TCPware 6.1 Management Guide

September 2023

This manual provides the system manager with the procedures for managing the TCPware family of software products.

Operating System/Version: OpenVMS VAX V5.5-2 or later

OpenVMS Alpha V6.2 or later

OpenVMS Itanium V8.2 or later

Software Version: TCPware 6.1

Process Software
Framingham, Massachusetts
USA

The material in this document is for informational purposes only and is subject to change without notice. It should not be construed as a commitment by Process Software. Process Software assumes no responsibility for any errors that may appear in this document.

Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013.

Third-party software may be included in your distribution of TCPware, and subject to their software license agreements. See www.process.com/products/tcpware/3rdparty.html for complete information.

All other trademarks, service marks, registered trademarks, or registered service marks mentioned in this document are the property of their respective holders.

TCPware is a registered trademark and Process Software and the Process Software logo are trademarks of Process Software.

If the examples of URLs, domain names, internet addresses, and web sites we use in this documentation reflect any that actually exist, it is not intentional and should not to be considered an endorsement, approval, or recommendation of the actual site, or any products or services located at any such site by Process Software. Any resemblance or duplication is strictly coincidental.

Preface

Introducing This Guide

This guide describes how to manage a TCP/IP network and the TCPware components. It is for system managers and administrators.

What You Need to Know Beforehand

Before using TCPware, you should be familiar with:

- The TCPware for OpenVMS products, components, features, and capabilities (see the *User's Guide* for more information)
- Computer networks in general
- The OpenVMS operating system and file system

How This Guide Is Organized

This guide has the following contents:

- Part I, *Managing Hosts* - Includes chapters on the Dynamic Host Configuration Protocol (DHCP) Client, Domain Name Services, and the Dynamic Host Configuration Protocol (DHCP) and BOOTP servers.
- Part II, *Managing Networks* - Includes chapters on the PPP and SLIP serial link interfaces, Cluster Alias Failover, the Simple Network Management Protocol (SNMP), and X.25 networks.
- Part III, *Managing Routing* - Includes a chapter on the routing protocols (primarily GateD).
- Part IV, *Managing Time Services* - Includes chapters on the Network Time Protocol (NTP) and TIMED protocol.
- Part V, *Managing Applications* - Includes chapters on managing the FTP-OpenVMS Client and Server, NFS-OpenVMS Client and Server, Print Services, Berkeley R Command services, the mail services (SMTP-OpenVMS and IMAP), and TELNET-OpenVMS Server.
- Part VI, *Managing Security* - Includes chapters on general TCPware security, Access Restrictions, Packet Filtering, Token Authentication, Kerberos server and applications, the IP Security Option (IPSO), and Secure Shell (SSH).

- Part VII, *Managing Additional Support* - Includes chapters on PATHWORKS support, tunneling DECnet over IP, X Display Manager (XDM), and DECwindows support.
- Part VIII, *Network Testing Tools* - Includes a chapter on the network testing tools, such as DISCARD, FINGER, NETCU DEBUG, NSLOOKUP, PING, QUOTED, TCPDUMP, and TRACEROUTE.
- Appendixes, including NFS-to-OpenVMS filename mapping rules, Data Network Identification Codes for X.25 networks, and TCPware logicals.
- Index to this guide.

Online Help

You can use help at the DCL prompt to find the following:

- Topical help - Access TCPware help topics as follows:

  ```
  $ HELP TCPWARE [topic]
  ```

  The topic entry is optional. You can also enter topics and subtopics at the following prompt and its subprompts:

  ```
  TCPWARE Subtopic?
  ```

  Online help is also available from within certain TCPware components: FTP client and server, Network Control Utility (NETCU), TELNET client, NSLOOKUP, and TRACEROUTE. Use the `HELP` command from within each component:

  ```
  NETCU>HELP [topic]
  ```

- Error messages help - Access help for TCPware error messages as follows:

  ```
  $ HELP TCPWARE MESSAGES
  ```

  If the error message is included in the `MESSAGES` help, it identifies the TCPware component and provides a meaning and user action. See the `Instructions` under `MESSAGES`.

Obtaining Customer Support

You can use the following customer support services for information and help about TCPware and other Process Software products if you subscribe to our Product Support Services. (If you bought TCPware products through an authorized TCPware reseller, contact your reseller for technical support.) Contact Technical Support directly using the following methods:

Electronic Mail

E-mail relays your question to us quickly and allows us to respond as soon as we have information for you. Send e-mail to `support@process.com`. Be sure to include your:

- Name
- Telephone number
- Company name
- Process Software product name and version number
- Operating system name and version number
- Process Software support contract number

Describe the problem in as much detail as possible. You should receive an immediate automated response telling you that your call was logged.

Telephone

If calling within the continental United States or Canada, call Process Software Technical Support toll-free at (800) 394-8700. If calling from outside the continental United States or Canada, dial +1 (508) 628-5074. Please be ready to provide your name, company name, Process Software support contract number, and telephone number.

World Wide Web

There is a variety of useful technical information available on our World Wide Web home page, http://www.process.com/

License Information

TCPware for OpenVMS includes a software license that entitles you to install and use it on one machine. Please read and understand the *Software License Agreement* before installing the product. If you want to use TCPware on more than one machine, you need to purchase additional licenses. Contact Process Software or your distributor for details.

Maintenance Services

Process Software offers a variety of software maintenance and support services. Contact us or your distributor for details about these services.

Documentation Set

The documentation set for TCPware consists of the following:

- ***Installation & Configuration Guide*** - For system managers and those installing the software. The guide provides installation and configuration instructions for the TCPware products.
- ***Management Guide*** - For system managers. This guide contains information on functions not normally available to the general network end user. It also includes implementation notes and troubleshooting information.
- ***Network Control Utility (NETCU) Command Reference*** - For users and system managers. This reference covers all the commands available with the Network Control Utility (NETCU) and contains troubleshooting information.
- ***Programmer's Guide*** - For network application programmers. This guide gives application programmers information on the callable interfaces between TCPware and application programs.
- ***Release Notes*** for the current version of TCPware - For all users, system managers, and application programmers. The *Release Notes* are available online on your TCPware media and are accessible before or after software installation.
- ***User's Guide*** - For all users. This guide includes an introduction to TCPware products as well as a reference for the user functions arranged alphabetically by product, utility, or service.

Conventions Used

Convention	Meaning
host	Any computer system on the network. The local host is your computer. A remote host is any other computer.
`monospaced type`	System output or user input. User input is in **`reversed bold`** `type`. Example: `Is this configuration correct?` **`YES`** Monospaced type also indicates user input where the case of the entry should be preserved.

italic type	Variable value in commands and examples. For example, *username* indicates that you must substitute your actual username. Italic text also identifies documentation references.
[*directory*]	Directory name in an OpenVMS file specification. Include the brackets in the specification.
[*optional-text*]	(Italicized text and square brackets) Enclosed information is optional. Do not include the brackets when entering the information. Example: `START/IP line address [info]` This command indicates that the `info` parameter is optional.
{value | value}	Denotes that you should use only one of the given values. Do not include the braces or vertical bars when entering the value.
Note	Information that follows is particularly noteworthy.
Caution	Information that follows is critical in preventing a system interruption or security breach.
key	Press the specified key on your keyboard.
Ctrl+key	Press the control key and the other specified key simultaneously.
Return	Press the Return or Enter key on your keyboard.

1. Common Interfaces

Introduction

This chapter describes the following network interfaces:

- Ethernet
- Fiber Distributed Data Interface (FDDI)
- Token Ring
- LAN Emulation over ATM
- Classical IP over ATM
- HYPERchannel
- IP-over-DECnet
- proNET
- HP Wide Area Network (WAN) device drivers
- Pseudo devices

Ethernet, FDDI, Token Ring, and ATM

TCPware for OpenVMS supports all HP Ethernet, FDDI, Token Ring, LAN Emulation over Asynchronous Transfer Mode (ATM), and Classical IP over ATM (CLIP) interfaces, so that you can send IP datagrams over these types of LANs.

These interfaces generally include the use of the Address Resolution Protocol (ARP) and the Reverse Address Resolution Protocol (RARP), except CLIP, which does not support RARP.

For details on configuring LAN network devices and their addresses, see the *Installation & Configuration Guide*, Chapter 3, *Configuring the TCP/IP Core Environment*.

Address Resolution Protocol

ARP dynamically maps between internet and physical addresses. TCPware provides an ARP table of mappings that it keeps in cache. If a mapping is not in this cache when a datagram is transmitted, TCPware queues the datagram and broadcasts an ARP request over the network. When a host responds with an internet-to-physical address mapping, TCPware adds it to its cache and transmits the queued datagrams.

TCPware's implementation of ARP does not probe for imposters. However, it may generate the following OPCOM message, which includes the physical address of the system with a duplicate internet address:

```
Duplicate IP address: Sent from physical address...
```

In the rare case where the remote system does not support ARP, you can use the `ADD ARP` and `REMOVE ARP` commands in TCPware's Network Control Utility (NETCU) to add or remove entries for the system. You can also find specific ARP entries using `FIND ARP`, and show the entire ARP table using the `SHOW ARP` command.

You can also set various ARP parameters using `/ARP_`*x* qualifiers with the `SET INTERFACE` command. The ARP qualifiers control when to check the age of an ARP entry, how long to keep it in cache, how long to wait for an unresolved entry to be removed from cache, and the maximum size of the cache. TCPware provides defaults for these parameters, so you do not normally need to use these commands. For example, it normally removes entries from its ARP table if it does not receive a packet for an entry within 10 minutes, or if the table is more than 512 entries long.

