMDCTL connects to the spmd interface which is specified in the racoon2 configuration file, and requests operations to SPMD. These requests can be changes to the currently managed security policies and name servers, if SPMD is acting as a proxy name server.

usage:

```
spmdctl [-d] [-f RACOON2_CONF_FILE] COMMAND
  -d : display messages corresponded with spmd
  -f RACOON2_CONF_FILE : specify racoon2 configuration file
  COMMAND:
   ns {add|delete} address   : add/delete nameserver
   ns list                   : show nameservers
  policy add selector_index \
   lifetime(sec) {transport|tunnel} \
   sp_src_ipaddr sp_dst_ipaddr \
   [sa_src_ipaddr sa_dst_ipaddr]
       : add policy
  policy delete selector_index
       : delete policy
  policy dump : show policies under spmd management
   interactive               : process only login
   shutdown                  : tell spmd to shutdown
   status                    : show statistics
```

Note

When using command line options, make sure to enclose them in double quotes to preserve case.

16.11.4. IKED

IKED is a key management daemon, which supports the Internet Key Exchange (IKE) protocol version 1 (RFC2409) and version 2 (RFC4306). It is driven by messages from the kernel via the PF_KEYv2 interface or by negotiation requests from remote peers, and manages IPSec Security Associations according to *`racoon2.conf`*. Users who are familiar with IKED on UNIX platforms will notice that IKED on VSI TCP/IP only runs in "foreground mode".

usage:

```
iked [-f file] [-p port] [-46] [-I address] \
  [-S selector_index]
 [-D number] [-dvVh] [-l logfile]
  -f specify the configuration file.
  -p specify the isakmp port number to listen to.
  -4 use IPv4 only.
  -6 use IPv6 only.
  -I immediately initiate to the address specified.
  -S immediately initiate using the selector specified.
  -D specify the debug flags.
  -d increase the debug level.
  -v specify to output messages to standard error,
   in addition to syslog.
  -V show the iked version.
  -h show this help.
  -l specify log output file (instead of syslog).
 Debug option:
  0x0001  DEBUG_FLAG_DEBUG    log debug messages.
  0x0002  DEBUG_FLAG_TRACE    show internal processing trace.
  0x0004  DEBUG_FLAG_CONFIG   show config parsing trace.
  0x0008  DEBUG_FLAG_PFKEY    PFKEY and SPMIF are ignored.
```

Note

When using command line options, make sure to enclose them in double quotes to preserve case.

16.11.5. Authentication with pre-shared keys

IKED uses the entire contents of the PSK file as the key, so watch for any trailing newlines that might get inadvertently added. This is different from Racoon, which uses a PSK file that is keyed by IP Address to designate the key.

16.11.6. Authentication with Certificates

The following steps are required to get two hosts using Racoon2 IKED to authenticate each other with certificates when using the IKEv1 or IKEv2 protocol:

- Create a Certificate Signing Request with an unencrypted private key
- Send the CSR file to the Certificate Authority (CA) for signing. Make sure that the CA includes the following information in the CRT (or PEM) file that is returned:
  - SubjectAtlName = DNS:*FQDN*
  - SubjectAltName = email:*email_address_of_contact*
  - SubjectAltName = IP$:*ip_address_of_system*

If the OpenVMS OpenSSL software is used, then the above information needs to be added to the `SSL$CONF:SSL$CA.CNF` file before signing each certificate under the [CA_x509_extensions] section.

- Convert the CRT files to PEM files (if necessary) with the following commands:

  - Openssl x509 –in host.crt –out host.der –outform DER
  - Openssl x509 –in host.der –inform DER –out host.pem –outform PEM
- Hash certificates (PEM files) (option 9 with `SSL1$COM:SSL1$CERT_TOOL`)
- Distribute the PEM files and hashed files to the hosts involved.

The default format for the SubjectAltName IP address is a binary value. If your CA encodes this as a text string then define the logical `IP$RACOON2_BINARY_IPV4_SUBJECTALTNAME FALSE` (or `IP$RACOON2_BINARY_IPV6_SUBJECTALTNAME` for IPv6).

In addition to the above, IKEv2 needs the following in the Racoon2 configuration:

```
Send_peers_id on;
```

The OpenSSL code that IKED is built with looks in **SSLCERTS:** (not SSL$CERTS:) for the CA certificate, so you need to define the logical SSLCERTS to point to the directory that contains the CA certificate and any intermediate certificates in the chain.

16.11.7. Scripts

Command files (or scripts) can be invoked when certain events occur. The command files are passed up to 8 parameters in the following format:

- Parameter_name=parameter_value
- Possible parameter names:
- LOCAL_ADDR
- LOCAL_PORT
- REMOTE_ADDR
- REMOTE_PORT
- SELECTOR_INDEX
- IPSEC_MODE
- UPPER_LAYER_PROTOCOL
- INTERNAL_ADDR4
- APPLICATION_VERSION
- OLD_SRC
- OLD_DST
- NEW_SRC
- NEW_DST
- INTERNAL_ADDR

- INTERNAL_DNS4
- INTERNAL_WINS4
- INTERNAL_DHCP4
- INTERNAL_ADDR6
- INTERNAL_DNS6
- INTERNAL_DHCP6
- LOCAL_NET_ADDR
- LOCAL_NET_PREFIXLEN
- LOCAL_NET_PORT
- REMOTE_NET_ADDR
- REMOTE_NET_PREFIXLEN
- REMOTE_NET_PORT

16.11.8. Compatibility with Racoon

The VSI TCP/IP implementation of IKED has been modified so that it works with Racoon with more encryption keys than the typical IKED implementation would. Unfortunately, these changes will prevent it from working with other implementations of IKED in IKEv1 mode for some encryption methods. To disable this change define the following logical to No/False/0 before starting IKED:

```
$ DEFINE/SYSTEM IP$RACOON2_IKEV1_MORE_DEFAULT_KEYLENS NO
```

16.11.9. Troubleshooting

The first step in troubleshooting is to start IKED with “-D7”. If possible, this should be done on both sides of the connection because one side may provide more useful information than the other as to why the security associations could not be negotiated. Potential causes of problems are:

- Identification parameters (FQDN and IPADDR)
- Authentication (pre-shared key or certificate)
- IPSec requirements (requested combinations of ESP and AH, encryption and hash algorithms.)

For IKEv1 there may be a set of lines similar to:

```
2017-06-24 09:34:24 [DEBUG]: IPSEC_DOI.C;23:372: Compared: DB:Peer
2017-06-24 09:34:24 [DEBUG]: IPSEC_DOI.C;23:373: (lifetime = 28800:600)
2017-06-24 09:34:24 [DEBUG]: IPSEC_DOI.C;23:376: (lifebyte = 0:0)
2017-06-24 09:34:24 [DEBUG]: IPSEC_DOI.C;23:382: enctype = 3DES-CBC:3DES-
CBC
2017-06-24 09:34:24 [DEBUG]: IPSEC_DOI.C;23:387: (encklen = 0:0)
2017-06-24 09:34:24 [DEBUG]: IPSEC_DOI.C;23:389: hashtype = MD5:SHA1
2017-06-24 09:34:24 [DEBUG]: IPSEC_DOI.C;23:394: authmethod = pre-shared
 key:pre-shared key
```

```
2017-06-24 09:34:24 [DEBUG]: IPSEC_DOI.C;23:399: dh_group = 1536-bit MODP
 group:1536-bit MODP group
2017-06-24 09:34:24 [DEBUG]: IPSEC_DOI.C;23:271: no suitable proposal
 found.
```

The above information points out the differences in the IKEv1 transport configurations between the two systems. The values compared are a combination of program defaults, values in the default configuration file, and values in the transport_ike configuration file, so all configuration files must be checked for the differences.

16.11.10. PSKGEN

PSKGEN is a simple program to write a text string to a specified file such that there is no record information in the file and no extraneous carriage control characters. This makes the file compatible with a file that might be generated on a Unix system. Use PSKGEN to generate a pre-shared key file, if that is the authentication method you have chosen. To use the program, define a symbol:

```
$ pskgen :== $IP$:pskgen
$ pskgen IP$:racoon2_psk.txt deadbeef
```

usage:

```
pskgen file_specification pre_shared_key_text
```

16.11.11. Starting Racoon2 on VSI TCP/IP

Perform the following steps to start Racoon2 on VSI TCP/IP.

1. Set up IP$CONFIG:LOCAL_INITIALIZATION.COM to enable IPsec in the kernel before services are started.

   ```
   $ COPY [.CONFIG.IP.CONFIG]LOCAL*.* IP$CONFIG:/LOG
   %COPY-S-COPIED, SYS$SYSDEVICE:
   [KITS.CONFIG.IP.CONFIG]LOCAL_INITIALIZATION.COM;2
   COPIED TO SYS$SYSROOT:[IP.CONFIG]LOCAL_INITIALIZATION.COM;2 (1 BLOCK)
   $ TYPE IP$CONFIG:LOCAL_INITIALIZATION.COM
   $ IP SET /KERNEL NO_IPSEC 0
   $
   $ @SYS$STARTUP:IP$STARTUP

   %IP-I-STARTUP, starting VSI TCP/IP
   ```

   The system displays status messages incuding:

   ```
   %IP-I-LDRIMG, added Loaded Image Descriptor for IP$KERNEL, base address
    89937D40
   %IP-I-LOADING, loading configuration from SYS$SYSROOT:
   [IP.CONFIG]NETWORK_DEVICES
   .CONFIGURATION;2
   .
   .
   .
   VSI TCP/IP Mail Configuration Utility V5.5(28)
   [Assembling configuration information from old locations]
   [Writing configuration to IP$COMMON_ROOT:[SYS
   $STARTUP]START_SMTP_LOCAL.COM]
   [Writing configuration to IP$COMMON_ROOT:[SYS$STARTUP]START_SMTP.COM]
   ```

```
$
```

2. Copy over the needed templates:

```
$ SET DEF IP$CONFIG:
$ DIR *.TEMPLATE

$ COPY RACOON2_CONF.TEMPLATE RACOON2.CONF/LOG
%COPY-S-COPIED, SYS$COMMON:[IP.CONFIG]RACOON2_CONF.TEMPLATE;1 COPIED TO
 SYS$SYSR
OOT:[IP.CONFIG]RACOON2.CONF;1 (3 BLOCKS)
$ COPY RACOON2_DEFAULTS_CONF.TEMPLATE RACOON2_DEFAULTS.CONF/LOG
%COPY-S-COPIED, SYS$COMMON:[IP.CONFIG]RACOON2_DEFAULTS_CONF.TEMPLATE;1
 COPIED TO
 SYS$SYSROOT:[IP.CONFIG]RACOON2_DEFAULTS.CONF;1 (4 BLOCKS)
$ COPY TRANSPORT_IKE_CONF.TEMPLATE TRANSPORT_IKE.CONF/LOG
%COPY-S-COPIED, SYS$COMMON:[IP.CONFIG]TRANSPORT_IKE_CONF.TEMPLATE;1
 COPIED TO SY
S$SYSROOT:[IP.CONFIG]TRANSPORT_IKE.CONF;1 (3 BLOCKS)
$ COPY VALS_CONF.TEMPLATE VALS.CONF/LOG
%COPY-S-COPIED, SYS$COMMON:[IP.CONFIG]VALS_CONF.TEMPLATE;1 COPIED TO SYS
$SYSROOT
:[IP.CONFIG]VALS.CONF;1 (6 BLOCKS)
$
```

3. Create IP$CONFIG:SPMD.PWD that is used to validate communication between SPMD and IKED:

```
$ EDIT/TPU SPMD.PWD
```

   Add the password to the first line of the file and save it.

4. Create the preshared key that matches the key that is found on the peer system:

```
$ EDIT/TPU PSK.TXT
```

   Add the key to the first line of the file and save it.

5. Adjust the values in VALS.CONF with the appropriate 'this host' and peer IP and names:

```
setval {
### Directory Settings ###
        # Preshared key file directory : specify if you want to use
 preshared k#
        PSKDIR          "ip$config:";

        # Cert file directory : specify if you want to use certs
        CERTDIR         "sys$specific:[ip.certs]";

### ID Settings ###
        # your FQDN : specify if you want to use FQDN as your ID
        MY_FQDN         "tbs.eng.vmssoftware.com";

        # Peer's FQDN : specify if you want to use FQDN as peer's ID
        PEERS_FQDN      "tba.eng.vmssoftware.com";

### Preshared Key Setting ###
        # Preshared Key file name
        # You can generate it by pskgen.
        PRESHRD_KEY     "psk.txt";

### Certicate Setting ###
        # Set following parameters if you use certificates in IKE
 negotiation
        # and _SET_ 'kmp_auth_method { rsasig; };' in each remote{}
 section of
        # tunnel_ike{_natt}.conf/transport_ike.conf files.
        # For more information, please see USAGE.
        #
        # Your Public Key file name
        MY_PUB_KEY      "thishost_signed.crt";

        # Your Private Key file name
        MY_PRI_KEY      "thishost.key";

        # Peer's Public Key file name
        PEERS_PUB_KEY   "peerhost_pub.pem";

### Transport Mode Settings ###
        # Your IP Address
        MY_IPADDRESS    "192.168.1.187";

        # Peer's IP Address
        PEERS_IPADDRESS "192.168.2.188";

### Tunnel Mode Settings ###
        # Your Network Address or Host Address (host-to-host tunnel
 mode)
        MY_NET          "10.0.0.0/24";
        # Peer's Network Address or Host Address (host-to-host tunnel
 mode)
        PEERS_NET       "10.0.1.0/24";
```

6. Start Racoon2 with the following command:

```
$ @IP$STARTUP:START_RACOON2
```

16.11.12. Configuration

Note

The following is copied from the doc directory in the Racoon2 distribution; some misspellings and grammatical errors have been corrected and some information has been changed to match the VSI TCP/IP implementation.

The iked of Racoon2 supports Configuration Payload in IKEv2 protocol. This section describes how to use it with iked.

16.11.12.1. Introduction

The Configuration Payload (CP) is used to exchange configuration information between IKE peers. There are several kinds of information to be exchanged with CP in RFC4306. The current implementation however supports only the following pieces of information:

- INTERNAL_IP4_DNS

- INTERNAL_IP4_NBNS
- INTERNAL_IP4_DHCP
- APPLICATION_VERSION
- INTERNAL_IP6_DNS
- INTERNAL_IP6_NBNS
- INTERNAL_IP6_DHCP

The Configuration Payload has two types of exchange. One is request/reply and the other is set/ack. The current specification only defines the request/reply usage. The IKED support only request/reply exchange accordingly.

16.11.12.2. How IKED works

The IKED of an initiator side sends a request to the responder and it receives the reply. Then it parses the reply and validates. The IKED exports the information from the Configuration Payload as environment variables and calls the scripts in the configuration files. An administrator can use the information to configure callback scripts.

The IKED of a responder side receives a request from the initiator. It looks up the configuration and sends the reply. There is no callback interface which the responder calls in the process of the Configuration Payload.

Initiator side

There are two directives to configure the Configuration Payload. One is "request" and the other is "script". They are in the "ikev2" directive in the "remote" directive.

The "request" is used to configure what information will be requested. For example:

```
request { ip4_dns; application_version; };
```

Specifies that IKED will request INTERNAL_IP4_DNS and APPLICATION_VERSION.

"script" is used to configure the callback scripts of the Configuration Payload exchange. To use the Configuration Payload, we should configure child_up and child_rekey. For example:

```
script {
child_up "IP$:child-up";
child_rekey "IP$:child-rekey";
};
```

In child-up and child-rekey, the information from the Configuration Payload is available as parameters to the command procedure containing the name and value of environment variables corresponding to the information. For example, the environment variable INTERNAL_DNS4 stores INTERNAL_IP4_DNS.

Responder side

There are directives to configure the Configuration Payload. The "provide" directive specifies the information that will be provided to the initiators. The directive is in the "ikev2" directive in the "remote" directive. For example:

```
provide {
dhcp { 192.168.39.10; };
application_version "Racoon2";
};
```

It specifies that the IKED can provide "192.168.39.10" as an INTERNAL_IP4_DNS and "Racoon2" as an APPLICATION_VERSION.

16.11.12.3. Configuration Syntax

This section describes the syntax of the configuration of Racoon2.

The syntax of the Racoon2 configuration is not compatible with the syntax of old Racoon. Note that this syntax might be changed in the future because Racoon2 is still under development.

Structure

There are ten main directives in the configuration. For the detailed information on the directives, refer to Section 16.11.12.4.

- setval - It defines a constant string value which is unique in the whole configuration file.
- default - It defines default values.
- interface - It defines interfaces of each protocol.
- resolver - It defines the resolver.
- remote - It defines parameters for the remote KMP peer.
- selector - It defines parameters of a selector.
- policy - It defines a behavior when a packet is matched to a selector.
- ipsec - It defines a SA or a SA bundle.
- sa - It defines parameters of a SA.
- addresspool - It defines the address ranges for address pool.

The following picture shows how each directive relates to the others.

```
setval    default    interface    resolver
     +---(selector_index)--- remote
     |                         ^
     |                         |
     |                   (remote_index)                      +-(sa_index)-
> sa
     v                         |                             |
selector -+                    |     +-(ipsec_index)-> ipsec -+-(sa_index)-
> sa
          |                    |     |
selector -+-(policy_index)-> policy -+-(ipsec_index)-> ipsec ---(sa_index)-
> sa
          |                          |
selector -+                          +-(ipsec_index)-> ipsec ...
```

```
          :                          :
```

Limitation of string

A string consists of the following characters:

```
0x30-0x39 0-9
0x41-0x5a A-Z
0x61-0x7a a-z
0x25      %
0x2a      *
0x2d      -
0x2e      .
0x2f      /
0x3a      :
0x3f      ?
0x40      @
0x5f      _
```

A non-reserved string must be enclosed by double-quotations (" : 0x22"). When characters are enclosed by double-quotations, it is distinguished as just a string. An index, selector_index for example, should consist of alpha-numeric characters (0-9 a-z A-Z). An index is not required to be enclosed by ". An IP address and a port number are not required to be enclosed by ".

Representation of IP addresses

An IPv4 address must consist of numeric characters (0-9) and periods (.).

```
e.g. 203.178.141.194
```

An IPv6 address must consist of hexadecimal-digit (0-9 a-f A-F), colons (:) and a percentage-mark (%) if necessary.

```
e.g. 2001:200:0:8002:203:47ff:fea5:3085
```

When a port number is required, the string "port" must follow an IP address string, and must be followed by a port number or a service name defined by the platform generally defined by /etc/services. The string "any" means that it will match with all port numbers.

```
e.g.
  2001::1 port 80
  203.178.141.194 port any
```

A network prefix is represented by a number delimited by a slash (/).<screen>e.g. ::1/0</screen>

Some reserved strings can be used.

- MY_IP - all of IP addresses assigned to the interfaces.
- MY_IPV6 - all of IPv6 addresses assigned to the interfaces.
- MY_IPV6_GLOBAL - all of IPv6 global addresses assigned to the interfaces.
- MY_IPV6_LINKLOCAL - all of IPv6 link-local addresses assigned to the interfaces.
- MY_IPV4 - all of IPv4 addresses assigned to the interfaces.

An interface name can be specified with a percentage-mark (%) followed by the interface name.

`e.g. MY_IPV6%fxp0`

The following strings will be implemented.

`IP_ANY means ::0 and 0.0.0.0.`

Representation of bytes

The following directives are followed by a numeric byte amount specification as its parameter.

- nonce_size
- max_pad_len
- max_retry_to_send
- kmp_sa_lifetime_byte
- ipsec_sa_lifetime_byte

The following units can be used.

- B
- Byte
- Bytes
- KB
- MB
- GB

Representation of time

The following directives are followed by a numeric time data.

- interval_to_send
- times_per_send
- kmp_sa_lifetime_time
- kmp_sa_nego_time_limit
- kmp_sa_grace_period
- ipsec_sa_nego_time_limit
- ipsec_sa_lifetime_time
- dpd_delay
- dpd_retry

The following units can be used.

- infinite
- sec
- secs
- second
- seconds
- min
- mins
- minute
- minutes
- hour
- hours
- day
- days
- 0 (zero) means infinite.

Cryptographic algorithm and its representation

The following directives define an algorithm type.

- kmp_enc_alg
- esp_enc_alg
- esp_auth_alg
- ah_auth_alg

All of them are sent as a proposal, and a receiver evaluates the proposal with logical OR.

They can define a size of a key and a key if needed.

```
(algorithm name)[,(key length)[,(key)]]
```

(key) is hexadecimal-digits or string. A string of hexadecimal-digits must start with (0x).

```
e.g. 0x0123456789abcdef
```

In the string case, it must be closed by ("). Note that some algorithms specify the length of the key.

If you do not specify size of key, you can define like below:

```
(algorithm name),,(key)
```

If you need to specify multiple algorithms, defining an algorithm list for example, you can define it by using (;) as delimiter, e.g.:

```
kmp_enc_alg { aes192_cbc,,0x1234; aes192_cbc; 3des_cbc; };
```

The following lists cryptographic functions and algorithm names. Note that some algorithms are not implemented.

kmp_enc_alg directive and esp_enc_alg directive can have one of the following algorithm types.

- des_cbc_iv64
- des_cbc
- 3des_cbc
- rc5_cbc
- idea_cbc
- cast128_cbc
- blowfish_cbc
- 3idea_cbc
- des_cbc_iv32
- rc4_cbc
- null_enc
- rijndael_cbc
- aes128_cbc
- aes192_cbc
- aes256_cbc
- twofish_cbc

kmp_hash_alg directive can have one of the following algorithm types.

- md5
- sha1
- tiger
- sha2_256
- sha2_384
- sha2_512

In case of IKEv2, kmp_hash_alg directive is used to specify an integrity check (MAC) algorithm for IKE_SA communication, and the following algorithm types are accepted.

- hmac_md5

- hmac_sha1
- sha2_256
- sha2_384
- sha2_512
- aes_xcbc
- aes_cmac
- kmp_prf_alg directive can have one of the following algorithm types.
- hmac_md5
- hmac_sha1
- hmac_sha2_256
- hmac_sha2_384
- hmac_sha2_512
- aes_xcbc
- aes_cmac
- des_mac
- kpdk_md5

In case of IKEv1, kmp_prf_alg directive is not used. Instead, HMAC version of hash algorithm specified by kmp_hash_alg is used.

kmp_dh_group directive can have the group number or one of the following algorithm types.

- 1 modp768
- 2 modp1024
- 3 ec2n155
- 4 ec2n185
- 5 modp1536
- 14 modp2048
- 15 modp3072
- 16 modp4096
- 17 modp6144
- 18 modp8192

kmp_auth_method directive can have one of the following algorithm types.

- psk
- dss
- rsasig
- rsaenc
- rsarev
- gssapi_krb

esp_auth_alg and ah_auth_alg directive can have one of the following algorithm types.

- hmac_md5
- hmac_sha1
- aes_xcbc
- hmac_sha2_256
- hmac_sha2_384
- hmac_sha2_512
- kpdk_md5
- non_auth

ipcomp_alg directive can have one of the following algorithm types.

```
oui
deflate
lzs
```

Variable substitution

Environment variables can be referred from configuration files. The form is like $[environment variable].

```
e.g. $[HOME]
```

The C getenv routine is used to determine the value of the variable. You can also define variables by using setval directives. They are evaluated at once after all configuration files are read. When there is a duplicate string, it fails and stops evaluation of the configuration file.

To define a variable, a variable name must be followed by a string:

```
(string) (value) ;
```

The string must begin with a capital alphabet, followed by alpha-numeric capital characters. Alpha-numeric capital characters are:

```
0x30-0x39 0-9
```

```
0x41-0x5a A-Z
0x5f      _
```

To refer the variable, the form is like ${string}.

```
e.g. ${HISNAME}
```

setval can have a environment variable referred by $[variable] as its parameter.

When (# 0x23) is found, all characters from it to a new line are ignored.

16.11.12.4. Directives details

A directive consists of a string, a value and ";"(semi-colon) e.g.

```
string value ;
```

Values can be enclosed by "{" and "}" like

```
directive {
  value ;
   value ;
   :
  };
```

A value might be a directive recursively.

Values enclosed with ({) and (}) must also end with a semi-colon (;). The word is case-sensitive. The following lists the syntax of each directive.

```
include
    include (file) ;
```

The parser includes (file). Note that variables defined by setval are not allowed here. Environment variables can be used.

```
setval
 setval { (definitions) } ;
```

It defines a constant string value which is unique in the whole configuration file. It will be basically expanded after the whole configuration files are loaded.

```
default
 default { (directives) } ;
```

It defines default values. Each directive can be included. Default values are overwritten by each specific value.

```
interface
 interface { (directives) } ;
```

It defines interfaces of each protocol. Sub-directives are:

```
ike (address) [port (port)] ;
```

It defines port numbers which IKED uses.

```
spmd (address) [port (port)] ;
```

Defines port numbers by which spmd makes communication with IKED. The loopback address (127.0.0.1) is recommended.

```
 spmd_password (file) ;
```

Defines the file name which contains the password to be used for communication between spmd and iked.

```
 application_bypass (on|off) ;
```

When on (default) KMP daemons bypass the IPsec policies. When off, they rely on explicit policies but they can run encapsulated into IPsec tunnels for instance. Note the last configuration can be unsafe when a KMP uses a non-privileged port.

```
resolver
 resolver { (directives) } ;
```

It defines the SPMD resolver proxy configuration. Sub-directives in the resolver directive are the followings.

```
resolver (on|off) ;
```

It controls the behavior of spmd as the resolver. When the directive is on, the spmd behaves as a resolver. Default is off.

```
nameserver (address) [port (port)] ;
```

It defines IP addresses of the upper DNS servers. The port number can be defined if needed. The default port number is 53.

```
dns_query (address) [port (port)] ;
```

It defines IP addresses to be listened to for DNS requests. The port number can be defined if needed. The default port number is 53.

```
remote
 remote (remote_index) { (directives) } ;
```

It defines parameters for the remote KMP peer. remote_index is a string value to identify a remote directive. A remote directive is referred from one or more policies by the remote_index. A remote might refer to the selector directive with the selector_index. Sub-directives in the remote directive are the followings.

```
ikev1 { (directives) } ;
```

It defines the IKEv1 configuration.

```
ikev2 { (directives) } ;
```

It defines the IKEv2 configuration.

```
acceptable_kmp (ikev1|ikev2) ;
```

It defines key management protocols to be used. The list defines the KMPs whom the responder allows to accept. The first protocol in the list defines the mode that the initiator uses.

```
passive (on|off) ;
```

It controls the behavior of the daemon. When this directive is on, the daemon acts as responder only. The default is off.

```
peers_ipaddr (address) [port (port)];
```

It defines the IP addresses of the peer. The port number may be specified if needed. It can be used as the first search key to match with the IKE initial packet from initiator. Omitting this field or writing it as IP_RW requires default directive to handle IKE initial packets.

```
my_id (ipaddr|email|fqdn|keyid|x509_subject) (value) ;
```

They are valid in all KMP directives. It defines identifiers for the local-side entity.

```
ipaddr (ip address)
```

IPv4 or IPv6 address

```
fqdn (FQDN)
```

Fully Qualified Domain Name

```
email (e-mail address)
```

E-Mail address

```
keyid ([file]|tag) (filename|data)
```

A binary data a.k.a KEY-ID. Use a filename (default) or directly the data.

```
x509_subject (filename)
```

Subject Name in the certificate.

```
peers_id (ipaddr|email|fqdn|keyid|x509_subject) (value) ;
```

They are valid in all KMP directives. It defines the identifiers for the remote-side entity.

```
send_peers_id (on|off) ;
```

It enables the initiator to send the peer's ID. The default is off.

```
obey
```

It means that the policy from the initiator will be installed. It is same as "generate_policy on;" in the IKEv1 case.

```
exact
```

If there is no policy matching with the intiator's one, it is rejected. The default is obey.

```
obey
```

It means that the initiator's policy will be used.

```
strict
```

It means that the initiator's policy will be used if the lifetime in the policy from the initiator is less than the responder's one, AND the PFS is required. If it is not suitable, it is rejected.

`claim`

It is valid for the ikev1 directive. It means that the responder will use own policy if the specified value is smaller than peer's proposal, and sends a RESPONDER_LIFETIME notification.

`exact`

It means that the policy from the initiator exactly matches with the responder's one. If it is not suitable, it is rejected. The default is obey.

```
random_pad_content (on|off) ;
padlen_random (on|off) ;
max_padlen (number) ;
```

They are valid in all KMP directives. They define the padding.

```
max_retry_to_send (number) ;
interval_to_send (number) ;
times_per_send (number) ;
```

They are valid in all KMP directives. They define the retransmission timer.

```
kmp_sa_lifetime_time (number) ;
kmp_sa_lifetime_byte (number) ;
kmp_sa_nego_time_limit (number) ;
kmp_sa_grace_period (number) ;
ipsec_sa_nego_time_limit (number) ;
```

They are valid for both directives of the ikev1 and ikev2. They define the lifetime.

```
kmp_enc_alg (algorithm) ;
kmp_hash_alg (algorithm) ;
kmp_prf_alg (algorithm) ;
kmp_dh_group (algorithm) ;
kmp_auth_method (algorithm) ;
```

They are valid for both directives of the ikev1 and ikev2. They define the algorithms for each function. (algorithm) is described at the cryptographic algorithm and its representation. For IKEv1, kmp_prf_alg directive is not used. Instead, HMAC version of hash algorithm specified by kmp_hash_alg is used as prf algorithm.

```
cookie_required (on|off);
```

It is valid for the ikev2 directive. It controls whether the responder requires the cookie. Default is off.

```
need_pfs (on|off) ;
```

It is valid for both directives of the ikev1 and ikev2. It controls to enable PFS or not. The daemon will send a KE payload in the phase 2 of IKEv1. Default is off.

`x509pem`

X.509 PEM format

`pkcs12`

PKCS12 format

`ascii`

PGP ASCII ARMORED format

`pre_shared_key (file)`

It is valid for both directives of the ikev1 and ikev2. It defines the file name contained the pre-shared key.

`my_principal (principal-id)`

It is valid for the kink directive. It defines my principal identifier to be used. (principal-id) is like "principal@realm".

`peers_principal (principal-id)`

It is valid for the kink directive. It defines the peer's principal identifier.

`mobility_role (agent|mobile|correspondent)`

It is valid in all KMP directives. It is used by mobile IPv6 daemons.