Reverse Address Resolution Protocol

The Reverse Address Resolution Protocol (RARP) enables a diskless client to find its IP address at startup from a RARP server. The diskless client broadcasts a request that contains its physical hardware address. The server maps the hardware address to the IP address corresponding to the physical address of the client. The TCPware system only responds to RARP requests for permanent address entries in its ARP cache. (Note that BOOTP provides the same type of services.)

RARP support is enabled by default for all Ethernet, FDDI, and Token Ring interfaces. RARP support is disabled for LAN Emulation over Asynchronous Transfer Mode (ATM) and Classical IP over ATM (`CLIP-`*n*) lines. You can explicitly disable RARP support using the `NETCU STARTUP` command with the `/FLAGS=NORARP` qualifier.

Ethernet Trailer Packets

TCPware can operate with trailer packets enabled or disabled. Trailer packets have some of the packet headers at the end of the packet rather than at the beginning. TCPware disables trailer packet support by default on Ethernet lines. However, some implementations use trailer packets, such as those running under UNIX. TCPware can receive and process trailer packets, but it will never transmit them.

To disable trailer packets on UNIX systems, use the `ifconfig` command with the `-trailers` option. To disable trailer packets on OpenVMS systems, use the `NETCU START/IP /FLAGS=NOTRAILERS` qualifier.

Qualifiers with LAN Device Lines

The `START/IP` command supports several qualifiers that you can use with Ethernet, FDDI, Token Ring, LAN Emulation over Asynchronous Transfer Mode (ATM), and Classical IP over ATM lines. See Table 2-11 and Table 2-13 in the *NETCU Command Reference*.

VMS Communications Interfaces Support

TCPware supports VMS Communications Interfaces (VCIs). VCI is a high speed interface to the LAN drivers. If you want to disable VCI support for some reason, use the `/FLAGS=NOVCI` qualifier to the `NETCU START/IP` command.

Limiting Receive Packet Rate

TCPware supports placing a limit on the number of receive packets it processes per second. If a limit is set on an interface and that limit is exceeded, TCPware may issue the following OPCOM message:

```
Warning - maximum receive packet rate exceeded on line line-id (rate
packets/second).
```

This indicates that the interface specified by *`line-id`* received more packets than were allowed. This may indicate that either the receive packet rate limit is too low or that a flood of packets arrived at the system and a network problem exists that should be corrected. If the limit is too low, raise it using the `NETCU SET INTERFACE /RECEIVE_LIMIT` command. If a network problem exists, investigate it and correct it.

HYPERchannel

TCPware supports any HYPERchannel interface supported by Network System Corporation's H269 device driver, including the UNIBUS, QBUS, MASSBUS, and BIBUS interfaces.

The HYPERchannel interface support includes the use of the Address Resolution Protocol (ARP). Use ARP to automatically map an internet address to a physical address.

Address Format

When starting a HYPERchannel line, you must specify line-specific-information. This is the local HYPERchannel address in 32-bit address. The format is *`aa-bb-cc-dd`*, where *`aa`*, *`bb`*, *`cc`*, and *`dd`* are hexadecimal values representing each byte of the address:

- *`aa`* is the global network address domain (if none, specify `00`)
- *`bb`* is the global network address network (if none, specify `00`)

- *cc* is the physical unit
- *dd* is the logical unit

If you are using 16-bit addresses, specify the address as `00-00-cc-dd`.

The H269 driver's `IO$_ATTACH` function uses the *cc*-*dd* portion of the local HYPERchannel address as the path address. You should always specify the local HYPERchannel address parameter as the 32-bit HYPERchannel address.

Address Mapping

TCPware needs to map the 32-bit internet addresses into 32-bit HYPERchannel addresses. These two address families are not related. That is, there is no mathematical formula that you can use to convert from one address family to the other. Instead, you must either configure an ARP server or pre-load the address resolution table with the mappings.

> **Note:** You must properly configure the server with address mappings for all systems on the HYPERchannel network.

To pre-load the address resolution table, use `NETCU ADD ARP` commands (see the *NETCU Command Reference*). The `ADD ARP` command requires that you specify a 48-bit value (in hexadecimal format). The syntax is:

```
aa-bb-cc-dd-ee-ff
```

where the additional *ee* is the HYPERchannel trunks-to-try mask (typically `FF`) and *ff* is the HYPERchannel flags mask (typically `00`). For ease of use, the `ADD ARP` command allows you to specify most addresses using *aa-bb-cc-dd*`-00-00`. It supplies the proper trunks-to-try mask of `FF`. See the HYPERchannel documentation for information on other values for the trunks-to-try and flags fields.

Note that if you load the address resolution table through the `ADD ARP` command, specify `/PERMANENT`. Otherwise, TCPware removes the added entries after a short time.

Qualifiers with HYPERchannel Lines

The `START/IP` command supports several qualifiers that you can use with HYPERchannel lines. See Table 2-11 in the *NETCU Command Reference*.

IP-over-DECnet

TCPware provides support for DECnet interface implementations so that you can send IP datagrams over DECnet links. This lets you connect separate TCP/IP LANs over DECnet WAN links.

Configuring DECnet Lines

To configure a DECnet line:

1. Enter the DECnet line identification, internet address, and host name for the local internet address in the response to the applicable prompts in the `CNFNET` network configuration utility. Note that you need a different IP address for the DECnet line.
2. Enter the appropriate line-specific information.

Line-Specific Information

The `START/IP` command *`line-specific-information`* parameter provides the required DECnet link information. Enter the *`line-specific-information`* in the following format:

```
node-name::"TASK=object-name"
```

Parameter	Identifies the...
`node-name`	listener node when the master mode issues it. It identifies the master node when the listener mode issues it.
`object-name`	object used on the listener node. Both the master and listener nodes must specify the same *`object-name`*.

An IP-over-DECnet line has a master node at one end and a listener node at the other end.

Sample Configuration

The below example shows selections from a sample IP-over-DECnet configuration in TCPware's `CNFNET`.

```
Line Id       Network Device

QNA-n          for HP's DELQA, DESQA, or DEQNA (UQDRIVER)
UNA-n          for HP's DELUA or DEUNA (XEDRIVER)

Enter the line identifications [LPB-0, ISA-0]: LPB-0, ISA-0, DECNET-0
```

```
What is the local host's INTERNET ADDRESS for line ISA-0 [10.16.1.1]:
RETURN

What is the NAME for line ISA-0 [BVA2]: BVA2

What is the SUBNET MASK for the line SVA-0 [255.255.0.0]: RETURN

Do you want to enable TRAILER packet support for line ISA-0 [NO]: RETURN

Do you want to enable RARP (Reverse ARP) support for line ISA-0 [YES]:
RETURN

What is the local host's INTERNET ADDRESS for line ISA-0 [10.16.1.2]:
RETURN

What is the NAME for line DECNET-0 [BVA2]: BVA2

What is the SUBNET MASK for the line DECNET-0 [255.255.0.0]: RETURN

What is the DECnet link information for line DECNET-0: ONA1::"TASK=OZONE"

Is this the LISTENER end of the DECnet link for line DECNET-0 [NO]: RETURN

The network devices are configured as follows:

Line        Address     Name       Options

LPB-0       127.0.0.1   LOOPBACK
SVA-0       10.16.1.1   BVA2        /MASK=255.255.0.0
                                    /FLAGS=(NOTRAILERS)
DECNET-0    10.16.1.2   ONA1        /MASK=255.255.0.0
ONA1::"task=OZONE" /FLAGS=LISTENER
```

Qualifiers with DECnet Lines

The `START/IP` command supports several qualifiers that you can use with DECnet lines. See Table 2-11 in the *NETCU Command Reference*.

proNET-10/80

The proNET-10 and proNET-80 token ring controllers produced by Proteon, Inc. form a link between the hardware devices, the token ring, and TCPware.

Configuring proNET Lines

When you configure proNET lines, the D component of the internet address (using the standard A.B.C.D convention) must match the node address of the proNET controller.

Qualifiers with proNET Lines

The `START/IP` command supports a qualifier that you can use with proNET lines. See Table 2-11 in the *NETCU Command Reference.*

HP Wide Area Network (WAN) Device Drivers

The HP WAN Device Drivers are synchronous interfaces that form a link between the hardware devices and TCPware. TCPware for OpenVMS supports the DSV11, DSB32, and DST32 HP WAN interfaces.

Line-Specific Information

The `NETCU START/IP` command's *`line-specific-information`* parameter provides the required DECnet link information. The *`line-specific-information`* is a quoted string of the line configuration options shown in *Chapter 4*.

For details on these parameters, such as the possible values for the line speed, CRC, and so on, see the *VAX Wide Area Network Device Driver's Programmer's Guide.*

An example of *`line-specific-information`* is

```
"PROTOCOL DDCMP POINT CLOCK INTERNAL LINE SPEED 64000"
```

When specifying *`line-specific-information`* for HP WAN device drivers lines, be aware of the following:

- You must enclose *`line-specific-information`* in quotes for these lines.
- You can use keyword abbreviations.

Qualifiers with VAX WAN Device Driver Lines

The `START/IP` command supports several qualifiers you can use with HP WAN device driver lines. See Table 2-11 in the *NETCU Command Reference*.

Parameter	**Takes...**	**Description**
`PROTOCOL`	`DDCMP POINT LAPBE`	Line protocol used

	LAPB SDLC	
DUPLEX	HALF FULL	Line operation used
CLOCK	INTERNAL EXTERNAL	Line clocking used
CRC	type	Type of CRC used (*not recommended*)
LINE SPEED	baud	Line speed (only useful with CLOCK INTERNAL)
RECEIVE BUFFERS	number	Number of receive buffers
RETRANSMIT TIMER	time	Retransmission tine (for PROTOCOL, DDCMP POINT only)

Pseudo Devices

Pseudo devices are a way to configure a physical device to have multiple Internet addresses. Pseudo devices are typically used when a system is connected to a network that needs extra network numbers assigned to it. Pseudo devices can also be used in place of secondary addresses (for example, when a system has multiple addresses on the same network).

When starting a pseudo device, you specify the local Internet address, network mask, and the physical device to which the pseudo device is connected.

Adding a Pseudo Device

CNFNET allows the configuration of pseudo devices.

The TCPware line-id for a pseudo device is PSD-*n*, and *n* is 0 to 255.

To configure one or more pseudo devices via CNFNET, include the line-id or line-ids for the pseudo devices when prompted to enter the line identifications for all the network devices. Be sure to enter the pseudo device line-id *after* the physical device line-id. CNFNET prompts you for the standard information (Internet address, host name, and subnet mask) and for the physical device line-id for the pseudo device.

You can also start pseudo devices by using the NETCU START/IP command. See START/IP in Chapter 2 of the *NETCU Command Reference*.

Characteristics of Pseudo Devices

Pseudo devices are interchangeable and usable just like physical devices.

However, there are a few special characteristics that are important to point out:

- Multicast joins/leaves are redirected to the physical device.
- NETCU SHOW NETWORK shows no transmit/receive counts for pseudo devices. The physical device reflects the transmit/receive activity.
- Pseudo devices are removed automatically whenever the physical device is removed (such as by a NETCU STOP/IP line-id command.
- Starting a pseudo device on a pseudo device (by specifying a pseudo device line-id as the *Real-Line-ID* for a NETCU START/IP command) is allowed; however, the underlying physical device is used.
- Packet filtering is not available for pseudo devices as these devices never receive any packets (the physical device does). Therefore, you must do all packet filtering on the physical device and must take this into consideration when creating the packet filter list. Attempting to issue a NETCU SET FILTER or NETCU SHOW FILTER command on a pseudo device returns an error message.
- Once a pseudo device is started, the command NETCU SHOW INTERFACE *line-id* can be utilized to display the physical device information. For example:

```
$ NETCU SHOW INTERFACE PSD-1
For Network Line PSD-1:
On Physical Line EWA-0:

No receive packet rate limit has been set.
The maximum receive packet rate was 0 packets/second.

The ARP entry limit is 512 entries.
The ARP age check interval is 30 seconds.
The ARP entry age limit is 600 seconds.
The ARP entry wait limit is 20 seconds.
```

- The line-id value for pseudo devices is 00*nn*0042 (hex), where *n* is the unit number (PSD-*n*).
- For proper operation of pseudo devices (and TCPware is general), the LPB-0 (loopback) device must exist. The LPB-0 device is technically not optional.
- Pseudo devices cannot be started on unnumbered interfaces.

When to Use Pseudo Devices, Secondary Addresses, and Interface Routes

TCPware continues to support secondary addresses (`NETCU ADD SECONDARY`) and interface routes (`NETCU ADD ROUTE`) in addition to the new pseudo devices. Some recommendations as to which method to use and the conditions under which to use them are described next.

- If a TCPware system is connected to a network via a single interface that has multiple networks numbers assigned to it:
    - Use a pseudo device for each network number on which the TCPware system has an Internet address (other than the one that is used to start the physical device).
    - Use an interface route for each network number on which the TCPware system does not have an Internet address. For an interface route, specify the line-id of the physical device in place of the gateway address parameter.
- If a TCPware system has multiple addresses on a single network number:
    - Use either pseudo devices or secondary addresses for the additional addresses. Using a pseudo device has some advantages and is recommended (especially if a DNS server is running on the system).
- To use the cluster alias failover support:
    - The secondary address feature must be used.

> **Note:** If your site is using secondary addresses you might want to consider whether switching to pseudo devices makes sense for these addresses.

2. DHCP Client

Introduction

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

General Description

The DHCP client resides on the client host and dynamically sets the network configuration. The TCPware DHCP client communicates with a DHCP server to get an IP 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. Also, see Chapter 4 of this guide for general DHCP process and TCPware DHCP server information.

Beginning with TCPware 6.0, TCPware supplies two clients: v3 and v4. Only one can be run at any given time.

Because the DHCP client supports a single network interface on the host, you can only use it to configure a single network line in TCPware.

Process Software recommends that you do not use the DHCP client with other TCPware components that usually need a static IP address on the same host, such as DHCP server, authoritative DNS server, and GateD.

Setting Up the DHCP Client

If this is your first time using a DHCP client on the host, you need to create a configuration file for it (`TCPWARE:DHCLIENT.CONF`). If you have been running the DHCP V3 client and want to change to running the DHCP V4 client, you can use the same configuration file.

If you need to create a configuration file, there are template configuration files available for both the DHCP V4 client (DHCLIENT) and the DHCP V4 client (DHCLIENT4) in the TCPWARE common directory.

To create a configuration file from one of the templates, do the following. For DHCLIENT:

```
$ COPY TCPWARE:DHCLIENT_CONF.TEMPLATE TCPWARE:DHCLIENT.CONF
```

Or for DHCLIENT4:

```
$ COPY TCPWARE:DHCLIENT4_CONF.TEMPLATE TCPWARE:DHCLIENT.CONF
```

The DHCP client configuration file should now be edited to specify the name of your host. Talk to your network administrator: the administrator may want to assign you a host name.

To specify a host name, edit the configuration file to replace this line:

```
#send host-name "testing";
```

with this line:

```
send host-name "any hostname you want";
```

To configure your local host to use the DHCP client, run the TCPware configuration utility CNFNET. To use the V3 DHCP client, you can run CNFNET in one of two ways:

```
$ @TCPWARE:CNFNET
```

```
$ @TCPWARE:CNFNET DHCLIENT
```

To use the V4 DHCP client, you must configure it individually:

```
$ @TCPWARE:CNFNET DHCLIENT4
```

CNFNET can also be used to disable the DHCP client on the host. If you have been running the V3 DHCP client and want to change to the V4 DHCP client, run CNFNET to disable DHCLIENT then run CNFNET again to enable DHCLIENT4.

After you configure the local host to use a DHCP client, you can run STARTNET to start TCPware:

```
$ @TCPWARE:STARTNET
```

Here are two examples:

Using CNFNET

```
$ @TCPWARE:CNFNET
TCPware (R) for OpenVMS Version 6.1-0 Network Configuration procedure for:
TCP/IP Services:
              FTP-OpenVMS
              NFS-OpenVMS Client
              NFS-OpenVMS Server
              SMTP-OpenVMS
              TELNET-OpenVMS
              Kerberos Services
              SSH-OpenVMS Server

This procedure helps you define the parameters needed to get
TCPware (R) for OpenVMS running on this system.
This procedure creates the configuration data file,
TCPWARE_SPECIFIC:[TCPWARE]TCPWARE_CONFIGURE.COM, to reflect your
system's configuration.

Type Return to continue... Return

... ...
... ...

You need to supply the following information for each line:

        - The internet address for the line
        - The name for the line (same as the host name if single
          line host, fully qualified domain name if using DNS)
        - The subnet mask for the line
        - The line specific information (depends on the line)

If there is a DHCP server running on the network and this is a single line
host, you may get the information from DHCP server automatically. To do so,
please select 2.

        1. Configure Internet address and related items manually.

        2. Configure Internet address and related items automatically

        Continue with selection [1]: 2

Configure line SVA-0:

Set DHCP client Host Name

You can press Enter to let the system choose a host name. Or you can specify
a name you would like to use for the host. However, the final name for the
host will be up to the DHCP server to decide, it may not be the name you
specify.

Host Name (Return to end) []: Return
```

```
You need to specify local time zone information.  Time zone maybe specified
as fixed value which must be manually set for the daylight savings time
change, or you can use NTP (Network Time Protocol) Daemon to change the
system clock and time offset automatically.

Do you want to have NTP set the time and time offset automatically [NO]?

... ...
... ...
```

Using CNFNET DHCLIENT

```
$ @TCPWARE:CNFNET DHCLIENT

TCPware(R) for OpenVMS Version 6.0-0 Network Configuration procedure for:

TCP/IP Services:

                FTP-OpenVMS
                NFS-OpenVMS Client
                NFS-OpenVMS Server
                SMTP-OpenVMS
                TELNET-OpenVMS
                Kerberos Services
                SSH-OpenVMS Server

This procedure helps you define the parameters needed to get
TCPware(R) for OpenVMS running on this system.

This procedure creates the configuration data file,
TCPWARE_SPECIFIC:[TCPWARE]TCPWARE_CONFIGURE.COM, to reflect your system's
configuration.

Type Return to continue... Return

Configuring the Dynamic Host Configuration Protocol (DHCP) Client:

Do you want to use the DHCP Client [YES]: Return

Set the DHCP client host name.

You can press Enter to let the system choose a host name. Or you can specify
a name you would like to use for the host. However, the final name for the
host will be up to the DHCP server to decide. It may not be the name you
specify.

Host Name (press Return to end [ ]: RETURN

The DHCP Client can perform error and debug message logging to OPCOM and a
```

```
log file.

Do you want to enable logging [NO]: Return

Do you want to restart DHCLIENT [NO]: Return
$
```

DHCP Client Functions and Logicals

The DHCP client is started as a VMS detached process when TCPware is started up.

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 TCPware 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.

When the DHCP client is enabled, the logical `TCPWARE_DHCP_CLIENT` is equal to `1`.