`request { (config_request_list) };`

(available with IKEv2 only) Request Configuration Payload option to peer. config_request_list is an arbitrary list of following:

```
dns;
ip4_dns;
ip6_dns;
dhcp;
ip4_dhcp;
ip6_dhcp;
application_version;
```

dns is equivalent to specifying both ip4_dns and ip6_dns; dhcp is equivalent to specifying both ip4_dhcp and ip6_dhcp;

`provide { (provide_option_list) } ;`

(available with IKEv2 only) Provide Configuration Payload option to peer. provide_option_list is a list of following options:

```
addresspool (addresspool_index) ;
dhcp (address) ;
dns (address) ;
application_version (string) ;
```

`dpd (boolean) ;`

(available with IKEv1 only) This option (default on) enables negotiating RFC 3706 Dead Peer Detection. For IKEv2, DPD (liveliness check) is always enabled.

`dpd_delay (number) ;`

(available with IKEv1 or IKEv2) This option activates the DPD and sets the time (in seconds) allowed between two proof of life requests. For IKEv1, the default value is 0, which disables DPD monitoring, but still negotiates DPD support. For IKEv2, the default value is 3600 (1 hour).

`dpd_retry (number) ;`

(available with IKEv1 only) If dpd_delay is set, this sets the delay (in seconds) to wait for a proof of life before considering it failed and sending another request. The default value is 5. For IKEv2, normal retransmission time is used instead.

```
dpd_maxfail (number) ;
```

(available with IKEv1 only) If dpd_delay is set, this sets the maximum number of proofs of life to request before considering the peer dead. The default value is 5. For IKEv2, normal retransmission algorithm with max_retry_to_send is used instead.

```
script { (script_list) } ;
```

(available with IKEv1 or IKEv2) Defines a list of hook scripts. script_list is a list of following items.

```
phase1_up (script_path) ;
phase1_down (script_path) ;
phase2_up (script_path) ;
phase2_down (script_path) ;
phase1_rekey (script_path) ;
phase2_rekey (script_path) ;
migration (script_path) ;
```

Also, ike_sa_up, ike_sa_down, ike_sa_rekey, child_up, child_down, child_rekey are synonymous to phase1_up, phase1_down, phase1_rekey, phase2_up, phase2_down, phase2_rekey, respectively. For IKEv1, only the phase1_up and phase1_down are effective. No other events are available. Scripts' argv[1] is equivalent to the event name. Parameters are passed using the environment variables. For phase1_up and phase1_down, following environment variables are defined:

```
LOCAL_ADDR
LOCAL_PORT
REMOTE_ADDR
REMOTE_PORT
```

For phase2_up and phase2_down:

```
LOCAL_ADDR
REMOTE_ADDR
SELECTOR_INDEX
IPSEC_MODE
LOCAL_NET_ADDR
LOCAL_NET_PREFIXLEN
LOCAL_NET_PORT
REMOTE_NET_ADDR
REMOTE_NET_PREFIXLEN
REMOTE_NET_PORT
UPPER_LAYER_PROTOCOL (decimal number or any)
INTERNAL_ADDR (only if an address is leased to peer)
INTERNAL_ADDR4 (leased from peer)
INTERNAL_DNS4
INTERNAL_WINS4
INTERNAL_DHCP4
INTERNAL_ADDR6
INTERNAL_DNS6
INTERNAL_DHCP6
```

For migration:

```
OLD_SRC
```

```
OLD_DST
NEW_SRC
NEW_DST
```

```
selector
```

```
selector (selector_index) { (directives) } ;
```

It defines parameters of a selector. selector_index is a string value to identical a selector directive. A selector directive refers to the policy directive with the policy_index. Sub-directives in the selector directive are the followings.

```
direction (inbound|outbound);
```

It defines the direction of the packet.

```
src (address) [port (port)];
```

```
dst (address) [port (port)];
```

It defines an IP address to be matched with packets. It can not be listed. A port number can be defined if needed.

```
upper_layer_protocol (protocol) [(options)];
```

It defines the last upper layer protocol to be matched with packets. Any and strings in /etc/protocols can be used in (protocol). (options) depends on the protocol. For example, ipv6-icmp (type) (code)

```
tagged (pf_tag_name);
```

It overloads at the bootstrap installation the previous selectors by a pf tag. The usual selectors are still taken into account by KMPs but not by the kernel. Dynamic operations have to be done on pf rules, not on the SPD.

```
policy_index (policy_index) ;
```

It has policy_index to define a policy.

```
reqid (number);
```

It defines the request ID for SA sharing

```
(cf. unique ipsec_level).
```

It is used to support Mobile-IP.

```
policy
```

```
policy (policy_index) { (directives) } ;
```

It defines a behavior after a packet is matched to a selector. policy_index is a string value to identical a policy directive. A policy is referred from one or more selectors by the policy_index(es). A policy refers to one or more IPSEC directives. The IPSEC directives will evaluate in logical OR by a KMP daemon. A policy may refer to a remote directive. Sub-directives in the policy directive are the followings.

```
action (auto_ipsec|static_ipsec|discard|none) ;
```

It defines an action of the policy.

`auto_ipsec`

It means the policy needs a key management.

`static_ipsec`

IT IS NOT IMPLEMENTED.

`discard`

It means the policy discards packets.

`none`

It means the policy bypasses the IPsec stack.

`install (on|off) ;`

Default is on, it makes possible to only declare the policy.

`remote_index`

It has a remote_index.

`ipsec_index`

It has ipsec_index(es) to define proposals of IPsec. It is valid when the action directive is auto_ipsec.

`my_sa_ipaddr (address) ;`

It defines an IP address of the end points of the SAs on my side. (address) is an IP address. It must be defined when the action directive is static_ipsec or tunnel.

`peers_sa_ipaddr (address) ;`

It defines an IP address of the end points of the SAs on the peer's side. Write IP_RW to generate policy dynamically.

`ipsec_level (unique|require|use) ;`

`use`

NOT IMPLEMENTED. When there is no SA for the packet, the kernel sends an acquire for the SA to KMPs, and sends the packet.

`require`

When there is no SA for the packet, the kernel sends an acquire for the SA to KMPs, and discards the packet. This SA installed will be used from other policies.

`unique`

PARTIALLY IMPLEMENTED, NOT EXPECTED TO WORK In addition to require directive, this SA will not be used from any other policy.

`ipsec_mode (transport|tunnel) ;`

It defines a IPsec mode.

`ipsec`

```
ipsec (ipsec_index) { (directives) } ;
```

It defines a SA or a SA bundle. ipsec_index is a string value to identical a ipsec directive. An ipsec directive refers to one or more sa directives with sa_index(es). An ipsec directive is referred from the policy directive by the ipsec_index. Sub-directives in the ipsec are the followings.

```
ipsec_sa_lifetime_time (number) ;
```

It defines a life time of the SA in time.

```
ipsec_sa_lifetime_byte (number) ;
```

It defines a life time of the SA in bytes.

```
ext_sequence (on|off) ;
```

It enables extended sequence number processing described in the 2401bis. The default is off.

```
sa_index (sa_index) ;
```

It has sa_index(es) to define an SA bundle. It can have three sa_indexes in maximum. When you want to define multiple SAs as an SA bundle, note that the following patterns are only allowed.

```
AH
ESP
IPCOMP
AH_ESP
AH_IPCOMP
ESP_IPCOMP
AH_ESP_IPCOMP
```

```
sa
```

```
sa (sa_index) { (directives) } ;
```

It defines parameters of an SA. sa_index is a string value to identical the sa directive. A sa directive is referred from the ipsec directive by the sa_index. Sub-directives in the sa are the followings.

```
sa_protocol (ah|esp|ipcomp) ;
```

It defines a protocol to be used.

```
esp_enc_alg (algorithm) ;
```

It defines encryption algorithms to be used. It is valid when the sa_protocol is esp.

```
esp_auth_alg (algorithm) ;
```

It defines authentication algorithms to be used. It is valid when the sa_protocol is esp.

```
ah_auth_alg (algorithm) ;
```

It defines authentication algorithms to be used. It is valid when the sa_protocol is ah.

```
ipcomp_alg (algorithm) ;
```

It defines compression algorithms to be used. It is valid when the sa_protocol is ipcomp.

```
spi (spi) ;
```

NOTE THAT IT MAY BE OBSOLETE. It defines a SPI for a static SA.

```
addresspool
```

```
addresspool (addresspool_index) { (address_ranges) } ;
```

It defines address ranges to make available for remote host. For the address range, the first and last IP addresses of the range should be specified, separated with a - (hyphen-minus), followed by a ; (semicolon):

```
(address) - (address) ;
```

Note: you should put spaces before and after the hyphen, or else it may be parsed as a part of address string.

16.11.12.5. Sample configuration

The sample configuration files below are intended to show some of the functionality available, and do not illustrate the complete configuration language.

racoon2.conf

```
# Edit racoon2_vals.conf for your environment
include "IP$:racoon2_vals.conf";
# interface info
interface
{
  ike {
   MY_IP port 500;
  };
#
# For OpenVMS specify loopback address and port number.
#
  spmd {
   127.0.0.1 port 5500;
  };
  spmd_password "IP$:spmd.pwd";
};

# resolver info
resolver
{
  resolver off;
#  resolver on;
#  nameserver {
#   192.168.0.3 port 53;
#  };
#   dns_query {
#   127.0.0.1 port 53;
#   ::1 port 53;
#  };
};

#
# This line includes default configuration file;
# Please don't touch this line (especially novice user);
#

include "IP$:racoon2_default.conf";
```

```
## Transport mode IKEv2 or IKEv1
include "IP$:transport_ike.conf";
```

racoon2_vals.conf

```
setval {
### Directory Settings ###
# Preshared key file directory : specify if you want to use preshared
# keys
  PSKDIR  "IP$:";

  # Cert file directory : specify if you want to use certs
  CERTDIR "SSL$CERTS:";
### ID Settings ###
  # your FQDN : specify if you want to use FQDN as your ID
  MY_FQDN "client.example.com";
  # Peer's FQDN : specify if you want to use FQDN as peer's ID
  PEERS_FQDN "server.example.com";
### Preshared Key Setting ###
  # Preshared Key file name
  # You can generate it by pskgen.
  PRESHRD_KEY "psk2.txt";

### Certicate Setting ###
 # Set following parameters if you use certificates in
 # IKE negotiation and
 #_SET_ 'kmp_auth_method { rsasig;};' in each remote{}
 # section of tunnel_ike{_natt}.conf/transport_ike.conf
 # files.
 # For more information, please see USAGE.
 #
 # Your publickey file name
 MY_PUB_KEY "client.pem";

 # Your Private Key file name
 MY_PRI_KEY "client.key";

 # Peer's publickey file name
 PEERS_PUB_KEY  "server.pem";

### Transport Mode Settings ###

 # Your IP Address

 MY_IPADDRESS  "192.168.0.1”;

 # Peer's IP Address
 PEERS_IPADDRESS "198.168.0.2";

### Configuration Payload Settings (for IKEv2)###
 # IPv4 Address Pool For Assignment
 CP_ADDRPL4_START "10.7.73.128";
 CP_ADDRPL4_END "10.7.73.254";

 # IPv6 Address Pool For Assignment
 CP_ADDRPL6_START "fd01::1000";
 CP_ADDRPL6_END "fd02::2000";
 # DNS Server Address(es) (ex. "10.7.73.1; 10.7.73.2")
 CP_DNS   "10.7.73.1";

 # DHCP Server Address(es)
 CP_DHCP  "10.7.73.1";

 # Application Version String
 CP_APPVER  "Racoon2 iked"

### Scripts
 ## IKEv2
# IKESAUP_SCR  "IP$:ikesa-up.com";
# IKESADOWN_SCR  "IP$:ikesa-down.com";
# CHILDUP_SCR  "IP$:child-up.com";
# CHILDOWN_SCR  "IP$:child-down.com";
# IKESAREKEY_SCR "IP$:ikesa-rekey.com";
# CHILDREKEY_SCR "IP$:child-rekey.com";
# MIGRATION_SCR "IP$:migration.com";
  ## IKEv1
# PH1UP_SCR  "IP$:ph1-up.com";
# PH1DOWN_SCR  "IP$:ph1-down.com";
```

racoon2_default.conf

```
#
# default section
#
default
{
  remote {
  acceptable_kmp { ikev2; ikev1; };
  ikev1 {
   logmode normal;
   kmp_sa_lifetime_time 600 sec;
   kmp_sa_lifetime_byte infinite;
   interval_to_send 10 sec;
   times_per_send 1;
   ipsec_sa_nego_time_limit 40 sec;
   kmp_enc_alg { 3des_cbc; };
   kmp_hash_alg { sha1; md5; };
   kmp_dh_group { modp3072; modp2048; modp1024; modp1536;};
   kmp_auth_method { psk; };
   random_pad_content off;
  };
  ikev2 {
   logmode normal;
   kmp_sa_lifetime_time infinite;
   kmp_sa_lifetime_byte infinite;
   max_retry_to_send 3;
   interval_to_send 10 sec;
   times_per_send 1;
   kmp_sa_nego_time_limit 60 sec;
   ipsec_sa_nego_time_limit 40 sec;
   kmp_enc_alg { 3des_cbc; };
   kmp_prf_alg { hmac_md5; hmac_sha1; };
   kmp_hash_alg { hmac_sha1; hmac_md5; };
   kmp_dh_group { modp3072; modp2048; modp1024;  };
   kmp_auth_method { psk; };
   random_pad_content on;
   random_padlen on;
   max_padlen 50 bytes;
  };
  };

  policy {
   ipsec_mode transport;
   ipsec_level require;
  };
  ipsec {
   ipsec_sa_lifetime_time infinite;
   ipsec_sa_lifetime_byte infinite;
  };

  sa {
   esp_enc_alg { 3des_cbc; };
   esp_auth_alg { hmac_sha1; hmac_md5; };
  };
};
ipsec ipsec_ah_esp {
  ipsec_sa_lifetime_time 28800 sec;
  sa_index { ah_01; esp_01; };
};
ipsec ipsec_esp {
  ipsec_sa_lifetime_time 28800 sec;
  sa_index esp_01;
};

sa ah_01 {
  sa_protocol ah;
  ah_auth_alg { hmac_sha1; hmac_md5; };
};

sa esp_01 {
  sa_protocol esp;
  esp_enc_alg { 3des_cbc; };
  esp_auth_alg { hmac_sha1; hmac_md5; };
};
```

transport_ike.conf

```
#
# default section
#
default
{
  remote {
  acceptable_kmp { ikev2; ikev1; };
  ikev1 {
   logmode normal;
   kmp_sa_lifetime_time 600 sec;
   kmp_sa_lifetime_byte infinite;
   interval_to_send 10 sec;
   times_per_send 1;
   ipsec_sa_nego_time_limit 40 sec;
   kmp_enc_alg { 3des_cbc; };
   kmp_hash_alg { sha1; md5; };
   kmp_dh_group { modp3072; modp2048; modp1024; modp1536;};
   kmp_auth_method { psk; };
   random_pad_content off;
  };
  ikev2 {
   logmode normal;
   kmp_sa_lifetime_time infinite;
   kmp_sa_lifetime_byte infinite;
   max_retry_to_send 3;
   interval_to_send 10 sec;
   times_per_send 1;
   kmp_sa_nego_time_limit 60 sec;
   ipsec_sa_nego_time_limit 40 sec;
   kmp_enc_alg { 3des_cbc; };
   kmp_prf_alg { hmac_md5; hmac_sha1; };
   kmp_hash_alg { hmac_sha1; hmac_md5; };
   kmp_dh_group { modp3072; modp2048; modp1024;  };
   kmp_auth_method { psk; };
   random_pad_content on;
   random_padlen on;
   max_padlen 50 bytes;
  };
  };

  policy {
   ipsec_mode transport;
   ipsec_level require;
  };
  ipsec {
   ipsec_sa_lifetime_time infinite;
   ipsec_sa_lifetime_byte infinite;
  };

  sa {
   esp_enc_alg { 3des_cbc; };
   esp_auth_alg { hmac_sha1; hmac_md5; };
  };
};
ipsec ipsec_ah_esp {
  ipsec_sa_lifetime_time 28800 sec;
  sa_index { ah_01; esp_01; };
};
ipsec ipsec_esp {
  ipsec_sa_lifetime_time 28800 sec;
  sa_index esp_01;
};

sa ah_01 {
  sa_protocol ah;
  ah_auth_alg { hmac_sha1; hmac_md5; };
};

sa esp_01 {
  sa_protocol esp;
  esp_enc_alg { 3des_cbc; };
  esp_auth_alg { hmac_sha1; hmac_md5; };
};
```

Chapter 17. Intrusion Prevention System (IPS)

This chapter describes the VSI TCP/IP Intrusion Prevention System (IPS). This security feature monitors network and/or system activities for malicious or unwanted behavior and can react, in real-time, to block or prevent those activities. IPS is highly flexible and customizable. When an attack is detected, pre-configured rules will block an intruder's IP address from accessing the VSI TCP/IP system, prevent an intruder from accessing a specific application, or both. The time period that the filter is in place is configurable. An API is provided so that VSI TCP/IP customers can incorporate the IPS functionality into user-written applications.