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 the following TCPware logicals:

- `TCPWARE_DOMAINNAME`
- `TCPWARE_NAMESERVERS`

It may change the following related OpenVMS logicals:

- `UCX$BIND_DOMAIN`
- `UCX$BIND_SERVER00x`
- `UCX$INET_HOST`
- `UCX$BIND_SERVER000`
- `UCX$INET_DOMAIN`

- UCX$INET_HOSTADDR

DHCP Client Configuration

The TCPware DHCP client uses the configuration file TCPWARE:DHCLIENT.CONF to control the behavior of the client. You can use one of the supplied template files to start with, as described above.

To explore more configuration possibilities, read the following dhclient.conf descriptions. The descriptions were edited based on the ISC's descriptions for the Unix version of the DHCP client configuration file. The original documents can be found in the ISC's website at http://www.isc.org.

The dhclient.conf 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.

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, however.

Statement	Description
backoff-cutoff *time;*	The client uses an exponential back off 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 that the client is allowed to back off. The actual value is set randomly between one-half to one and a half times the time specified. The default is two minutes (for V3) or fifteen seconds (for V4).
`initial-delay` *`time;`* (V4 only)	The `initial-delay` statement sets the maximum time the client can wait after starting before commencing its first transmission. Previous versions of the ISC DHCP client waited up to 5 seconds. Version 4 has no initial delay by default, to avoid an adverse impact on system startup time. To restore the old behavior, set `initial-delay` to 5.
`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. Each time a message is sent, the interval between messages 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 60 seconds (for V3) or 5 minutes (for V4).
`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, if it

	has received at least one such offer. If no offers have been received by the time the `select-timeout` 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 the time that 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 sixty (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.

Lease Requirements and Requests

The DHCP protocol allows the client to request the server to send it specific information, and not send it other information that it is not prepared to accept. 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 the DHCP Server chapter in this guide for a list of DHCP options.

Statement	**Description**

`request [`*`option`*`] [, ... `*`option`*`];`
 	
(V4 only)
`[also] request`
`[ [`*`option-space`*` .] `*`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;`

	(V4 only) By default the V4 client requests the `subnet-mask`, `broadcast-address`, `time-offset`, `routers`, `domain-name`, `domain-name-servers`, and `host-name` options. Note that if you specify a `request` statement, you override these defaults and these options will not be requested.
	
	In some cases, it may be desirable to send no parameter request list at all. To do this, simply write the request statement but specify no parameters:
	
	`request;`
	
	In most cases, it is desirable to simply add one option to the request list which is of interest to the client in question. In this case, it is best to 'also request' the additional options:
	
	`also request static-routes;`

`require [option] [, ... option];`
 	
(V4 only)
`[also] require`
`[ [option-space .] option]`
`[, … option];`
	The `require` statement lists options that must be sent for an offer to be accepted. Offers that do not contain all the listed options are ignored. There is no default require list.

`send { [option declaration]` `[, ... option declaration]}`	The `send` statement causes the client to send the specified options to the server with the specified values. These are full option declarations. 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";`

Dynamic DNS

The V4 DHCP client contains a prototype implementation with very limited support for doing DNS updates when a lease is acquired. Its use is not recommended. No further information is provided here. By default, the DHCP client will not do DNS updates.

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.

Statement	**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.

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 decide after some period of time (see *Protocol Timing*) 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 automatically its address. This is done with the `lease` statement.

> **Note:** The lease statement is also used in the `dhclient.db` (v3) or `dhclient.leases` (v4) 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`/`dhclient.leases` file. Such syntax is documented here for completeness.

The following lease declarations are possible:

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.
`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 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 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). For predefined leases, this can always be set to 0. *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. (V4 only) The V4 DHCP client can specify dates in two formats. The software will output times in one of these two formats depending on the setting of the `db-time-format` configuration parameter: `default` or `local`. The `default`

	format is the one described above. The `local` format is as follows: `epoch` *`seconds-since-epoch`*`;` The `seconds-since-epoch` is set according to the system's local clock (often referred to as "UNIX time"). Note that when defining a static lease, you may use either time format, and need not include the comment or values after it. If the time is infinite in duration, then the date is `never` instead of an actual date.
`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 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 `TCPWARE:DHCLIENT-SCRIPT.COM`.
`vendor option space "`*`name`*`";`	(V4 only) The `vendor option space` statement is used to specify which option space should be used for decoding the `vendor-encapsulate-options` option, if one is

	received. The `dhcp-vendor-identifier` can be used to request a specific class of vendor options from the server.

Alias Declarations

(V4 only) The `alias` declaration resembles a lease declaration, except that options other than the subnet-mask option are ignored by the client configuration script, and expiry times are ignored. A typical alias declaration includes an interface declaration, a fixed-address declaration for the IP alias address, and a subnet-mask option declaration.

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

Some DHCP clients may require that in addition to the lease they may acquire via DHCP, their interface also be configured with a predefined IP alias so that they can have a permanent IP address even while roaming. The DHCP V4 client doesn't support roaming with fixed addresses directly, but in order to facilitate such experimentation, the DHCP client can be set up to configure an IP alias using the alias declaration.

Other Declarations

The below table lists all of the other DHCP client declarations that are supported by TCPware. There are other declarations in the ISC DHCP V4 client which are not supported by TCPware and are not mentioned here.

Declaration	**Description**
`db-time-format [default | local];`	(V4 only) The `db-time-format` option determines which of two output methods are used for printing times in leases files. The `default` format provides day-and-time in UTC, whereas local uses a `seconds-since-epoch` to store the time value, and helpfully places a local time zone time in a comment on the same line. The formats are described in detail above.

`reject ip-address;`
 	
`reject cidr-ip-address [, … cidr-ip-address];`
	The `reject` statement causes the DHCP client to reject offers from servers whose server identifier matches the specified hosts or subnets. This can be used to avoid being configured by rogue or mis-

	configured DHCP servers, although it should be a last resort; better to track down the bad DHCP server and fix it. (V4 only) The `cidr-ip-address` configuration type is of the form *`ip-address`*`[/`*`prefixlen`*`]`, where *`ip-address`* is a dotted quad IP address, and *`prefixlen`* is the CIDR prefix length of the subnet, counting the number of significant bits in the netmask starting from the leftmost end. Example configuration syntax: `reject 192.168.0.0/16, 10.0.0.5;` The above example would cause offers from any server identifier in the entire RFC 1918 "Class C" network 192.168.0.0/16, or the specific single address 10.0.0.5, to be rejected.

Example

This is the template configuration for the V4 DHCP client:

```
# you can specify a host name here
#send host-name "testing";
#
# you can specify the length of the lease for the client
#send dhcp-lease-time 7200;
#
# you can request certain options from the server
# note: these are the options that are looked at by the tcpware client
request subnet-mask, broadcast-address, routers, static-routes,
      domain-name, domain-name-servers, host-name;
#
# you can require certain options
require subnet-mask;
#
# you can modify options received from the server using
```

```
prepend/append/supersede
#prepend domain-name-servers 127.0.0.1;
#
# you can supply defaults for options not sent by the server
#default domain-name-servers 127.0.0.1;
#
# you can reject offers from certain servers
#reject 10.10.10.10;
#
# client configuration script
#script "tcpware:dhclient-script.com";
```

Troubleshooting the DHCP Client

How do I know the DHCP client has configured my network successfully?

Check if the `TCPWARE_DHCP_CLIENT` logical is equal to `1`, you can do:

```
$ SHOW LOGICAL TCPWARE_DHCP_CLIENT
"TCPWARE_DHCP_CLIENT" = "1" (LNM$SYSTEM_TABLE)
$
```

Then run the `NETCU SHOW NET` command to check if the line is assigned with an IP address.

```
$ NETCU SHOW NET
Line   Local Address   Subnet Mask    MTU   Xmits  Errs  Recvs  Errs  RBU
----   -------------   -----------    ---   -----  ----  -----  ----  ---
SVA-0  10.10.10.10     255.255.255.0  1500   49     0    3999     0    0
LPB-0  127.0.0.1       255.0.0.0      64512  64     0    64       0    0

        4191 IP datagrams were transmitted, of which
             0 were fragmented
             0 were forwards
             0 were IGMP reports

      115244 IP datagrams/fragments were received, of which
             0 were fragments
             0 were forwarded
          1397 were IGMP queries/reports
       61785 IP datagrams were delivered to receivers.
$
```

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.