IPS is implemented by instrumenting components (e.g., VSI TCP/IP SSH or FTP, or user-supplied components) with a VSI supplied API that allows them to report events, such as invalid login attempts, to the FILTER_SERVER process. The filter server, started when VSI TCP/IP starts, maintains the component rulesets and lists of events, based on the originating address for the offending connection, and when defined limits are reached, creates and sets timed filters in the VSI TCP/IP kernel to filter that traffic.

17.1. IPS Operation

All of the operating parameters such as the definition of rule, the number of events/unit time to trigger a filter, the duration of a filter, etc. are all defined by component configuration files.

Events are recorded per source address, per rule, per destination address, per component. This provides the ability to have differing filtering criteria for different interfaces (for example, an internal network vs. an external network). Addresses or networks may be excluded from consideration when an event is logged. This feature allows, for example, different settings to be used for internal vs. external networks.

Events “age”; after a time period, old events are discarded from the list of events so that infrequent event occurrences (e.g., mistyping a password) have less chance of inadvertently causing a filter to be set. Note that when a filter is triggered for an address and rule, the list of known events for that rule and address are deleted.

Multiple filters may be set in sequence for a component/rule/source address/destination address as events are logged. The purpose of this is to make a filter progressively longer. For example, the first filter set for an address and rule might be 5 minutes long; the next, 10 minutes long; the next, 15 minutes long; etc., up to 5 filter times.

17.2. Configuring IPS

IPS is configured in two steps:

1. Configuring the main process-specific parameters of the FILTER_SERVER process (for example, the size of the mailbox used by applications to communicate with the FILTER_SERVER process).

2. Editing the FILTER_SERVER configuration files to set the operating parameters of IPS; for example, the applications that will use IPS and setting the rule parameters for reporting events.

Note

The FILTER_SERVER process will not be started if the file `IP$:FILTER_SERVER_CONFIG.TXT` does not exist.

17.2.1. Configuring Process-Specific Parameters

Logical names are used to set process-specific parameters for the FILTER_SERVER mailbox and some of the process-specific quotas for the FILTER_SERVER process. These logical names are:

`IP$FILTER_SERVER_TQELM`

`IP$FILTER_SERVER_ASTLM`

`IP$FILTER_SERVER_MBX_MSGS`

17.2.2. Determining the Correct FILTER_SERVER Process Quotas

It is important to determine correctly the correct process quotas for the `FILTER_SERVER` process. High-volume systems, for example, an E-mail server where SMTP may detect many anomalies, may log large numbers of events in a short time. If the TQELM and ASTLM quotas for `FILTER_SERVER` are too low, the `FILTER_SERVER` process could enter MUTEX state and hang, preventing any events from being logged and possibly leading to other problems such as processes hanging while trying to log events.

The amount of additional TQELM quota in addition to the default value (specified via the *`PQL_DTQELM SYSGEN`* parameter) required for the `FILTER_SERVER` process can be calculated as follows:

- 1 for automated hourly reporting
- 1 for automated 24-hour maintenance
- 1 for each source address per rule per component for which an event has been received. These timers are used to clean up internal address structures and disappear after 24 hours of inactivity from that address.
- 1 for each non-empty event queue per source address per rule per component. These timers are used to delete aged events from the event queue.

Thus, the event frequency must be anticipated and the quotas adjusted appropriately for each installation. The hourly FILTER_SERVER logs will be of use for determining traffic patterns.

The ASTLM quota tends to track the value for TQELM closely, but should have an additional 10% added to it for other uses.

Both the ASTLM and TQELM quotas are controlled by logical names described in the previous section. Both of the ASTLM and TQELM values default to 500.

17.2.3. Determining the Correct FILTER_SEVER Mailbox Size

In addition to setting the TQELM and ASTLM process quotas correctly, the size of the mailbox used for communication with the FILTER_SERVER process must be correctly determined. Failure to do can result in events reported by instrumented components being lost. The mailbox is sized to handle 400 simultaneous event messages by default.

Once the mailbox size has been configured, either the system must be rebooted to allow the new mailbox size to be used (this is the preferred method), or the following procedure can be used to avoid a reboot in the near term:

1. Stop IPS (**IP SET /IPS /STOP**).
2. Stop all applications using IPS (e.g., telnet sessions, ftp session, etc.).
3. Delete the old mailbox by running `IP$:DELMBX.EXE`.
4. Start IPS (**IP SET /IPS /START**).
5. Start any other applications previously stopped.

17.2.4. Filter Server Main Configuration

The filter server is configured using a main configuration file and per-component configuration files. The main configuration file is used to set overall configuration options for filter server operation, while the per-component configuration files contain configuration information for each instrumented component such as the ruleset to use, the prototype filter to be set, etc. Per-component configuration files are referenced by the main configuration file by using the “INCLUDE” keyword.

Sample configuration files are supplied in the VSI TCP/IP distribution and must be copied and modified as necessary to conform to the particular site’s security profile and interface configuration. These files are copied to the IP$: directory when VSI TCP/IP is installed. Once these have been copied and modified, the filter server configuration may be reloaded via the **IP SET /IPS /RELOAD** command. The template files supplied are:

FILTER_SERVER_CONFIG.TEMPLATE

SSH_FILTER_CONFIG.TEMPLATE

IMAP_FILTER_CONFIG.TEMPLATE

POP3_FILTER_CONFIG.TEMPLATE

SNMP_FILTER_CONFIG.TEMPLATE

SMTP_FILTER_CONFIG.TEMPLATE

TELNET_FILTER_CONFIG.TEMPLATE

REXEC_FILTER_CONFIG.TEMPLATE

RSHELL_FILTER_CONFIG.TEMPLATE

RLOGIN_FILTER_CONFIG.TEMPLATE

The following table lists the main configuration file keywords. These are found in the file `IP$:FILTER_SERVER_CONFIG.TXT`:

Table 17.1. Main configuration Keywords

Keyword	Default	Description
BLOCK_AT_DESTINATION_PORT	NO	If set to YES, indicates that all filters generated by the filter server will be for

Keyword	**Default**	**Description**
		a specific destination port (the equivalent of a filter line of “EQ *portnumber*”). The port number to be used is specified for each component in the per-component configuration file. If set NO, all filters generated by the filter server will deny access to all destination ports.
DEBUG *value*	0	Indicates the amount of debug to output. Zero means no debug, while higher number mean more debug. This value should ordinarily never be set above 4 without direction.
ENTERPRISE_STRING		Defines the location in the MIB tree that the trap used to send filter logging events via SNMP pertains to.
GENERIC_TRAP_ID		An integer representing the generic trap value when filter logging events are sent via SNMP.
INCLUDE *filename*		Specifies a per-component configuration file to load. Any number of INCLUDE statements may occur in the main configuration file.
LOG_TO_LOGFILE	NO	If YES, information log messages are sent to the log file specified by the *logfile* keyword.
LOG_TO_OPCOM	NO	If YES, informational messages are reported via OPCOM.
LOG_TO_SNMP	NO	If YES, informational messages are reported via an SNMP trap.
LOGFILE		Specifies the log file used when the *log_to_logfile* keyword is specified.
OPCOM_TARGET	NETWORK, SECURITY	Specifies the list of operator types to which events are written when the keyword LOG_TO_OPCOM is set. This is a comma-separated list, and may contain any of the values that are valid for the OpenVMS **REPLY/ENABLE** command.
SPECIFIC_TRAP_ID		An integer representing the specific trap value when filter logging events are sent via SNMP.

17.2.5. Filter Server Per-Component Configuration File

The per-component configuration files are loaded using the INCLUDE keyword in the main filter server configuration file. Each of these configuration files have the following format. The definition must begin with a COMPONENT keyword. Comments lines begin with a “#” character.

```
COMPONENT component-name
 DESTINATION_ADDRESS
 EXCLUDE_ADDRESS
 DESTINATION_PORT
 PROTO_FILTER
RULE rulename
 DESTINATION_ADDRESS
 DESTINATION_PORT
 MAX_COUNT
 DELTA_TIME
 FILTER_DURATIONS
RULE rulename
 MAX_COUNT
 DELTA_TIME
 FILTER_DURATIONS
```

Each component may have as many rules defined for it as are appropriate for the component. However, the more rules defined for a component, the more complex it may be to instrument the component to actually report those rules. All entries in configuration files are not case-sensitive.

The following table shows the keywords for a per-component configuration file:

Table 17.2. Per-Component Configuration Keywords

Keyword	Scope	Description
COMPONENT *component-name*	component	Name of the component to which this applies (e.g., SSH).
DELTA_TIME *time*	rule	Time, in seconds, where if *max_count* events are received for a rule from the same address, that will cause a filter to be set for that address. This is also the time for aging events. If the age of an event exceeds *delta_time* seconds, the event is dropped from the event list.
DESTINATION_ADDRESS *address*	component or rule	Destination IP address (the VSI TCP/IP interface address) in CIDR format to check. This may be in ipv4 or ipv6 format. If *destination_address* occurs before the first rule in the per-component configuration file, it will be used as a default for any rule for the component that does not have a destination address specified. **Note** Multiple *destination_address* lines may be specified at the component level if all the interfaces specified have the same filtering criteria.

Keyword	Scope	Description
DESTINATION_PORT `port`	component or rule	Optional destination port. This will only be effective if the keyword BLOCK_AT_DESTINATION_PORT is set in the main configuration file.
EXCLUDE_ADDRESS `address`	component or rule	A source address/mask in CIDR format from which events are ignored. This allows, for example, events from an internal network to be ignored while counting events from external networks. Multiple *EXCLUDE_ADDRESS* lines may be specified for each rule.
FILTER_DURATIONS `list`	rule	List of durations for filters. This is a comma-delimited list of up to five filter durations, and it must be terminated with a -1.
MAX_COUNT `count`	rule	Maximum number of events from the same address for a specific rule over `delta_time` seconds that will trigger a filter.
PROTO_FILTER	component	This is a prototype filter to be used to build the filter set against an interface when a filter is triggered. The format of this filter is the same used in a filter file.
RULE `rulename`	component	The user-defined name for a rule.

17.3. Sample Main Configuration File

```
#==========================================================================
#
#    FILTER_SERVER_CONFIG.TEMPLATE
#
#==========================================================================
#
#  The following parameter determines the level of debug information
#  written to the debug log file. This should normally be set to a
#  value of 2 or less, and shouldn't be set above 4 without the
#  recommendation, as higher debug levels will
#  negatively impact the filter server (and possibly, system)
#  performance. The debug messages will be found in the file
#  IP$:FILTER_SERVER.OUT.
#
debug  4
#
#  The following parameters define the logging locations. Note
#  that debug messages are not written to the logging locations.
#
#  The first two parameters are used when logging to a log file.
#
logfile IP$:filter_logfile.log
log_to_logfile yes
```

```
#
#  The next parameter is used when logging to OPCOM.
#
log_to_opcom   yes
opcom_target   NETWORK,SECURITY,OPER3
#
#  The next parameters are used when logging via SNMP. Details
#  on the values for enterprise_string, generic_trap_id and
#  specific_trap_id can be found in the VSI TCP/IP for OpenVMS
#  Administrators Guide.
#
log_to_snmp   no
# enterprise_string
# generic_trap_id
# specific_trap_id
#
#  The following parameter determines how filters are created. If
#  set to YES, then the destination port field is added to the filter
#  (e.g., "192.168.0.11/32 eq 22"). If set to NO, then no source
#  port field is added, which will cause the filter to block all
#  traffic of the specified protocol from the source address. If
#  not set, default is NO.
#
# block_at_destination_port yes
#
#========================================================================
#
#  The following lines define the individual configuration files
#  for each configured component that uses the filter server
#
#========================================================================
#
include IP$:ftp_filter_config.txt
include IP$:imap_filter_config.txt
include IP$:pop3_filter_config.txt
include IP$:smtp_filter_config.txt
include IP$:snmp_filter_config.txt
include IP$:ssh_filter_config.txt
#
```

For this configuration:

- Debug will be reported at level 4 (this produces detailed information, normally useful only by VSI when debugging a problem).
- Log messages will be logged to `IP$:FILTER_LOGFILE.LOG` and OPCOM.
- When filters are logged, the destination port specified in the per-configuration files will be used.
- Per-component configuration files for the VSI TCP/IP FTP, IMAP, POP3, SMTP, SNMP and SSH servers will be loaded.

17.4. Sample Component Configuration File

The following is a configuration file for the SSH component:

```
component ssh
```

```
 proto_filter "deny tcp 192.168.0.100/32 192.168.0.11/32 log"
 destination_address 192.168.0.16/32
 exclude_address 192.168.0/24
 destination_port 22
 rule ssh_bogus_id
  max_count    10
  delta_time   90
  filter_durations 300,600,1800,3600,-1
 rule ssh_authfailed
  max_count    10
  delta_time   90
  filter_durations 300,600,1800,3600,-1
 rule ssh_authfailed
  destination_address 192.168.10.2/16
  max_count    10
  delta_time   90
  filter_durations 300,600,1800,3600,-1
 rule ssh_userauth
  max_count    10
  delta_time   90
  filter_durations 300,600,1800,3600,-1
 rule ssh_invaliduser
  max_count    10
  delta_time   90
  filter_durations 300,600,1800,3600,-1
```

For component SSH, a “deny tcp” filter will be used. The source address/mask and destination address/mask parts of the prototype filter are ignored and are overwritten by the actual data specified by the source information gathered from the event that triggered the filter, and by the destination address/mask/port information specified by the corresponding keywords in this file. Events from the 192.168.0 network are all excluded from being counted.