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 TCPWARE_NAMESERVERS
  ```

  and

  ```
  $ show logical TCPWARE_DOMAINNAME
  ```

  to make sure the DNS client information is correct.

- The DNS server may be down.
- The DNS client may be down. Check if the `TCPware_DNS` process exists.

Why is the local address 0.0.0.0 when I use `netcu show net`?

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.
There are no IP addresses available in the DHCP server.	Increase the IP address on the DHCP pool server.

Where can I find the status information of the DHCP client?

• The file TCPWARE:DHCLIENT-SCRIPT-ENV.TMP contains the most recent environment variables used by the DHCP client script file to configure the network.
• The file TCPWARE:DHCLIENT.DB (V3) or TCPWARE:DHCLIENT.LEASES (V4) contains the DHCP client lease history.
• The file TCPWARE:DHCLIENT.LOG contains information about the DHCP client process.

Note: The TCPWARE: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 using the command:

```
$ @tcpware:cnfnet dhclient
```

or

```
$ @tcpware:cnfnet dhclient4
```

3. Domain Name Services

Introduction

This chapter describes TCPware's Domain Name Services and how it relates to the Domain Name System (DNS). Specific sections describe:

- The different types of DNS servers, the DNS client, and what they do.
- The syntax, special commands, and fields used in each type of database record.
- TCP/IP cluster load balancing.

Domain Name Services allows local hosts to obtain information about other hosts by accessing a distributed DNS database. This database supplies Internet addresses and hostnames throughout the network. Any upper-layer protocol, such as FTP or SMTP, can use DNS when it needs host information.

Domain Name Services also provides the services for cluster load balancing.

Domain Name System (DNS) Concepts

For a full description of the concepts behind the Domain Name System (DNS) that forms the basis of the Domain Name Services, see *DNS Defined: A Practical Guide to TCP/IP Domain Name System and Services. DNS Defined* is one of the volumes in Process Software's useful *TCP/IP Reference Library*.

DNS Client

A DNS client (also called a resolver) communicates with a DNS server to resolve a host name and internet address. The client does not maintain a database. The client only sends queries and provides the received answers to applications.

You can configure your local host to support a DNS client when you run the TCPware configuration. The host can support a DNS client only, or both a client and name server.

The configuration procedure (CNFNET) prompts you to specify the internet addresses of up to three name servers the client can query. The client reads this information from two logicals you set through CNFNET:

- TCPWARE_DOMAINNAME
- TCPWARE_NAMESERVERS

The client also allows you to set up to six domains in a search list, as well as the minimum number of dots to recognize in a host name to make it fully qualified. The client reads this information from two logicals you set through CNFNET:

- TCPWARE_DOMAINLIST
- TCPWARE_RES_OPTIONS "ndots: *ndots*"

The configuration procedure (CNFNET) allows you to specify a resolver timeout setting that the client reads from two logicals you set through CNFNET:

- TCPWARE_RES_RETRANS_MIN (specifies minimum retransmit time value in seconds)
- TCPWARE_RES_RETRIES (specifies retry count)

See the *Installation & Configuration Guide*, Chapter 4, *Configuring the TCP/IP Services*, the *Configure the DNS Resolver* section.

When an application needs to resolve a host name or internet address, the client queries the first name server the TCPWARE_NAMESERVERS logical defines. The client continues to query the other name servers on its list until it receives an answer, or the list is exhausted. The amount of time varies for what is needed to send a query and receive a response.

The TCPware resolver queries domain names in the following sequence:

1. The resolver queries the fully qualified domain name if the domain name contains *ndots* number of dots (see the TCPWARE_RES_OPTIONS logical syntax); the default is 1.
2. If the first step fails or the domain name contains less than *ndots*, the resolver queries *name.default-domain* or the list established by the TCPWARE_DOMAINLIST logical.
3. If the second step fails, the resolver queries for *name* only.

You can restart the resolver process (TCPware_DNS) if it goes down (or the NETCU STOP/DNS command was used) by specifying @TCPWARE:STARTUP_RESOLVER DETACH on the command line.

Domain Name Server

To set up a domain name server on your local host, perform these steps:

1. Determine whether the names on the local host will be authoritative for a zone or will be just a caching server.
2. Run the CNFNET configuration procedure described in the *Installation & Configuration Guide*, Chapter 4. When you set up a server and (there is no existing NAMED.CONF file), CNFNET converts the existing TCPWARE_NAMED_ROOT:NAMED.BOOT file to a TCPWARE_NAMED_ROOT:NAMED.CONF file. If there is no NAMED.BOOT file, CNFNET creates a default NAMED.CONF file set up as a caching server.
3. If the host is to be authoritative, add zone statements in the NAMED.CONF file (see *Zone*) for each zone.
4. Gather the information you need for the database files so that the server can resolve queries. The information is in the *Resource Records* section of this chapter.
5. Enter the information in the database files using a text editor. (See *Editing Database Files*.)

Starting and Stopping

The name server starts up when you execute the STARTNET.COM procedure and shuts down with the SHUTNET.COM procedure.

1. Log in as the system manager.
2. Stop the name server process:

```
$ @TCPWARE:SHUTNET DNS
```

3. Start the name server process:

```
$ @TCPWARE:STARTNET DNS
```

When you start the name server, it creates a log file, TCPWARE:NAMESERVER.LOG. This file contains information about any problems uncovered when loading the database or during name server operation.

If the name server is already active and you edited the database files, you can restart the name server or you can reload it using the NETCU RELOAD NAMED command. See Chapter 2 of the *Network Control Utility (NETCU) Command Reference*.

You can investigate this file after updating the database or if the name server exits immediately after startup. Syntax errors in the database files are the most common source of errors. If there is an error in one of the database files, Domain Name Services records the error type and relevant database file in the log file.

Editing Database Files can help you find the cause of these errors.

The TCPWARE_NAMED_ROOT:NAMED.CONF File

The main DNS configuration file, from which the name server gets its initial data, is `TCPWARE_NAMED_ROOT:NAMED.CONF`. The equivalent of this file in UNIX-based BIND implementations is `/etc/named.conf`. Use this file to add information about your site when setting up a master DNS server. An example configuration file follows.

```
/*
** Sample Configuration File for DNS server
*/

options {
    directory "TCPWARE_ROOT:[TCPWARE.NAMED]";
//  forward only;
    forwarders { 128.0.1.1; 128.0.2.10; };
};

zone "example.com" in {
    type master;
    file "domain-name-service.iris";
};

zone "0.128.in-addr.arpa" in {
    type master;
    file "domain-name-service.iris-net";
};

zone "cc.example.com" in {
    type slave;
    masters { 128.0.1.1; };
    file "domain-name-service.cc";
};

zone "1.0.128.in-addr.arpa" in {
    type slave;
    masters { 128.0.1.1; };
    file "domain-name-service.cc-net";
};

zone "0.0.127.in-addr.arpa" in {
    type master;
    file "domain-name-service.local";
};

zone "." in {
    type hint;
    file "domain-name-service.cache";
};
```

The following sections describe the zone, options, and logging sections.

Zone

A *zone* is that part of a name server that contains complete information about the domain name space. You specify a zone is the following way:

```
zone "domain_name" [class] {
            type type;
        };
```

The below table defines the `NAMED.CONF` zone fields.

Field	Description
class	The class to which this zone applies. If the class is not specified, the type `IN` is used by default. The syntax is `[ (in | hs | hesiod) ]` • `in` (default) - Used for objects connected to the Internet. This is the only supported type. • `hs` or `Hesiod` - Confined mostly to MIT. `hs` is the abbreviation for hesiod.
"*domain-name*"	The name of the domain for which this zone is authoritative.
`file "`*`filename`*`";`	Specifies the name of the file.
`masters {ip_addr; [ip_addr; ... ]};`	Specifies the IP addresses from where the server is to transfer the zone data. This statement is meaningful only for slave or stub zones.
`type (master | slave | stub | hint);`	See the zone types table below for a description of these zones.

Optional Zone Statements

Statement	Description

`allow-query {address_match_list}; allow-transfer {address_match_list};`	Overrides the respective statement in the global options section for this zone. See *Logging Options*.
`allow-update {address_match_list};`	Specifies the addresses of hosts that are allowed to modify the zone with dynamic updates. Defaults to none.
`also-notify {ip_addr; [ip_addr; ...]};`	Lists the servers to send zone change notifications to as well as to the slave servers specified via the NS records for the zone.
`check-names (warn | fail | ignore);`	Overrides the default name checking specified in the global options section. See the `check-names` statement in *NAMED.CONF Options* **Error! Reference source not found.** for more details.
`notify (yes | no);`	Specifies if zone change notifications should be sent to the slave servers for the zone. This overrides the `notify` statement in the global options section. See the `notify` statement in *NAMED.CONF Options* for more details.

Zone Types

Type	Description
`hint`	Specifies that data in the `DOMAIN-NAME-SERVICE.CACHE` file, which is in standard resource record format, should be placed in the bootstrap cache. The hint zone definition is used to specify locations of root domain servers. An up-to-date list of root name servers is automatically obtained and stored in memory without replacing the cache file.
`master`	Specifies data for the zone and the domain. The first master zone definition states that the file `DOMAIN-NAME-SERVICE.IRIS` contains authoritative data for the `EXAMPLE.COM` zone, in standard resource record format.

	The second master zone definition states that the file `DOMAIN-NAME-SERVICE.IRIS-NET` contains authoritative data for the domain `0.128.IN-ARPA.ARPA`, which is used in translating addresses in network 128.0.0.0 to host names. Be sure each zone master file begins with an SOA (Start of Authority) resource record for the zone, as shown in the *DNS Zone Information Files* section.
`slave`	Specifies the zones for which this DNS server acts as a secondary name server. After this name server receives a "zone transfer," it becomes authoritative for the specified zone. The first slave zone definition specifies that all authoritative data under `CC.EXAMPLE.COM` is to be transferred from the name server at 128.0.1.1. The file statement in this section is the file name in which to back up the transferred zone. When it boots, the name server loads the zone from this backup file, if it exists, providing a complete copy even if the master DNS server is unreachable. This file is updated whenever a new copy of the domain is received by automatic zone transfer from one of the master servers. The file statement is optional but recommended to speed up server startup and eliminates needless bandwidth. The second slave zone definition states that the address-to-hostname mapping for the subnet 128.0.1.0 should be obtained from the same list of master servers as the previous zone.
`stub`	Works like a slave zone, except it transfers only the nameserver records for the master zone rather than the full zone information.

Options

The `options` statement sets up global options to be used by NAMED. Use this statement only once in a configuration file. If it is used more than once, the first occurrence determines what options to use, and

a warning is generated. If no `options` statement is present, an options block is used setting each option to its default value.

You specify `options` in the following way:

```
options {
options-statements
         };
```

The below table defines the `NAMED.CONF` options.

Option	**Description**
`allow-query {address_match_list};`	See the *Address_match_list* section. Specifies the addresses of hosts that are allowed to query the server for information. It defaults to all.
`allow-transfer {address_match_list};`	See the *Address_match_list* section. Specifies the addresses of hosts that are allowed to perform zone transfers from the server. It defaults to all.

`check-names (master | slave | response) (warn | fail | ignore);`	The server checks names in three areas:
	
	• Master zone files.
	• Slave zone files.
	• Responses to queries the server has initiated.
	
	The server assumes the following defaults:
	
	`options {`
	`  check-names master fail;`
	`  check-names slave warn;`
	`  check-names response ignore;`
	`};`

	• `ignore` - No checking is done. • `warn` - Names are checked against their expected client contexts. Invalid names are logged, but processing continues normally. • `fail` - Names are checked against their expected client contexts. Invalid names are logged, and the offending data is rejected. If `check-names response fail` has been specified, and answering the client's question requires sending an invalid name to the client, the server sends a `REFUSED` response code to the client.
`directory "`*`path`*`";`	Causes the server to change its default directory to the specified directory. This can be important for the correct processing of `$INCLUDE` files in primary zone files, or `file` statements in zone definitions.
`forward (only | first);`	This statement is meaningful only if there is a `forwarders` statement. When `first` (default) is used, the server queries the `forwarders` first before consulting the root domain servers. When `only` is used, the server queries the `forwarders` only. If the `forwarders` fail to find an answer, the server does not query the root domain servers.

	`options { forward only; forwarders {192.1.1.98; 192.1.1.99;}; };`
`forwarders {`*`ip_addr`*`; [`*`ip_addr`*`; ...]};`	Specifies the addresses of site-wide servers that accept recursive queries from other servers. If the DNS server configuration file specifies one or more `forwarders`, the server sends all queries for data not in the cache to the `forwarders`. Central name servers designated to handle forwarded requests can then develop a cache of answers to external queries. The central cache reduces the number of requests sent to root name servers and improves DNS performance.

`listen-on [`*`port ip_port`*`] { `*`address_match_list`*` };`	See the *`Address_match_list`* section.
	
	Specifies what port on what interface to listen on. The default is:
	
	`listen-on port 53 { any };`
	
	Example:
	
	`options { // listen on port 53 for // external interfaces. listen-on { 192.168.1.0;}; // listen on port 43 for // internal interfaces. listen-on port 43`

	`{ 127.0.0.1; 10.0.0.0; };` `};`
`max-transfer-time-in` *number*	Terminates the inbound zone transfers (`named-xfer` processes) running longer than the minutes specified. The default is 120 minutes (2 hours).
`notify (yes | no);`	If `yes` (default), the server notifies slave servers if there are any changes to a domain for which the server is master. The server determines the slave servers by the nameserver records contained in the zones data file. For more information, see the `also-notify` statement in **Error! Reference source not found.**.
`recursion (yes | no);`	If `yes` (default), the server attempts to do all the work required to answer a query that has requested recursion. Turning this off results in the server responding to the client with referrals. To prevent the server's cache from growing, use `recursion no;` in combination with `fetch-glue no;`.
`transfer-format` *number*	The server supports two zone transfer methods. `one-answer` uses one DNS message per resource record transferred. `many-answers` packs as many resource records as possible into a message. `many-answers` is more efficient, but is only known to be understood by BIND 8.1 and patched versions of BIND 4.9.5. The default is `one-answer`. `transfer-format` may be overridden on a per-server basis by using the `server` statement.

`transfers-in` *number*	The maximum number of inbound zone transfers that can be running concurrently. The default value is 10. Increasing `transfers-in` may speed up the convergence of slave zones, but it also may increase the load on the local system.
`transfers-per-ns` *number*	The maximum number of inbound zone transfers (`named-xfer processes`) that can be concurrently transferred from a given remote nameserver. The default value is 2. Increasing `transfers-per-ns` may speed up the convergence of slave zones, but it also may increase the load on the remote nameserver. `transfers-per-ns` may be overridden on a per-server basis by using the `transfers` phrase of the `server` statement.

Address_match_list

The following can be address match lists:

- an IP address (in dotted-decimal notation)
- an IP address match list
- an IP prefix (in /- notation)
- an address match list defined with the `acl` statement

The following ACLs are predefined:

- `any`
- `none`
- `localhost`
- `localnets`

Place the `!` character in front of elements you want to negate.

When an IP address or prefix is compared to an address match list, the list is examined and the first match (regardless of its negated state) is used. The interpretation of a match depends on the conditions defined in the following table.

When a list is being used...	A non-negated match...	A negated match...
as an access control list	allows access.	denies access. You can use the `listen-on` clause to define local addresses not normally used to accept nameserver connections.
with the topology clause	returns a distance based on its position on the list; the closer the match to the start of the list, the shorter the distance between it and the server.	A negated match is assigned the maximum distance from the server. If there is no match, the address gets a distance that is further than any non-negated list element, and closer than any negated element.

Since the address match list uses a first-match algorithm, care must be taken when using negation. In general, if an element is a subset of another element, the subset should be present in the list before the broader element.

For example, 10.0.0/24; !10.0.0.1 will never negate to the 10.0.0.1 address because a 10.0.0.1 address will match with the 10.0.0/24 element and not traverse any farther. So the 10.0.0.1 address will be accepted in the match list.

Using !10.0.0.1; 10.0.0/24 will elicit the desired effect. The 10.0.0.1 will be matched against the first, negated, element. All other 10.0.0.* addresses will pass by the 10.0.0.1 element and be matched against the 10.0.0/24 subnet element.

Logging

The `logging` section configures a wide variety of logging options for the nameserver. Its channel phrase associates output methods, format options and severity levels with a name that can be used with the category phrase to select how various classes of messages are logged. The basic `logging` syntax is as follows:

```
logging {
        channel channel_name {
            file pathname;
            severity type;
            print-category yes_or_no;
            print-severity yes_or_no;
            print-time yes_or_no;
        };
        category category_name {
channel_name; [ channel_name; ...]
        };
```

Only one logging section is used to define as many channels and categories as you want. If there are multiple `logging` sections in a configuration, the first one defined determines the logging, and warnings are issued for the others. If there is no `logging` section, the default logging configuration will be:

```
logging {
     category default { default_syslog; default_debug; };
     };
```

The following is an example:

```
channel xfers {
     file "TCPWARE:XFERS.LOG";
     severity info;
     print-severity yes;
     print-time yes;
};
     category xfer-in {
     xfers;
};
```

The below table describes the logging options.

Options	**Description**
`channel`	Specifies where the logging data goes: to syslog, to a file, to stderr, or to null.

`category`	Specifies what data is logged. You can send a category to one channel or to many channels. These are the valid categories:
	
	`default general client config database`
	`dnssec lame-servers network notify queries`
	`resolver security update xfer-in xfer-out`
	`lame-servers`

`file`	Specifies the path name of the file you want the message to go to.
`syslog daemon`	Specifies that the message goes to syslog (OPCOM) instead of to a file.
`severity`	Specifies the severity level for this channel. The severity choices are `critical`, `error`, `warning`, `notice`, `info`, `debug` *`level`*, and `dynamic`.

`print-category`
`print-severity`
`print-time`
	Specifies the category, severity level, and time stamp of the messages. The default is `NO` for each item.

Editing Database Files

The CNFNET configuration procedure creates templates of the database files. Some of the templates are only examples of the database records necessary for configuration. You must enter information specific to your configuration for the server to function properly. The type of server you configure determines which database file you need to edit.

> **Note:** If you edit the database files, restart the DNS software so that the name server can update its database.

Special Characters

The characters listed in the below table have special uses in the database files.

The character...	**Is used...**
`\`*`char`*	To quote a single character, *`char`*, that otherwise has special meaning. For example, use `\$` to place a dollar sign in the text.
`( )`	To group data that exceeds a line boundary. The SOA record requires parentheses. See the Start of Authority (SOA) resource record.

;	To start a comment. TCPware ignores the remaining characters in the line.
*	As a wildcard. For example, the domain name *.EXAMPLE.COM refers to any host within the EXAMPLE.COM domain.

Use a period (.) to end a domain name. If the domain name does not end with a period, DNS appends the current domain name to it. The NAMED.CONF file defines the current domain name, or you can use the $origin command to redefine it.

Special Commands

You can use two special commands in database files:

- $origin
- $include

These commands are described on the following pages.

$origin

Indicates that all the records following the command belong to a different domain than the previous records. The name server has authority over all records listed. This command gives you a shorthand way of expressing the domain name of the host.

Format

```
$origin domain-name
```

Parameter

domain-name

Domain name to which the records belong.

Example

This example defines the DAISY.EXAMPLE.COM, IRIS.EXAMPLE.COM, SPARROW.EXAMPLE.ORG, DOVE.EXAMPLE.ORG, MAPLE.EXAMPLE.EDU, and ACORN.EXAMPLE.EDU domains.

```
;the default domain is example.com.
daisy       in    a    192.168.62.1
iris        in    a    192.168.62.2
$origin example.org.
sparrow     in    a    192.168.95.1
dove        in    a    192.168.95.3
$origin example.edu.
maple       in    a    192.168.74.1
acorn       in    a    192.168.74.2
```

$include

Includes an external file in a DNS database file. Useful for organizing different types of information into separate files.

Format

```
$include file [origin]
```

Parameter

file

File you want to include in the database.

origin

(Optional) Origin of the include file (see the `$origin` command on the previous page).

Examples

1. This command includes the file `TCPWARE:MAILLIST.TXT` in the database file.