To examine the first three rules specified above:

The rule is “ssh_bogus_id”. Since no address or mask is specified for this rule, it will use the default destination address of 192.168.0.16 and mask of 255.255.255.255 specified at the beginning of the component configuration. The rule states that if 10 events from the same source address are seen over 90 seconds, a filter is created using the *`proto_filter`* specified above. The first filter is 5 minutes long, the second, 10 minutes, and so on, until at the 5th time, a permanent filter is put in place for the address and interface that is causing the problem.

The second rule is “ssh_authfailed”, and applies to events received as a result of connections on the interface with the default address of 192.168.0.16 and mask of 255.255.255.255, respectively.

The third rule is also “ssh_authfailed”, but applies to events received a result of connections on the interface with the address 192.168.10.2, using a mask of 255.255.0.0. The *`max_count`* and *`delta_time`* parameters are different for this interface from for the previous *`ssh_authfailed`* rule in the system.

The remaining rules for this component will use the default address 192.168.0.16 and mask of 255.255.255.255.

Note

If a rule specifies a destination address for an interface that does not currently exist, events for that interface will be dropped until the interface becomes available.

If your system has multiple interfaces (for example, SE0, SE1 and PD0), you must specify all interfaces in the same config file. For each rule in the config file, you must supply a separate section for each destination address (i.e., interface). The ***component*** keyword may occur exactly once in the configuration file. The following example shows a config file for component ftp for 5 interfaces (SE0, SE1, PD0, PD1, PD2):

```
=======================================================================
#
# FTP_FILTER_CONFIG.TXT
#
# Filter server configuration file for the FTP component.
#
#======================================================================
component ftp
 proto_filter "deny tcp 192.168.0.100/32 192.168.0.1/24 log"
 destination_port 21
#
# For SE0 and SE1
#
 destination_address 192.168.0.29/32
 destination_address 192.168.0.25/32
 rule  ftp_invaliduser
  max_count    10
  delta_time   300
  filter_durations 300,600,1800,3600,-1
  destination_address 192.168.0.29/32
 rule ftp_userauth
  max_count    21
  delta_time   180
  filter_durations 300,600,1800,3600,-1
  destination_address 192.168.0.29/32
 rule ftp_authfailed
  max_count    21
  delta_time   90
  filter_durations 300,600,1800,3600,-1
  destination_address 192.168.0.29/32
 rule ftp_timeout
  max_count    21
  delta_time   90
  filter_durations 300,600,1800,3600,-1
  destination_address 192.168.0.29/32
#
# For PD0
#
 rule ftp_invaliduser
  max_count    10
  delta_time   300
  filter_durations 300,600,1800,3600,-1
  destination_address 192.168.0.28/32
  destination_port 1521
 rule ftp_userauth
  max_count    21
  delta_time   180
  filter_durations 300,600,1800,3600,-1
  destination_address 192.168.0.28/32
  destination_port 1521
 rule ftp_authfailed
```

```
  max_count    21
  delta_time   90
  filter_durations 300,600,1800,3600,-1
  destination_address 192.168.0.28/32
  destination_port 1521
 rule ftp_timeout
  max_count    21
  delta_time   90
  filter_durations 300,600,1800,3600,-1
  destination_address 192.168.0.28/32
  destination_port 1521
#
# For PD1
#
 rule ftp_invaliduser
  max_count    10
  delta_time   300
  filter_durations 300,600,1800,3600,-1
  destination_address 192.168.0.27/32
  exclude_address 192.168.0.0/24
    rule  ftp_userauth
  max_count    21
  delta_time   180
  filter_durations 300,600,1800,3600,-1
  destination_address 192.168.0.27/32
  exclude_address 192.168.0.0/24
 rule  ftp_authfailed
  max_count    21
  delta_time   90
  filter_durations 300,600,1800,3600,-1
  destination_address 192.168.0.27/32
  exclude_address 192.168.0.0/24
 rule ftp_timeout
  max_count    21
  delta_time   90
  filter_durations 300,600,1800,3600,-1
  destination_address 192.168.0.27/32
  exclude_address 192.168.0.0/24
#
# For PD2
#
 rule ftp_invaliduser
  max_count    10
  delta_time   300
  filter_durations 300,600,1800,3600,-1
  destination_address 192.168.0.21/32
 rule ftp_userauth
  max_count    21
  delta_time   180
  filter_durations 300,600,1800,3600,-1
  destination_address 192.168.0.21/32
 rule ftp_authfailed
  max_count    21
  delta_time   90
  filter_durations 300,600,1800,3600,-1
  destination_address 192.168.0.21/32
 rule ftp_timeout
  max_count    21
  delta_time   0
  filter_durations 300,600,1800,3600,-1
  destination_address 192.168.0.21/32
```

The above example shows some configuration options for the system with 5 interfaces. Specifically:

- Interfaces SE0 and SE1 will use identical rules, because they did not specify destination addresses within their rulesets and the destination addresses for SE0 and SE1 were specified at the component level. All other interface rules specified their own destination addresses at the rule level, so they will use specific rules for those specific addresses.
- A default port of 21 was been specified for all interfaces. However, interface PD0 has specified a port of 1521, so that port will be used for PD0 only. All other interfaces will use the default port of 21.
- Interface PD1 has an *`exclude_address`* specified for net 192.168.0.0/24. All events for PD1 that originated from that source net will be excluded from being counted by IPS. All other interfaces will count events from that network.

17.5. Configuring IPS for Paired Network Interfaces

To configure IPS for a paired netowrk interface environment where multiple interfaces are treated as a common link set, the rules are fairly simple.

- Each physical and pseudo interface must be specified in the configuration files via *`destination_address`* rules for each interface.
- All physical interfaces are treated equally. When an event is logged for any interface in the set, it is as if it was logged against each interface in the set. Thus, when a filter is set on any interface in the set, the same filter is set on all physical interfaces in the set.
- Filters are set only on the physical interfaces. Since pseudo devices (PD *`nnn`*) are not true interfaces, they cannot have filters set on them.
- When a filter is created as a result of events coming in via a pseudo device, the destination address shown in the filter (using the **IP SHOW INTERFACE /FILTER** command) will show the destination address for the pseudo device.

When **IP SET INTERFACE** is used to perform any of the following tasks:

- Create a paired network interface set via **SET INTERFACE /COMMON_LINK**
- Start an interface via **SET INTERFACE/UP**

IPS is notified of the change being made. This allows the `FILTER_SERVER` process to reevaluate all interfaces it knows about, so it can determine if modifications must be made to paired network interface sets about which it currently knows.

The following example shows a configuration for the SSH component for a paired network interface configuration that consists of SE0, SE1, and PD0 where PD0's physical interface is SE1:

```
component ssh
 proto_filter "deny tcp 192.168.0.100/32 192.168.0.11/24 log"
 #
  # SE0’s address
 #
 destination_address 192.168.0.70/32
 #
 # SE1’s address
 #
 destination_address 192.168.0.71/32
 #
 # PD0’s address
 #
 destination_address 192.168.0.72/32
 #
 destination_port 22
 rule ssh_bogus_id
  max_count    10
  delta_time   90
  filter_durations 300,600,1800,3600,-1
 rule ssh_authfailed
  max_count    10
  delta_time   90
  filter_durations 300,600,1800,3600,-1
 rule  ssh_authfailed
  destination_address 192.168.10.2/32
  max_count    10
  delta_time   90
  filter_durations 300,600,1800,3600,-1
 rule ssh_userauth
  max_count    10
  delta_time   90
  filter_durations 300,600,1800,3600,-1
 rule ssh_invaliduser
  max_count    10
  delta_time   90
  filter_durations 300,600,1800,3600,-1
```

Using the above configuration, the next item illustrates a filter being set due to events that occurred on line PD0:

```
RAPTOR_$
%%%%%%%%%% OPCOM 29-MAR-2017 13:00:55.77 %%%%%%%%%%%(from node VOODOO at
 29-MAR-2017 13:00:59.12)
Message from user JOHNDOE on VOODOO
FILTER_SERVER: Filter queued on SE0 (192.168.0.70/32) at 29-MAR-2017
 13:00:59.12
  Component: ssh, Rule: ssh_bogus_id
Deny  tcp 192.168.0.11/32
  192.168.0.72/32 eq 22
  FLTSVR,LOG
  START: 29-MAR-2017 13:00:59.12 END: 29-MAR-2017 14:00:59.12
RAPTOR_$
%%%%%%%%%%% OPCOM 29-MAR-2017 13:00:55.80 %%%%%%%%%%%(from node VOODOO at
 29-MAR-2017 13:00:59.15)
Message from user JOHNDOE on VOODOO
FILTER_SERVER: Filter queued on SE1 (192.168.0.71/32) at 29-MAR-2017
 13:00:59.15
  Component: ssh, Rule: ssh_bogus_id
Deny  tcp 192.168.0.11/32
```

```
  192.168.0.72/32 eq 22
  FLTSVR,LOG
  START: 29-MAR-2017 13:00:59.15 END: 29-MAR-2017 14:00:59.15
RAPTOR_$
```

Note some things illustrated above:

- Each physical address (SE0 and SE1) had a filter set on it.
- No filter was set on interface PD0 because it is a pseudo interface.
- The destination address for each event is that of interface PD0, since that was the source of the events that caused the filters to be set.

17.6. Filter Reporting via OPCOM and Log File

When a filter is set for an address/rule/destination/component, an informational message will appear either in OPCOM (if *`LOG_TO_OPCOM`* is set) or in the logfile (if *`LOG_TO_LOGFILE`* is set). The following message illustrates an OPCOM message, but the message to a logfile will have the same format.

```
TWEET_$
%%%%%%%%%%% OPCOM 16-MAY-2017 10:33:19.74 %%%%%%%%%%%
Message from user SYSTEM on TWEET
FILTER_SERVER: Filter queued on se0 (192.168.0.16) at 16-MAY-2017
 10:33:19.74
  Component: ssh, Rule: ssh_bogus_id
Deny  tcp 192.168.0.11/32
  192.168.0.0/24   eq 22
  FLTSVR,LOG
  START: 16-MAY-2017 10:33:19 END: 16-MAY-2017 10:38:19
TWEET_$
```

This message is in essentially the same format as that when a **IP SHOW /INTERFACE /FILTER** command is performed:

```
TWEET_$ IP show/interface se0/filter
Device se0: flags=8863<UP,BROADCAST,NOTRAILERS,RUNNING,MULTICAST,D2>
 VMS Device = EWA0
 IP Address = 192.168.0.16
 No common links defined
VSI TCP/IP for OpenVMS Packet Filter List for se0:
Logging is disabled
  Source Address / Port
Action   Proto   Hits    Destination Address / Port
------   -----   -----   ----------------------------------------
Deny             tcp     0   192.168.0.11/32
                 192.168.0.0/24 eq 22
                 FLTSVR,LOG
         START: 16-MAY-2017 10:33:19 END: 16-MAY-2017 10:38:19
Permit   ip      13484   0.0.0.0/0
                         0.0.0.0/0
                 FLTSVR
         Average 0 bytes out, 0 bytes in per second
         Average 0 packets out, 0 packets in per second
```

```
TWEET_$
```

Note the second filter (the “permit IP” filter) that is shown. If there are currently no filters set for an interface when the filter server determines it needs to set a filter, it will add an explicit “permit IP” filter. This is done because the existence of any filter in a list of filters causes VSI TCP/IP to act as if a “deny everything” filter terminates the list. The “permit IP” filter will essentially prevent that problem from happening.

17.7. Filter Reporting via SNMP

When logging a filter via SNMP, the configuration keywords *`ENTERPRISE_STRING`*, *`GENERIC_TRAP_ID`* and *`SPECIFIC_TRAP_ID`* must be specified (as well as the keyword *`LOG_TO_SNMP`*). In addition, the SNMP configuration file must be properly set up on the VSI TCP/IP system.

When a filter is logged, the following fields will be reported:

FILTER_SERVER: Filter queued on *`interface`* (*`address`*) at *`time`*

COMPONENT=*`component-name`*

RULE=*`rulename`*

ACTION=*`actionname`* (e.g., “deny”)

PROTOCOL=*`protocol`* (e.g., “TCP”)

SOURCE=*`source address in CIDR format`*

SOURCE_PORT=*`operator port`*(e.g., “EQ 22”)

DESTINATION=*`destination address in CIDR format`*

DEST_PORT=operator *`port`* (e.g., “EQ 22”)

START=*`VMS absolute time`*

END=*`VMS absolute time`*

17.8. Correcting a Filter List

If a filter is inadvertently created by the filter server, the system manager should first correct the configuration problem (if one exists) that allowed the filter to be incorrectly set. Then, the system manager may retrieve the current list of filters in “manual filter form” that can be edited then reloaded onto the interface. The list is retrieved via the **IP SHOW/INTERFACE/EXTRACT_FILTER** command. For example:

```
TWEET_$ IP show/int se0/filt
Device se0: flags=8863<UP,BROADCAST,NOTRAILERS,RUNNING,MULTICAST,D2>
 VMS Device = EWA0
 IP Address = 192.168.0.16
 No common links defined
VSI TCP/IP for OpenVMS Packet Filter List for se0:
Logging is disabled
```

```
   Source Address / Port
Action   Proto   Hits    Destination Address / Port
------   -----   -----   ----------------------------------------
Deny             tcp     0   192.168.0.11/32
                 192.168.0.0/24 eq 22
                 FLTSVR,LOG
         START: 16-MAY-2017 10:33:19 END: 16-MAY-2017 10:38:19
Deny     tcp     15      192.168.0.38/32
                         192.168.011/24 eq 22
Permit   IP              13484  0.0.0.0/0
         0.0.0.0/0
         FLTSVR
 Average 0 bytes out, 0 bytes in per second
 Average 0 packets out, 0 packets in per second
TWEET_$ IP show/interface se0/extract_filter=filter.txt
TWEET_$ type filter.txt
#
# FILTER.TXT
#
# Generated 16-MAY-2017 10:51:31
#
#====================================================================
deny tcp 192.168.0.100/32 192.168.0.11/24 eq 22 start “16-MAY-2017
 10:33:19” end “16-MAY-2017 10:38:19”LOG
deny tcp 192.168.0/32.192.168.0.11/24
permit ip 0.0.0.0/32 0.0.0.0/32
TWEET_$ <edit to remove the first (filter) line>
TWEET_$ IP set/interface se0/filter=filter.dat
TWEET_$ IP show/interface se0/filt
Device se0: flags=8863<UP,BROADCAST,NOTRAILERS,RUNNING,MULTICAST,D2>
 VMS Device = EWA0
 IP Address = 192.168.0.16
 No common links defined
VSI TCP/IP for OpenVMS Packet Filter List for se0:
Logging is disabled
  Source Address / Port
Action   Proto   Hits    Destination Address / Port
------   -----   -----   ----------------------------------------
Deny             tcp     15   192.168.0.38/32
         192.168.011/24 eq 22
Permit   IP      13484   0.0.0.0/0
         0.0.0.0/0
 Average 0 bytes out, 0 bytes in per second
 Average 0 packets out, 0 packets in per second
TWEET_$
```

17.9. Configuring PMDF to use IPS on VSI TCP/IP

The IMAP, POP3 and SMTP servers referred to in the configuration template files above refer to the VSI TCP/IP servers only. Beginning with PMDF V6.4, PMDF has been instrumented to use IPS. The following PMDF template files are available in the `PMDF_TABLE` directory:

```
FILTER_SERVER_PMDF_IMAP.TEMPLATE
```

```
FILTER_SERVER_PMDF_POP3.TEMPLATE
```

`FILTER_SERVER_PMDF_SMTP.TEMPLATE`

These files should be copied to `IP$:*.TXT` and modified as appropriate for your installation. Edit `IP$:FILTER_SERVER_CONFIG.TXT` to add INCLUDE lines for these component files, and comment out the INCLUDE lines for the standard VSI TCP/IP IMAP, POP and SMTP files.

Next, make sure the appropriate PMDF images are installed. The legacy IMAP and POP servers (`PMDF_EXE:IMAPD.EXE` and `PMDF_EXE:POP3D.EXE`) are already installed. The msgstore IMAP and POP servers (`PMDF_EXE:IMAP_SERVER.EXE`, `PMDF_EXE:POP_SERVER.EXE`), as well as the SMTP server (PMDF_EXE:TCP_SMTP_SERVER.EXE) are not installed, so they will need to be added to your `PMDF_COM:PMDF_SITE_STARTUP.COM` file if your PMDF installation uses them. These must all be installed using the /OPEN qualifier.

At this point, define the logical name `PMDF_DO_FILTER_SERVER` to 1, using the **/SYSTEM** qualifier (this can be put in `PMDF_COM:PMDF_SITE_STARTUP.COM` as well).

Finally, restart IPS via the **IP SET/IPS/RESTART** command.

Note

The `PMDF_TABLE:FILTER_SERVER_PMDF_POP3.TEMPLATE` file as distributed in PMDF V6.4 has the incorrect port number of 143 specified. Make sure port 110 is specified in this file.

17.10. Controlling the Filter Server

The filter server is started at system startup time. However, it can be controlled using the **IP SET /IPS** command. The valid commands and their uses are:

Table 17.3. Filter Server Control Commands

Command	Description
/DEBUG_LEVEL=*`level`*	Change the debug level for the server. See the description for the *`debug`* main configuration keyword.
/CLEAR_FILTERS	Causes the `FILTER_SERVER` process to remove all filters set by IPS on all interfaces configured for IPS. This may be used with **SET /IPS /START** and **SET /IPS/RESTART**, or may be used by itself with **SET /IPS/CLEAR_FILTERS**. When used by itself this causes a running IPS subsystem to remove the IPS filters and reset the event count information for the source address associated with each filter being removed.
/RELOAD	Re-read and parse the configuration files. Note that this will not wipe out existing event and rule information; it will simply update it so no potential filter information will be lost.
/RESTART	Stop and restart the filter server. All existing event and rule information will be lost and reloaded from the configuration files.
/START	Start the filter server if it is not already running.
/STOP	Stop the filter server from running. All existing event and rule information will be lost.

The current configuration of the filter server may also be displayed using the **IP SHOW/IPS / CONFIG=*filename*** command. For example:

```
$ IP show/ips/config=server_stats.out
$ type server_stats.out
Filter server snapshot
Debug level 6
Block at destination port or system: PORT
Log to: OPCOM  SNMP trap
Component ssh
 Rule ssh_bogus_id
  dest address: 192.168.0.16/32
  interface: se0
  max event count: 10
  delta time: 0 00:01:30.00
  filter durations: 300 600 1800 3600 -1
  Address 192.168.0.11/32
   number of still-queued events: 1
   number of filters created: 0
   Address entry to be deleted: N/A
   Event
   event time: 29-APR-2017 10:00:12.41
   expires:  29-APR-2017 10:01:42.41
 Rule ssh_authfailed
  dest address: 192.168.0.16/32
  interface: se0
  max event count: 10
  delta time: 0 00:01:30.00
  filter durations: 300 600 1800 3600 -1
 Rule ssh_userauth
  dest address: 192.168.0.16
  interface: se0
  max event count: 10
  delta time: 0 00:01:30.00
  filter durations: 300 600 1800 3600 -1
 Rule ssh_invaliduser
  dest address: 192.168.0.16/32
  interface: se0
  max event count: 10
  delta time: 0 00:01:30.00
  filter durations: 300 600 1800 3600 -1
 Rule ssh_invalid_id_msg
  dest address: 192.168.0.16/32
  interface: se0
  max event count: 5
  delta time: 0 00:02:00.00
  filter durations: 300 600 1800 3600 -1
```

17.11. Filter Server Files

The following files are associated with the filter server:

`IP$:FILTER_SERVER_HOURLY_LOG.`*`yyyymmdd`*

This file is an hourly activity log for the filter server. The file extension changes daily at midnight to reflect the current day. What follows is a sample log segment for one hour:

```
Filter server hourly snapshot for hour 2 of 05/18/2017
```

```
Component ssh
 Rule ssh_bogus_id
  number of hits 0
 Rule ssh_authfailed
  number of hits 0
 Rule ssh_userauth
  number of hits 0
 Rule ssh_invaliduser
  number of hits 2
  Address 192.168.0.10/32
   number of still-queued events:   0
   number of all events:            0
   number of filters created:       1
   Address entry to be deleted:     18-MAY-2017 05:55:45.45
  Address 192.168.0.204
   number of still-queued events:   0
   number of all events:            2
   number of filters created:       0
   Address entry to be deleted:     18-MAY-2017 06:22:03.97
```

This log is showing that during the hour 01:00-02:00, 2 different source addresses were being tracked by the filter server:

The first address (192.168.0.10) had a filter created sometime in the last 4 hours (the time it takes an address to have no activity before its records are deleted by the filter server). The log indicates the address entry is to be deleted in 3 hours if there is no more activity; therefore, the filter was actually set in the previous hour (looking at the previous hour's entry in the log file will confirm this).

The second address (192.168.0.204) had 2 events during the hour that never triggered a filter and were deleted after they aged. This address entry is scheduled to be deleted in 4 hours if there is no more activity for it.

`IP$:FILTER_SERVER_CONFIG.TXT`

This is the main filter server configuration file. Optionally, the server will use the logical name `FILTER_SERVER_CONFIG` to determine the name of the main configuration file.

`IP$:FILTER_SERVER.OUT`

This file contains any output resulting from starting the filter server (e.g., the output from any DCL commands executed to start it) and all debug messages.

17.12. Instrumenting a User-Written Application with IPS

When instrumenting an application (aka, a component), there are several steps to be followed:

- The user determines the component-specific parameters. These include:
  - The prototype filter to be used when a filter is created. This is the same format as that used when using a filter file. All filter features are supported, with the exception of the *ESTABLISHED* and *REPEATING* keywords. Note that the source address/mask/port and destination address/mask/port fields of the filter will be overwritten during creation of the filter, according to the other parameters set in the configuration file.

- Whether the filters created will block at the destination port or simply block all traffic from the source system (the *BLOCK_AT_DESTINATION_PORT* keyword).
- The logging to be used.

- The user determines the ruleset:
  - The user determines what rules are to be supported. There is no limit on the number of rules the filter server can maintain; the limit is really on how complex you want to make the component.
  - For each rule, you need to determine:
    - The name of the rule. This string (maximum length 35 characters) will be used by the filter server and by the call to the filter server API call `send_filter_event`.
    - The number of events/unit time that will trigger a filter (the *MAX_COUNT* and *DELTA_TIME* fields).
    - The duration(s) of a filter. Up to 5 may be chosen, and the list must end with -1.
- The user creates the component-specific configuration file, then adds a reference to it via the *INCLUDE* keyword in the main filter server configuration file. At this point, the filter server can be made aware of this new (or updated) configuration by using the **IP SET /IPS/RELOAD** command.

Note

The filter server configuration may be reloaded multiple times without causing problems for the filter server.

17.13. Filter Server API

There are two calls available in the filter server API. The function prototypes are defined in `IP $COMMON_ROOT:[IP.INCLUDE]FILTER_SERVER_API.H`. The first call is used to register with the filter server:

```
int filter_server_register(char *component, 0, 0)
```

where

`component` is the name of the component

The remaining two arguments are there for future expansion, and are ignored, but must be specified.

The return values from this function are 1 (success) or 0 (an error occurred; most likely, this is because the filter server isn't running). Normally this function is called once when the first event is logged. However, if an error is returned, it may be called again when additional events are logged.

Note

The application that is registering MUST be an installed image, using **/OPEN** or **/SHARED**. It does not need to be installed with privileges. This is an attempt to help cut down on bogus applications registering with the server; it takes a conscious effort - and privileges - by the system manager to do this and therefore, to control this.

The next function is used to format and send events to the filter server:

```
int send_filter_event(char *rule,
  char *source_address,
  u_short source_port,
  char *dest_address)
```

where

rule is the name of the rule to be enforced. This must correspond to a *rule* keyword specified in the per-component configuration file for the component. If a match cannot be made, the event will be ignored by the filter server.

source_address is the address of the system that caused the event to be logged (e.g., “192.168.0.1”). This may be in ipv4 or ipv6 format, but must be of the same address family as that of the *destination_address* specified for the component in the per-component configuration file. Note that this is an address only. Do not specify address mask bits (e.g., “192.168.0.1/24”) with it.

source_port is the source port on the originating system.

dest_address is the destination address of the socket used to communicate to *source_address*. This information may be obtained by performing a *getsockname* function on the socket. Note that this is an address only. Do not specify address mask bits (e.g., “192.168.0.11/24”) with it.

To include these routines in your application, link using the library `IP$COMMON_ROOT:[IP.LIBRARY]FILTER_SERVER_API.OLB`.

The following is an example of code used to send events to the filter server:

```
void ssh_send_filter_event(int code, char *addr, int port, char *dest_addr)
{
 char *rule;
 static int filter_server = -1;
if (filter_server == -1)
filter_server = filter_server_register("SSH", 0, 0);
if (!filter_server)
return;
 switch(code)
 {
case LGI$_NOSUCHUSER:
case LGI$_NOTVALID:
rule = "SSH_INVALIDUSER";
break;
case LGI$_USERAUTH:
rule = "SSH_USERAUTH";
break;
case LGI$_DISUSER:
case LGI$_ACNTEXPIR:
 case LGI$_RESTRICT:
 case LGI$_INVPWD:
  case LGI$_PWDEXPIR:
  rule = "SSH_AUTHFAILED";
  break;
  default:
  printf("Unrecognized status code %d", code));
  return;
 }
 send_filter_event(rule, addr, (unsigned short)port, dest_addr);
}
```

Chapter 18. Configuring DECnet-over-IP Circuits

A special DECnet device driver allows the VSI TCP/IP system manager to configure a DECnet line and circuit between two cooperating VSI TCP/IP systems across an arbitrary IP network. This special driver encapsulates DECnet packets in UDP datagrams for transport via the IP protocols, in much the same way as OpenVMS PSI encapsulates DECnet packets in X.25 when doing Data Link Mapping.

18.1. Using the Configuration Tools

VSI TCP/IP provides one tool for configuring DECnet-over-IP connections: DECNET-CONFIG. The command-line configuration utility invoked with the **IP CONFIGURE /DECNET** command.

For details about these utilities, see the *VSI TCP/IP Administrator's Reference*, and the online command reference (invoked with the **HELP IP** command).

Once configured, a DECnet-over-IP circuit comes up automatically when the hosts on each side of the DECnet connection are rebooted.

When configuring a DECnet-over-IP circuit, you are prompted for:

- The IP address of the host on the opposite side of the connection
- The COST that DECnet assigns to the circuit
- The HELLO TIMER that DECnet should use on this circuit

For proper DECnet operation, the COST and HELLO TIMER must be the same on both sides of the connection.

Note

If you configure a DECnet-over-IP link, and you also run DECnet over another Ethernet interface, you must configure your system as a router as follows:

```
 $ RUN SYS$SYSTEM:NCP
NCP>DEFINE EXECUTOR TYPE ROUTING IV
```

Non-routing hosts can only communicate with the hosts reachable via the circuit with the lowest cost. If, for example, your Ethernet circuit has a cost of 4, and your DECnet-over-IP link has a cost of 1 (the default), your system will be unable to communicate with hosts accessible over the Ethernet link.

18.2. Examples of Connecting Two Systems

The following examples show how to make a connection between two systems, betty.urub.edu (IP address 192.0.0.6) and wilma.fstone.com (IP address 128.0.0.125). The first example shows the circuit configuration on the host "betty":

```
$ IP CONFIGURE /DECNET
VSI TCP/IP for OpenVMS DECNET Circuit Configuration Utility 10.5(nnn)
[Reading in configuration from IP$:DECNET-CIRCUITS.COM]
DECNET-CONFIG>ADD
```

```
[Adding new configuration entry for DECnet circuit "TCP-0-0"]
Destination IP Address: [NONE] 128.0.0.125
DECnet circuit cost: [1] 1
DECnet hello timer (in seconds): [300] 300
[TCP-0-0 => 128.0.0.125 (Cost=1, Hello Timer=300)]
DECNET-CONFIG>EXIT
[Writing configuration to IP$:DECNET-CIRCUITS.COM]
$
```

The next example shows the circuit configuration on the host "wilma":

```
$ IP CONFIGURE /DECNET
VSI TCP/IP for OpenVMS DECNET Circuit Configuration Utility 10.5(nnn)
[Reading in configuration from IP$:DECNET-CIRCUITS.COM]
DECNET-CONFIG>ADD
[Adding new configuration entry for DECnet circuit "TCP-0-0"]
Destination IP Address: [NONE] 192.0.0.6
DECnet circuit cost: [1] 1
DECnet hello timer (in seconds): [300] 300
[TCP-0-0 => 192.0.0.6 (Cost=1, Hello Timer=300)]
DECNET-CONFIG>EXIT
[Writing configuration to IP$:DECNET-CIRCUITS.COM]
$
```

18.3. DECnet Encapsulation Over Unreliable Networks

Both TCP and DECnet guarantee reliable delivery of data over unreliable networks. This is accomplished through an acknowledgment scheme in which the receiver of the data tells the transmitter of the data that the data has arrived intact. If the acknowledgment is not received within a certain period of time (known as the *`retransmission timer`*), the data is resent by the transmitter.

The data transmitter must make a good estimate of the retransmission timer. Too long an interval causes unnecessary waits before retransmission occurs, reducing the usable bandwidth of the network. Too short an interval means the transmitter might retransmit data that was merely delayed in transit, unnecessarily loading the network.

- TCP chooses the retransmission timer as a function of the mean and the variance in the *`roundtrip time`* so that a statistically small percentage of packets are unnecessarily retransmitted.
- DECnet chooses the retransmission timer as the product of the round trip time (with a minimum of one second) and the *delay factor*.

The method used by DECnet does not take into account the variance of the round-trip time and estimated roundtrip times of less than one second.

DECnet uses a very conservative value for the delay factor to avoid any unnecessary retransmissions into congested, low-speed links. A single lost packet results in a delay of at least five seconds in DECnet traffic. Over high-speed, low-latency circuits with any substantial packet loss, this delay results in a severe performance degradation.

If your network has these characteristics, you can substantially increase performance by reducing the delay factor using NCP on each of your nodes. Doing so gives DECnet a more aggressive retransmission strategy, which results in shorter delays following a lost packet. Specify the delay factor in units of 1/16th of the mean round-trip time using the NCP EXECUTOR parameter DELAY FACTOR.

Reasonable factors range from 1.5 to 3, or DELAY FACTOR values from 24 to 48. A retransmission factor of 1.5 is very aggressive and about as small as is reasonable before many extra retransmissions occur; a value of 3 more closely mimics TCP's behavior over lines which have typical variances in the roundtrip time.