```
$ include tcpware_named_root:maillist.txt
```

2. These commands include the file `TCPWARE:EXAMPLE.ORG` in a series of definitions, using the EXAMPLE.ORG domain as origin (see the `$origin` command) for the include file.

```
$ origin example.com.
daisy
iris
$ include tcpware_named_root:example.org example.org.
rose
```

If the `EXAMPLE.ORG` file includes records for SPARROW and DOVE, the extended version of the above definition would be:

```
daisy.example.com
iris.example.com
sparrow.example.org
dove.example.org
rose.example.com
```

Note that the include file origin definition is for the include file only and does not affect the "external" origin.

Resource Records

All database files contain entries called resource records (RRs). TCPware supports all of the resource record types supported by BIND. Some of the more commonly used resource records are detailed below.

This section provides the following information for each DNS resource record:

- Purpose
- Format
- Field definitions
- Example of usage

The fields that are common to most resource records are: `owner`, `ttl`, `class`, and `type`. The remaining `data` fields are different for each record type. This section documents these `data` fields under the appropriate record type.

All resource records are case insensitive. However, TCPware preserves the case you enter.

Format

`owner ttl class type data`

Fields

`owner`

Domain name of the owner of the record.

The domain name can be absolute or relative. An absolute domain name lists all the labels of the name and ends with a period. For example, `DAISY.EXAMPLE.COM.` is an absolute domain name. A relative domain name does not end with a period. DNS assumes it belongs to the current domain. For example, `DAISY` is a relative domain name in the EXAMPLE.COM domain.

Acceptable characters are A through Z (upper or lower case), 0 through 9, and dash (-). The period (.) is a label separator.

The values listed below have special meaning in the *`name`* field.

Value	Description
(all blank)	The resource record applies to the last explicitly stated domain name
.	Indicates the root (or top level) domain

`@`	Indicates this is the current domain name

`ttl`
Time-to-live (TTL).

This is the length of time (in seconds) the record is valid after a requestor host receives it from a primary server. For example, a TTL of 86400 equals 24 hours.

You can specify in the time-to-live field in the following ways (each of these is equivalent to one week):

- 604800
- 1w
- 7d
- 168h
- 10080m
- or any combination

For example:

```
sigma 2h46m40s IN A 10.1.1.97
```

Loads the TTL as: `ttl = 10000 (2 hours 46 mins 40 secs)`

If you leave this field blank, DNS uses the TTL designated in the SOA (start of authority) record *`minimum`* field. The *`minimum`* field value is the "default" TTL.

All resource records that have the same values in the *`name`*, *`class`*, and *`type`* fields should also have the same value in the *`ttl`* field.

`class`
Address class (it should be `IN` for "internet").

`type`
Resource record type. Some of the most commonly used ones are listed below and described more fully on the following pages.

Type	**Description**	**Type**	**Description**

A	IPv4 Address	MX	Mail exchange
AAAA	IPv6 Address	NS	Name server
CNAME	Canonical name	PTR	Pointer
DHCID	DHCP client ID	SOA	Start of authority
DNSKEY	DNSSEC key	SPF	SPF information
HINFO	Host information	TSIG	Transaction Signature
IPSECKEY	IPsec key	TXT	Descriptive text

`data`
Data specific to each entry. This field varies with each type of resource record.

A

Address record, or internet address of a host. The name server uses this record when it responds to a query for an internet address.

Use this record in any of the database files.

Format

owner ttl class A *address*

Fields

A

The *type*, which must be A.

address

Internet address of the host specified in the *name* field.

Example

This example includes two A records for local hosts DAISY and LILAC.

```
;name     ttl         class     type    address
daisy                 IN        A       192.168.95.3
lilac     99999999    IN        A       192.168.95.4
```

CNAME

Canonical Name (CNAME) record, or official name of the host. You can include a nickname, or if you rename the host, use the nickname field to give the old domain name.

Format

nickname ttl class CNAME *canonical-name*

Fields

nickname

Nickname or alias for the host. If you rename the host, this is the old domain name.

CNAME

The *type*, which must be CNAME.

canonical-name

Official domain name for the host. This can be an absolute or relative name. If you rename the host, this is the new domain name.

Example

This example includes two CNAME records. These records define SPRING as a nickname for host LILAC.EXAMPLE.COM and SUMMER as a nickname for host DAISY.EXAMPLE.COM. Because no period (.) follows the nicknames, DNS assumes they are in the current domain.

```
;nickname    ttl    class     type    canonical-name
spring              IN        CNAME   lilac.example.com.
summer              IN        CNAME   daisy.example.com.
```

HINFO

Host Information (HINFO) record, or hardware type and operating system of a host. DNS uses this information to answer queries.

Each host in a domain can have just one HINFO record.

Format

owner ttl class HINFO *hardware opsys*

The *hardware* and *opsys* fields require a space between them. If you need to use a space within either field, enclose the field within quotation marks (" ").

Fields

HINFO

The *type*, which must be HINFO.

hardware

Type of CPU. See the latest *Assigned Numbers* RFC for a list of standard hardware types.

opsys

Host operating system.

Example

This example includes two records for hosts IRIS and LILAC in the current domain and gives their hardware type and operating system. Some field entries include quotation marks because they contain space characters.

```
;owner    ttl    class    type   hardware              opsys
iris             IN       HINFO  "AlphaStation 8400"   VMS
lilac            IN       HINFO  "VAXStation 4000"     "VMS V7.0"
```

MX

Mail Exchanger (MX) record, or a host that can accept mail for another host. A host can have multiple MX records. Each record receives a preference value.

When a mailer tries to deliver mail to a host:

1. It reads all MX records defined for the destination host and sorts them by preference value.
2. It tries to deliver the mail to the host specified on the record with the highest preference. The record with the lowest value (beginning at 0) has the highest preference.
3. If the first attempt fails, it tries the host specified on the record with the next highest preference value.
4. It keeps trying until it delivers the mail, or until it tries the host specified on the record with the lowest preference.

If you assign the same preference value to multiple MX records for a host, the mailer tries the equally-preferenced records in random order.

See Chapter 17, *Managing Mail Services*, the *Completing SMTP/MR Configuration* section for MX records used with the SMTP mail facility.

Format

system ttl class MX *preference gateway*

Fields

system

Domain name of the host the *gateway* host accepts. The host might not be connected directly to the network. Using wildcards in domain names is strongly discouraged if the hosts are Internet-connected (see section 2.7 of RFC 1912 for details).

MX

The *type*, which must be MX.

preference

Delivery order when a host has multiple MX records. The lower the number (starting at 0), the higher the preference.

gateway
Name of the host accepting mail for the host specified in the *system* field.

Example

This example gives three MX records for host TULIP (in the current domain). The mailer tries to deliver the mail to host TULIP.EXAMPLE.COM first, because that record has the lowest *preference*-value. If the attempt fails, it tries host IRIS.EXAMPLE.COM and then LILAC.EXAMPLE.COM.

```
;system    ttl    class    type    preference    gateway
tulip             IN       MX      10            tulip.example.com.
tulip             IN       MX      15            iris.example.com.
tulip             IN       MX      20            lilac.example.com.
```

NS

Name Server (NS) record that lists the domain name of a host that provides domain name services, and the name of the domain being served. Therefore, the specified host is an authoritative name server for the specified domain.

You can enter NS records in any database file.

Format

owner ttl class NS *server*

Fields

NS

The *type*, which must be NS.

server

Domain name of the host that serves the domain.

Example

This example gives the syntax of three NS records. DAISY.EXAMPLE.COM and IRIS.EXAMPE.COM are both servers for the EXAMPLE.COM domain. The *owner* field is blank for IRIS.EXAMPLE.COM, indicating it serves the domain specified in the previous record. The *owner* field for NS.NASA.GOV server contains only a dot (.), indicating NS.NASA.GOV is a root server. IRIS.EXAMPLE.COM takes its time-to-live (TTL) value from the *min* field of the SOA record. The TTL for NS.NASA.GOV is 99999999 seconds (approximately three years).

```
;owner     ttl      class      type       server
example.com.        IN         NS         daisy.example.com.
                    IN         NS         iris.example.com.
.          99999999 IN         NS         ns.nasa.gov.
```

PTR

Domain Name Pointer (PTR) record that allows special names to point to another location in the domain. The most common use of PTR records is for reverse mapping: Domain Name Services finds a host domain name when given an internet address. The IN-ADDR.ARPA domain maintains reverse mapping information.

PTR records in the IN-ADDR.ARPA domain contain a host name that consists of the internet address specified in reverse order, combined with the IN-ADDR.ARPA domain name. This name points to the domain name of the host with that internet address.

Enter PTR records in the `NAMED.REV`, `NAMED.HOSTS`, or `NAMED.LOCAL` files.

Format

`rev-addr ttl class` `PTR` *`realname`*

Fields

`rev-addr`

Combination reverse internet address and domain name IN-ADDR.ARPA. Each *`rev-addr`* should be unique to the zone.

`PTR`

The *`type`*, which must be `PTR`.

`realname`

Full domain name of the host. If the host is not in the current domain, this name must end in a period (.). Do not use a nickname.

Example

This example gives PTR records for two hosts. The internet addresses are in the current domain, 95.168.192.IN-ADDR.ARPA.