You can set the DELAY FACTOR to 24/16ths (1.5) using the following NCP commands:

```
$ MCR NCP
NCP>SET EXECUTOR DELAY FACTOR 24
NCP>DEFINE EXECUTOR DELAY FACTOR 24
NCP>EXIT
$
```

18.4. Using IP SET /DECNET

Use the **IP SET /DECNET** command to configure the TCPA*x*: DECnet devices for running DECnet over UDP.

Note

You should configure DECnet circuits using DECNET-CONFIG, which invokes IP SET /DECNET as part of network startup to set up the DECnet link. You can use this utility to change the configuration once the network has started.

Appendix D. How NFS Converts File Names

The NFS to OpenVMS file name translation rules in the Table D.1 are based on the character mapping scheme in Table D.2. The OpenVMS to NFS mapping rules are the converse of these rules.

Table D.1. NFS Server to OpenVMS Client File Name Conversion Rules

Rule	What Happens to File Names from NFS to OpenVMS
1	Lowercase characters become uppercase (unless Rule 2 applies). For example, file becomes FILE.;1
2	Initial uppercase characters or a sequence of case-shifted characters are prefixed with the "$" escape character. For example, CaseShiftedFile becomes $C$ASE$S$HIFTED$F$ILE.;1
3	A file without a version gets a version number preceded by a semicolon. For example, file becomes FILE.;1
4	If a file name does not include a dot (.), a dot is added before the version number semicolon. For example, file becomes FILE.;1
5	After its name is converted, a file will not appear in an OpenVMS directory listing if any one of the following criteria are met: • The file name is more than 39 characters long. • The file extension is more than 39 characters long. • The version number is greater than 32767.
6	If the file name has a dot, the dot is preserved unless the resulting file name fails one of the tests in Rule 5; if so, the dot becomes "$5N" and the same rule applies to each subsequent dot found. For example, more.file.text becomes MORE.FILE$5NTEXT;1
7	If the file name is a directory, each dot becomes "$5N" and the file name gets the ".DIR" extension. For example, dot.directory.list becomes DOT $5NDIRECTORY$5NLIST.DIR;1
8	Invalid OpenVMS characters become the escape character sequences in the second column of Table D.2 ("$" followed by a digit and a letter). For example, special#character&file becomes SPECIAL$5CCHARACTER $5FFILE.;1 ("#" becomes "$5C" and "&" becomes "$5F")
9	Any existing "$" becomes "$$" (plus any "$" added due to Rule 2 or 8 above). For example, dollar$Sign$5cfile becomes DOLLAR$$$S$IGN$$5CFILE.;1

Table D.2 provides a complete list of OpenVMS character sequences, corresponding server characters, and octal values used for NFS name conversion.

Table D.2. NFS Client Name Conversion

OpenVMS Character Sequence	Server Character	Octal Value
$6A	<CTRL/@>	000

OpenVMS Character Sequence	**Server Character**	**Octal Value**
$4A	<CTRL/A>	001
$4B	<CTRL/B>	002
$4C	<CTRL/C>	003
$4D	<CTRL/D>	004
$4E	<CTRL/E>	005
$4F	<CTRL/F>	006
$4G	<CTRL/G>	007
$4H	<CTRL/H>	010
$4I	<CTRL/I>	011
$4J	<CTRL/J>	012
$4K	<CTRL/K>	013
$4L	<CTRL/L>	014
$4M	<CTRL/M>	015
$4N	<CTRL/N>	016
$4O	<CTRL/O>	017
$4P	<CTRL/P>	020
$4Q	<CTRL/Q>	021
$4R	<CTRL/R>	022
$4S	<CTRL/S>	023
$4T	<CTRL/T>	024
$4U	<CTRL/U>	025
$4V	<CTRL/V>	026
$4X	<CTRL/W>	027
$4X	<CTRL/X>	030
$4Y	<CTRL/Y>	031
$4Z	<CTRL/Z>	032
$6B	<CTRL/[>	033
$6C	<CTRL/\>>	034
$6D	<CTRL/]>	035
$6E	<CTRL/^>	036
$6F	<CTRL/_>	037
$7A	<SPACE>	040
$5A	!	041
$5B	"	042
$5C	#	043
$5E	%	045
$5F	&	046

OpenVMS Character Sequence	Server Character	Octal Value
$5G	'	047
$5H	(	050
$5I	)	051
$5J	*	052
$5K	+	053
$5L	,	054
$5N	.	056
i$5O	/	057
$5Z	:	072
$7B	;	073
$7C	<	074
$7D	=	075
$7E	>	076
$7F	?	077
$8A	@	100
$8B	[	133
$8C	\	134
$8D	]	135
$8E	^	136
$9A	`	140
$9B	{	172
$9C	|	174
$9D	}	175
$9E	~	176
$9F	<DEL>	177

OpenVMS Character Sequence	Server Character	Octal Value

Appendix E. DNSSEC

Cryptographic authentication of DNS information is possible through the DNS Security (DNSSEC-bis) extensions, defined in RFC 4033, RFC 4034, and RFC 4035. This section describes the creation and use of DNSSEC signed zones.

In order to setup a DNSSEC secure zone, there are a series of steps, which must be followed. BIND9 ships with several tools that are used in this process, which are explained in more detail below. In all cases, the -h option prints a full list of parameters.

Note

For use with VSI TCP/IP 10.5, define symbols for each of the tools, and call the symbol from the command line. For example, to use the dnssec-keygen tool, a symbol could be created as follows:

```
$ keygen :== $IP$:dnssec-keygen
$ keygen -h
```

There must also be communication with the administrators of the parent and/or child zone to transmit keys. A zone’s security status must be indicated by the parent zone for a DNSSEC capable resolver to trust its data. This is done through the presence or absence of a DS record at the delegation point.

For other servers to trust data in this zone, they must either be statically configured with this zone’s zone key or the zone key of another zone above this one in the DNS tree.

E.1. Generating Keys

The **dnssec-keygen** program is used to generate keys. A secure zone must contain one or more zone keys. The zone keys will sign all other records in the zone, as well as the zone keys of any secure delegated zones. Zone keys must have the same name as the zone, a name type of ZONE, and must be usable for authentication. It is recommended that zone keys use a cryptographic algorithm designated as “mandatory to implement” by the IETF; currently the only one is RSASHA1. For convenience, run the **dnssec-keygen** tool in the directory the zone data files are located, do you won’t need to use full pathnames as arguments.

The following command will generate a 768-bit RSASHA1 key for the child.example zone, the symbol `keygen` has been created to refer to the **dnssec-keygen** executable:

```
$ keygen -a RSASHA1 -b 768 -n ZONE child-example
```

Note

File names specified with the tools must conform to OpenVMS naming conventions. Be aware of using multiple dots, etc. which will generate errors upon file creation.

Two output files will be produced: `Kchild-example-005-12345.key` and `Kchild-example-005-12345.private` (where 12345 is an example of a key tag). The key filenames contain the key name (`child-example.`), algorithm (3 is DSA, 1 is RSAMD5, 5 is RSASHA1, etc.), and the key tag (12345 in this case). The private key (in the `.private` file) is used to generate signatures, and the publickey (in the `.key` file) is used for signature verification.

Note

Always protect the `.private` key, anyone who knows it can forge signed zone data. The `.private` key file will be written readable and writable only by the user who runs it. VSI recommends running the DNSSEC tools from a suitably privileged account.

To generate another key with the same properties (but with a different key tag), repeat the above command.

The **dnssec-keyfromlabel** program is used to get a key pair from a crypto hardware and build the key files. Its usage is similar to **dnssec-keygen**.

The publickeys can be inserted into the zone file by pasting in their contents, or better yet by including the `.key` file using `$include` statements. For example, to insert the publickey for child-example, add the following `$include` statement to the zone file:

```
$ include Kchild-example-005-12345.key ;
```

The zone file (for examples in this Appendix, the file name is zone.1) may look like this:

```
$TTL 100
$ORIGIN child-example.
@ IN SOA a.example. a.a.example. 1 360 36 60480 12
 NS a.example.
 NS b.example.
one  A 10.10.10.10
two  A 10.10.10.100
 MX 10 one.zz.example.
$include Kchild-example-005-12345.key ;
```

E.2. Signing the Zone

With the key included in the zone file, use the **dnssec-signzone** program to sign the zone.

Any keyset files corresponding to secure subzones should be present. The zone signer will generate NSEC, NSEC3 and RRSIG records for the zone, as well as DS for the child zones if ‘-g’ is specified. If ‘-g’ is not specified, then DS RRsets for the secure child zones need to be added manually.

The following command signs the zone, assuming it is in a file called `zone.1`. By default, all zone keys which have an available private key are used to generate signatures. First, define a symbol for dnssec-signzone:

```
$ signer :== $IP$:dnssec-signzone
```

The -o option specifies the zone origin, the default is the zone file name.

```
$ signer -o child-example zone.1
```

Note

You may see the message “No self signing KSK found.” This is normal, as no KSK (key signing key) has been generated at this point. Only a ZSK (zone signing key) is present.

One output file is produced: `zone.1_signed`.

This file should be referenced by `named.conf` as the input file for the zone.

The output file, zone.1_signed,shown in the following example:

```
; dnssec_signzone version 9.7.2-p3
child-example. 100 IN SOA a.example. a.a.example. (
 1 ; serial
 360 ; refresh (6 minutes)
 36 ; retry (36 seconds)
 60480 ; expire (16 hours 48 minutes)
 12 ; minimum (12 seconds)
 )
100  RRSIG SOA 5 1 100 20110428114855 (
 20110329114855 36111 child-example.
 rWVs/euooBTVk0MzhxHQio61rDBhzAId13sV
 KXphVsA64bqyayhJcCfikmxww6vq6gG0W3mR
 z1tbIQ7znZ0SN90dsWhEcoEaEmm1Sl6hwSVY
 OzaYrN8HgahzcrNlsX5l )
100  NS a.example.
100  NS b.example.
100  RRSIG NS 5 1 100 20110428114855 (
 20110329114855 36111 child-example.
 SOrA8BihARhE+SPl/iYjB8PTqk+8lc4sEE4b
 CYhgcF6d9VOZtCotQFUqVKrk65xoGqf60+9R
 kBJr6lsOwr6mqDVCiZzVnAy1frWD8T8q5HNK
 nzVR8gb7AXyPtbgKqOS3 )
12  NSEC one.child-example. NS SOA RRSIG NSEC DNSKEY
12  RRSIG NSEC 5 1 12 20110428114855 (
 20110329114855 36111 child-example.
 L0K9USccXSgO4iYBaXDOrQ0zzrxVVRECwjAb
 DAeZqVec525V6kNIB5F2mCxjSJqlJ5C40vr+
 lCqe/EGzjxplEzqq0nSN/fCtTgXqhLL6EfZx
 M1lvB5C+4K4hR20neVWy )
100  DNSKEY 256 3 5 (
 AwEAAcXIK+ljUWgMENcS9TUqnZGEFMOE5DBP
 WyQu5aIGSZqTTcvMWsaFtS7800LjapDB4kcs
 xwecfdA4I/0dUHPuHqmQREGfq/xstyxLPHKS
 MEkJthkVurf4MWzdX8dAVEd/GQ==
 ; key id = 36111
100  RRSIG DNSKEY 5 1 100 20110428114855 (
 20110329114855 36111 child-example.
 O8t91OOvLCSotc7mTG7iVr6fyeg7AA6ZuzHR
 GfN0dbOFzZHGxSAj2pRXPz8FC/eYz+ngy6rK
 23UhdklmuJN35IEA+qkXBilS7NJtEvaONOud
 1ANN6qQDtXyYFxnCuEN0 )
one.child-example. 100 IN A 10.10.10.10
100  RRSIG A 5 2 100 20110428114855 (
 20110329114855 36111 child-example.
 o3TPUffd5dLuxoac0TVVsT8HU3MFoJtIbfXV
 apidfBY7IbxU6YWgPPwkYO1oKgJ3CnWmKTZQ
 sUB+QRE1VHn8GmPbyjbg9QfhIKZDEQyT2f7x
 41QDNznnKnJyYjhmbyCf )
12  NSEC two.child-example. A RRSIG NSEC
12  RRSIG NSEC 5 2 12 20110428114855 (
 20110329114855 36111 child-example.
 w3RXqBeiUk/njCh/nHg2s1hv9kYynGdRsp2A
 vYm8ahrq4pGv1DLr6uuwCT5vBfjor1l5ePBj
 jsIO3FLkWyO7miBpfiLLPa7umKSQLN0AZGIE
 /5Z7LSc80o2fzwqcBkub )
two.child-example. 100 IN A 10.10.10.100
100  RRSIG A 5 2 100 20110428114855 (
 20110329114855 36111 child-example.
 jQAof31o6bO4oOVlhLAt6NQkifz1l4qnfN4a
 viZiB0RmLYuRNnHFRAPyZLkoI8PTgCuCdV/e
 co1ifFnXU9UauNnK/wQw8Djurvra/YMq8f5W
 ZZcOReQvZUoD8mS4C3ec )
100  MX  10 one.zz.example.
100  RRSIG MX 5 2 100 20110428114855 (
 20110329114855 36111 child-example.
 hIQI20XS9qYdi5/3qMp1VeU0aQqBwQsugwkw
 mCD9gY7BrpYjMeeg3XQHY0Qx7ElqLc9Q0F3C
 kC0ETM5CDnUAicXCy2TOc1DAKfSOYlKRnzVd
 a5LlFGymsi2gVyW7VssH )
12  NSEC child-example. A MX RRSIG NSEC
12  RRSIG NSEC 5 2 12 20110428114855 (
 20110329114855 36111 child-example.
 OhIM8y6IGXixOUtD+ZH/bicznRtX6YrdeXxg
 5bD3ROSUcfpCL5YAUxfk/B9nj2n1OStle88r
 O7EeMB2rSiAPqYW88ZbIXXhOHsE6z3ff7Plc
 B3pT56MBxUh5cm2WDYTL )
```

dnssec-signzone will also produce a keyset and dsset files and optionally a dlvset file. These are used to provide the parent zone administrators with the DNSKEYs (or their corresponding DS records) that are the secure entry point to the zone.

E.3. Configuring Servers

To enable **named** to respond appropriately to DNS requests from DNSSEC aware clients, the option **dnssec-enable** must be set to yes. (This is the default setting.)

To enable **named** to validate answers from other servers, the **dnssec-enable** and **dnssec-validation** options must both be set to yes, and at least one trust anchor must be configured with a **trusted-keys** or **managed-keys** statement in `named.conf`.

trusted-keys are copies of DNSKEY RRs for zones that are used to form the first link in the cryptographic chain of trust. All keys listed in **trusted-keys** (and corresponding zones) are deemed to exist and only the listed keys will be used to validated the DNSKEY RRset that they are from.

managed-keys are trusted keys which are automatically kept up to date via RFC 5011 trust anchor maintenance.

After DNSSEC is established, a typical DNSSEC configuration will look something like the following. It has one or more publickeys for the root. This allows answers from outside the organization to be validated. It will also have several keys for parts of the namespace the organization controls. These are here to ensure that named is immune to compromises in the DNSSEC components of the security of parent zones.

```
managed-keys {
/* Root Key */
 "." initial-key 257 3 3

"BNY4wrWM1nCfJ+CXd0rVXyYmobt7sEEfK3clRbGaTwS
JxrGkxJWoZu6I7PzJu/E9gx4UC1zGAHlXKdE4zYIpRh
aBKnvcC2U9mZhkdUpd1Vso/HAdjNe8LmMlnzY3zy2Xy
4klWOADTPzSv9eamj8V18PHGjBLaVtYvk/ln5ZApjYg
hf+6fElrmLkdaz MQ2OCnACR817DF4BBa7UR/beDHyp
5iWTXWSi6XmoJLbG9Scqc7l70KDqlvXR3M/lUUVRbke
g1IPJSidmK3ZyCllh4XSKbje/45SKucHgnwU5jefMtq
66gKodQj+MiA21AfUVe7u99WzTLzY3qlxDhxYQQ20FQ
97S+LKUTpQcq27R7AT3/V5hRQxScINqwcz4jYqZD2fQ
dgxbcDTClU0CRBdiieyLMNzXG3";
};

trusted-keys {
/* Key for our organization’s forward zone */
example.net. 257 3 5
"AwEAAaxPMcR2x0HbQV4WeZB6oEDX+r0QM6
5KbhTjrW1ZaARmPhEZZe3Y9ifgEuq7vZ/z
GZUdEGNWy+JZzus0lUptwgjGwhUS1558Hb
4JKUbbOTcM8pwXlj0EiX3oDFVmjHO444gL
kBOUKUf/mC7HvfwYH/Be22GnClrinKJp1O
g4ywzO9WglMk7jbfW33gUKvirTHr25GL7S
TQUzBb5Usxt8lgnyTUHs1t3JwCY5hKZ6Cq
FxmAVZP20igTixin/1LcrgX/KMEGd/biuv
F4qJCyduieHukuY3H4XMAcR+xia2nIUPvm
/oyWR8BW/hWdzOvnSCThlHf3xiYleDbt/o
1OTQ09A0=";

/* Key for our reverse zone. */

2.0.192.IN-ADDRPA.NET. 257 3 5
"AQOnS4xn/IgOUpBPJ3bogzwc
xOdNax071L18QqZnQQQAVVr+i
LhGTnNGp3HoWQLUIzKrJVZ3zg
gy3WwNT6kZo6c0tszYqbtvchm
gQC8CzKojM/W16i6MG/eafGU3
siaOdS0yOI6BgPsw+YZdzlYMa
IJGf4M4dyoKIhzdZyQ2bYQrjy
Q4LB0lC7aOnsMyYKHHYeRvPxj
IQXmdqgOJGq+vsevG06zW+1xg
YJh9rCIfnm1GX/KMgxLPG2vXT
D/RnLX+D3T3UL7HJYHJhAZD5L
59VvjSPsZJHeDCUyWYrvPZesZ
DIRvhDD52SKvbheeTJUm6Ehkz
ytNN2SN96QRk8j/iI8ib";
};

options { ...
dnssec-enable yes;
dnssec-validation yes;
};
```

Note

None of the keys listed in this example are valid. In particular, the root key is not valid.

When DNSSEC validation is enabled and properly configured, the resolver will reject any answers from signed, secure zones which fail to validate, and will return SERVFAIL to the client.

Responses may fail to validate for any of several reasons, including missing, expired, or invalid signatures, a key that does not match the DS RRset in the parent zone, or an insecure response from a zone, which, according to its parent, should have been secure.

When the validator receives a response from an unsigned zone that has a signed parent, it must confirm with the parent that the zone was intentionally left unsigned by verifying or via signed and validated NSEC/NSEC3 records,because the parent zone contains no DS records for the child. If the validator can prove that the zone is insecure, then the response is accepted. However, if it cannot, then it must assume an insecure response to be a forgery; it rejects the response and logs an error. The logged error reads “insecurity proof failed” and “got insecure response; parent indicates it should be secure”.

E.4. DNSSEC, Dynamic Zones, and Automatic Signing

As of BIND 9.7.0 it is possible to change a dynamic zone from insecure to signed and back again. A secure zone can use either NSEC or NSEC3 chains.

E.4.1. Converting from insecure to secure

Changing a zone from insecure to secure can be done in two ways: using a dynamic DNS update, or the **auto-dnssec** zone option.

For either method, you need to configure named so that it can see the `K*` files which contain the public and private parts of the keys that will be used to sign the zone. These files will have been generated by **dnssec-keygen**. You can do this by placing them in the key-directory, as specified in `named.conf`:

```
zone example.net {
 type master;
 update-policy local;
 file "example.net";
 key-directory "IP$COMMON_ROOT:[IP]";
};
```

If one KSK and one ZSK DNSKEY key have been generated, this configuration will cause all records in the zone to be signed with the ZSK, and the DNSKEY RR set to be signed with the KSK as well. An NSEC chain will be generated as part of the initial signing process.

E.4.2. Dynamic DNS update method

To insert the keys via dynamic update:

```
$ nsupdate :== $IP$:nsupdate.exe
$ nsupdate
> ttl 3600
> update add example.net DNSKEY 256 3 7
 AwEAAZn17pUF0KpbPA2c7Gz76Vb18v0teKT3EyAGfBfL8eQ8al35zz3Y
> update add example.net DNSKEY 257 3 7
 AwEAAd/7odU/64o2LGsifbLtQmtO8dFDtTAZXSX2+
> send
```

While the update request will complete almost immediately, the zone will not be completely signed until named has had time to walk the zone and generate the NSEC and RRSIG records. The NSEC record at the apex will be added last, to signal that there is a complete NSEC chain.

If you wish to sign using NSEC3 instead of NSEC, you should add an NSEC3PARAM record to the initial update request. If you wish the NSEC3 chain to have the OPTOUT bit set, set it in the flags field of the NSEC3PARAM record.

```
$ nsupdate
> ttl 3600
> update add example.net DNSKEY 256 3 7
 AwEAAZn17pUF0KpbPA2c7Gz76Vb18v0teKT3EyAGfBfL8eQ8al35zz3Y
> update add example.net DNSKEY 257 3 7
 AwEAAd/7odU/64o2LGsifbLtQmtO8dFDtTAZXSX2+X3e/
> update add example.net NSEC3PARAM 1 1 100 1234567890
> send
```

Again, this update request will complete almost immediately; however, the record will not show up until named has had a chance to build/remove the relevant chain. A private type record will be created to record the state of the operation (see below for more details), and will be removed once the operation completes.

While the initial signing and NSEC/NSEC3 chain generation is happening, other updates are possible as well.

E.4.3. Fully automatic zone signing

To enable automatic signing, add the **auto-dnssec** option to the zone statement in named.conf. **auto-dnssec** has two possible arguments: *`allow`* or *`maintain.`*

With **auto-dnssec allow**, named can search the key directory for keys matching the zone, insert them into the zone, and use them to sign the zone. It will do so only when it receives an **rndc sign** ***`zonename`*** or **rndc loadkeys** ***`zonename`*** command.

auto-dnssec maintain includes the above functionality, but will also automatically adjust the zone’s DNSKEY records on schedule according to the keys’ timing metadata. If keys are present in the key directory the first time the zone is loaded, it will be signed immediately, without waiting for an **rndc sign** or **rndc loadkeys** command. (Those commands can still be used when there are unscheduled key changes, however.)

Using the **auto-dnssec** option requires the zone to be configured to allow dynamic updates, by adding an **allow-update** or **update-policy** statement to the zone configuration. If this has not been done, the configuration will fail.

E.4.4. Private-type records

The state of the signing process is signaled by private-type records (with a default type value of 65534). When signing is complete, these records will have a non-zero value for the final octet (for those records that have a non-zero initial octet).

The private type record format: If the first octet is non-zero then the record indicates that the zone needs to be signed with the key matching the record, or that all signatures that match the record should be removed.

algorithm (octet 1)

key id in network order (octet 2 and 3)

removal flag (octet 4)

complete flag (octet 5)

Only records flagged as “complete” can be removed via dynamic update. Attempts to remove other private type records will be silently ignored. If the first octet is zero (this is a reserved algorithm number that should never appear in a DNSKEY record) then the record indicates changes to the

NSEC3 chains are in progress. The rest of the record contains an NSEC3PARAM record. The flag field tells what operation to perform based on the flag bits.

0x01 OPTOUT

0x80 CREATE

0x40 REMOVE

0x20 NONSEC

E.5. DNSKEY rollovers

As within secure-to-secure conversions, rolling DNSSEC keys can be done in two ways: using a dynamic DNS update, or the **auto-dnssec** zone option.

E.5.1. Dynamic DNS update method

To perform key rollovers via dynamic update, you need to add the K* files for the new keys so that named can find them. You can then add the new DNSKEY RRs via dynamic update. **named** will then cause the zone to be signed with the new keys. When the signing is complete the private type records will be updated so that the last octet is non-zero.

If this is for a KSK you need to inform the parent and any trust anchor repositories of the new KSK.

You should then wait for the maximum TTL in the zone before removing the old DNSKEY. If it is a KSK that is being updated, you also need to wait for the DS RRset in the parent to be updated and its TTL to expire. This ensures that all clients will be able to verify at least one signature when you remove the old DNSKEY.

The old DNSKEY can be removed via UPDATE. Take care to specify the correct key. **named** will clean out any signatures generated by the old key after the update completes.

E.5.2. Automatic key rollovers

When a new key reaches its activation date (as set by **dnssec-keygen** or **dnssec-settime**), if the **auto-dnssec** zone option is set to maintain, named will automatically carry out the key roll over. If the key's algorithm has not previously been used to sign the zone, then the zone will be fully signed as quickly as possible. However, if the new key is replacing an existing key of the same algorithm, then the zone will be re-signed incrementally, with signatures from the old key being replaced with signatures from the new key as their signature validity periods expire. By default, this rollover completes in 30 days, after which it will be safe to remove the old key from the DNSKEY RRset.

E.5.3. NSEC3PARAM rollovers via UPDATE

Add the new NSEC3PARAM record via dynamic update. When the new NSEC3 chain has been generated, the NSEC3PARAM flag field will be zero. At this point, you can remove the old NSEC3PARAM record. The old chain will be removed after the update request completes.

E.5.4. Converting from NSEC to NSEC3

To do this, you just need to add an NSEC3PARAM record. When the conversion is complete, the NSEC chain will have been removed and the NSEC3PARAM record will have a zero flag field. The NSEC3 chain will be generated before the NSEC chain is destroyed.

E.5.5. Converting from NSEC3 to NSEC

To do this, use nsupdate to remove all NSEC3PARAM records with a zero flag field. The NSEC chain will be generated before the NSEC3 chain is removed.

E.5.6. Converting from secure to insecure

To convert a signed zone to unsigned using dynamic DNS, delete all the DNSKEY records from the zone apex using nsupdate. All signatures, NSEC or NSEC3 chains, and associated NSEC3PARAM records will be removed automatically. This will take place after the update request completes.

This requires the `dnssec-secure-to-insecure` option to be set to *`yes`* in `named.conf`.

In addition, if the `auto-dnssec maintain` zone statement is used, it should be removed or changed to *`allow`* instead (or it will re-sign).

E.5.7. Periodic re-signing

In any secure zone which supports dynamic updates, named will periodically re-sign RRsets which have not been re-signed as a result of some update action. The signature lifetimes will be adjusted to spread the re-sign load over time rather than all at once.

E.5.8. NSEC3 and OPTOUT

named supports creating new NSEC3 chains where all the NSEC3 records in the zone have the same OPTOUT state. **named** also supports UPDATES to zones where the NSEC3 records in the chain have mixed OPTOUT state. **named** does not support changing the OPTOUT state of an individual NSEC3 record, the entire chain needs to be changed if the OPTOUT state of an individual NSEC3 needs to be changed.

E.6. Dynamic Trust Anchor Management

BIND 9.7.0 introduces support for RFC 5011, dynamic trust anchor management. Using this feature allows named to keep track of changes to critical DNSSEC keys without any need for the operator to make changes to configuration files.

E.6.1. Validating Resolver

To configure a validating resolver to use RFC 5011 to maintain a trust anchor, configure the trust anchor using a `managed-keys` statement.

E.6.2. Authoritative Server

To set up an authoritative zone for RFC 5011 trust anchor maintenance, generate two (or more) key signing keys (KSKs) for the zone. Sign the zone with one of them; this is the “active” KSK. All KSK’s which do not sign the zone are “stand-by” keys.

Any validating resolver which is configured to use the active KSK as an RFC 5011-managed trust anchor will take note of the stand-by KSKs in the zone’s DNSKEY RRset, and store them for future reference. The resolver will recheck the zone periodically, and after 30 days, if the new key is still there, then the key will be accepted by the resolver as a valid trust anchor for the zone. Any time after this 30-day acceptance timer has completed, the active KSK can be revoked, and the zone can be "rolled over" to the newly accepted key.

The easiest way to place a stand-by key in a zone is to use the "smart signing" features of **dnssec-keygen** and **dnssec-signzone**. If the key has a publication date in the past, but an activation date which is unset or in the future, "`dnssec-signzone -S`" will include the DNSKEY record in the zone, but will not sign with it:

```
$ dnssec-keygen -K keys -f KSK -P now -A now+2y example.net
$ dnssec-signzone -S -K keys example.net
```

To revoke a key, the new command **dnssec-revoke** has been added. This adds the REVOKED bit to the key flags and re-generates the `K*.key` and `K*.private` files. After revoking the active key, the zone must be signed with both the revoked KSK and the new active KSK. (Smart signing takes care of this automatically.)

Once a key has been revoked and used to sign the DNSKEY RRset in which it appears, that key will never again be accepted as a valid trust anchor by the resolver. However, validation can proceed using the new active key (which had been accepted by the resolver when it was a stand-by key).

See RFC 5011 for more details on key rollover scenarios.

When a key has been revoked, its key ID changes, increasing by 128, and wrapping around at 65535. So, for example, the key "Kexample-net-005-10000" becomes "Kexample-net-005-10128".

If two keys have ID's exactly 128 apart, and one is revoked, then the two key ID's will collide, causing several problems. To prevent this, **dnssec-keygen** will not generate a new key if another key is present which may collide. This checking will only occur if the new keys are written to the same directory that holds all other keys in use for that zone.

Older versions of BIND9 did not have this precaution. Exercise caution if using key revocation on keys that were generated by previous releases, or if using keys stored in multiple directories or on multiple machines.

It is expected that a future release of BIND9 will address this problem in a different way, by storing revoked keys with their original unrevoked key ID's.

Appendix F. Language Support for TN3270 and TN5250

All of the TN3270 and TN5250 clients have been modified to properly handle large key mapping files like MAP3270.DAT and MAP5250.DAT without causing any access violations.

The extended TN3270 client has been modified to allow you to change its notion of the local language and Icelandic has been added to the supported languages.

To use the extended TN3270 client, do the following:

```
$ DEFINE IP_TN3270_EMULATOR DPC_EXTENDED
```

To change the local language,

```
$ DEFINE IP_DPC_TN3270_LANGUAGE language
```

The language parameter can be one of the following:

BRAZILIAN	FRENCH CANADIAN	NEW HEBREW	SPANISH
BUILTIN HEBREW	GERMAN	OLD BELGIAN	SPANISH SPEAKING
DANISH	ICELANDIC	OLD HEBREW	SWISS
FINNISH	ITALIAN	OLD PORTUGUESE	UK ENGLISH
FRENCH	NEW BELGIAN	PORTUGUESE	US ENGLISH

Appendix G. Trademark and Copyright Notifications

This appendix contains a complete listing of trademarks and copyright notification contained in this manual.