```
;rev-addr   ttl    class    type    realname
$origin 95.168.192.in-addr.arpa
```

```
2                       IN        PTR      daisy.example.com.
4                       IN        PTR      lilac.example.com.
```

SOA

Start of Authority (SOA) record that defines the start of a zone. There is one SOA record for each zone, and it is on the primary server. If other servers in the zone have SOA records, these records must be identical to the one on the primary server. The SOA record is the first one listed in a database file.

You can enter this record in any database file. The `NAMED.CA` file can store an SOA record, but the record does not define the server as authoritative.

Format

`owner ttl class` `SOA` *`origin person`* (*`serial refresh retry expire minimum`*)

The parentheses are required if continuing onto one or more subsequent lines. At least one space must separate the parentheses from the text within it.

Fields

`SOA`
The *`type`*, which must be `SOA`.

`origin`
Name of the host on which the primary server resides.

If the local host is not the primary server, the local host periodically obtains database information from the specified host. See the *`refresh`*, *`retry`*, and *`expire`* fields.

`person`
Mailbox address of the person responsible for the DNS software on the local domain. Replace the `@` sign in the mailbox address with a period (`.`); for example: `gardener@iris.example.com` becomes `gardener.iris.example.com.`.

`serial`
Version number of the database file. A 32-bit unsigned integer that can theoretically start at 0.

Increment this field by a certain interval each time you edit a database file (using the YYYYMMDDVV date syntax provides a "safe" interval). If *`serial`* on the primary server is "higher" (based on serial

number arithmetic) than *serial* on the secondary server, the secondary server knows that the primary server contains new data and it performs a zone transfer to update its database. The serial number also tells the DNS software which of two file copies is the most recent.

refresh
Time interval (in seconds) after which the secondary server must request the SOA record from the primary server. For example, a refresh value of 86400 equals 24 hours. A value of 900 seconds (15 minutes) is the minimum value allowed.

retry
Time interval (in seconds) after which the secondary server should re-request the SOA record from the primary server after a refresh failure. 600 seconds (10 minutes) is a reasonable value.

expire
How long (in seconds) the secondary server can use its copy of the database file when it cannot obtain a refresh. A typical value is 3600000 seconds (approximately 41 days and 16 hours).

minimum
Minimum time to live (TTL) value, in seconds, for the records in the zone. DNS uses this value if you do not specify the *ttl* field for other resource records. A reasonable value is 86400 seconds (24 hours).

Example
This example shows a typical SOA record format. The values are described in the table.

```
;owner  ttl  class  type  origin            person
@            IN     SOA   iris.example.com. gardener.iris.example.com.(
                    1            ; serial
                3600             ; refresh (1 hour)
                 600             ; retry (10 minutes)
              3600000            ; expire (1000 hours)
              86400 )            ; minimum (24 hours)
```

Value	Description
@	Current domain name

`iris.example.com.`	Primary server host name
`gardener.iris.example.com.`	"Owner" of the DNS software on the local domain (the mailbox address gardener@iris.example.com becomes gardener**.**iris.example.com.)
`1`	Serial (version) number of the database file
`3600`	Refresh time – the secondary server requests an SOA record from the primary server every 3600 seconds (1 hour)
`600`	Retry time – the secondary server retries requests for the SOA record from the primary server every 600 seconds (10 minutes) if a refresh fails
`3600000`	Expiration time – the secondary server can use its copy of the database if all refreshes fail for a total of 360000 seconds (1,000 hours, or 41 days and 16 hours)
`86400`	Minimum time-to-live of 86400 seconds (24 hours) for records in the zone

The parentheses at the end of the second line indicate that one or more additional lines related to the record follow. The lines usually include the *`serial`*, *`refresh`*, *`retry`*, *`expire`*, and *`minimum`* field values and their commented out (`;`) descriptions. You must include a space between the parentheses and the text it encloses (such as by indenting the next line). You must also include a white between the last value (86400 in the example) and the closing parentheses.

TXT

Text (TXT) record. Holds descriptive text. The semantics of the text depend on the domain.

Format

```
owner ttl class TXT txt-data
```

Fields

TXT

The *type*, which must be TXT.

txt-data

One or more character strings of descriptive text.

Troubleshooting Domain Name Services

Access error messages help by entering **`HELP TCPWARE MESSAGES [identifier]`**.

When you start the name server, it creates a log file, `TCPWARE:NAMESERVER.LOG`. This file contains information about any problems uncovered when loading the database or during name server operation. TCPware sends system logging errors to this file. By default, TCPware logs all errors except Debug messages to the log file.

Dynamic TCP/IP Load Balancing

TCPware provides TCP/IP load balancing services for a TCP/IP cluster that are analogous to the load balancing services the LAT terminal service provides.

When a new TCP connection to a cluster name occurs, the TCPware Domain Name Services name server assigns the connection to one of a number of hosts. The host to which it assigns the connection depends on:

- The availability of the host.
- The observed load on the host.

TELNET most often uses TCP/IP cluster load balancing, although other TCP protocols do also. UDP-based protocols also work well with cluster load balancing, but only if:

- The server (such as DNS) does not retain state information.
- The client (such as TFTP) resolves the domain name only once at the start of a connection.

TCP/IP load balancing does NOT:

- Provide effective NFS server failover. Most clients do not resolve names again and remount filesystems when an NFS server fails to respond.
- Provide local preference to clients' selection of hosts.
- Actively re-balance the load: a failed and recovered host receives only new connections.
- Support non-TCPware hosts as part of the cluster.

In addition, the default metric is not very useful for RPC type services. It is oriented toward measurement of users.

Configuration Requirements

All hosts in the TCP/IP cluster must run TCPware. Also, all DNS servers for the zone defined with the TCP/IP cluster name must be on systems running TCPware. Client systems do not have to run a particular TCP/IP implementation.

The term *cluster* here means a *TCP/IP cluster*. Hosts in a TCP/IP cluster do not have to be part of a VMScluster. They even do not have to be on the same bridged LAN.

In a TCP/IP cluster, the hosts can be at least one of the following:

- Independent systems with TCP/IP connectivity
- Located anywhere so long as there is TCP/IP connectivity
- Part of several VMSclusters with TCP/IP connectivity

You can define multiple cluster names describing subsets, or overlapping or separate clusters.

Load Balancing Process

When a client wants to connect to a host within the cluster:

1. It sends the DNS server a name-to-address translation request for a host to which it wants to connect.
2. The DNS server looks in a cache that holds recent load information for the hosts in the cluster. If the name is a cluster name, a routine sorts the addresses by reported load. The server determines the load by exchanging UDP datagrams with each host in the cluster, which report their current load metrics. The server treats hosts that fail to respond as unavailable and does not offer them any new traffic. Specifically:
    a. The DNS server searches its resource record for host addresses and looks up each host in a private list of hosts. Different cluster names share this list. If a host appears in more than one cluster, the server requests its load metrics only once.
    b. The DNS server sends update requests to the hosts if there is no information or if there is outdated information in the cache.
    c. If a host fails to respond to the load information requests, it does not return its address to the server. In this case, the host could be down.

       Alternatively, the host also could have been administratively shut down by removing the UDP service that responds to load requests. Removing this UDP service effectively removes the host from the cluster.

    d. TCPware moves the address record corresponding to the host with the lowest load metric to the front of the DNS information list.

3. The server responds to the client with a list of addresses in preferred order of use. Most clients use the first address, or if it fails, second or subsequent addresses.
4. When DNS returns its reply to the client, it:
   - Rotates the first address down the list among hosts with similar load metrics. This means that DNS "round-robins" calls among similarly loaded hosts.
   - Sets the time to live (TTL) to a small value. This forces a new request for subsequent connections.

Cluster Names

In DNS, you define each cluster name as an ordinary host name with an IP address and resource records for each host address used. There are straightforward, overlapping, and subzone clusters.

Straightforward Clusters

The below example shows a cluster name defined as a DNS address (A) resource record as part of a zone file. The cluster is called `perennials` and is assigned an IP address of 192.168.3.50.

```
@               IN      SOA     rose.example.com. system.rose.example.com. (
                                   1        ; Serial
                                   3600     ; Refresh
                                   600      ; Retry
                                   3600000  ; Expire
                                   86400 )  ; Minimum
                IN      NS      rose.example.com.
rose            IN      A       192.168.3.50
lilac           IN      A       192.168.3.51
petunia         IN      A       192.168.3.52
hydrangea       IN      A       192.168.3.53
perennials      IN      A       192.168.3.50
                IN      A       192.168.3.51
                IN      A       192.168.3.52
                IN      A       192.168.3.53
```

As the DNS server consults its cache, it matches the cluster name against a list. If the name is in the list, TCPware sorts the addresses in the list by reported load.

After setting up the cluster name in the zone file, make sure the cluster name is known to DNS by reconfiguring DNS using `CNFNET`. During the configuration of DNS, you are asked if you would like to configure a list of cluster names.

Overlapping Clusters

The below example shows two clusters defined in a zone file. Note that the 192.168.100.2 address is common to both clusters.

```
$ORIGIN example.com
orders    IN  A   192.168.100.1
          IN  A   192.168.100.2
invoices  IN  A   192.168.100.2
          IN  A   192.168.100.4
```

Subzone Clusters

In some cases, you may want to load balance your cluster using an external name server instead of a local one. Since the external server cannot configure an internal load balanced cluster, the primary server must delegate authority on a subdomain to the internal server. The internal server then becomes primary for the subzone, which becomes the actual address of the cluster.

The following set of examples show three systems in a cluster: 10.0.0.1, 10.0.0.2 and 10.0.0.3. The domain is example.com. The first step is to set up a subdomain on the primary server by editing the zone file for the example.com domain and adding the line to delegate the authority to the internal server, homerdns.example.com:

```
homer     IN  NS  homerdns.example.com.
```

The next step is to set up `homerdns.example.com` as a primary name server for domain `homer.example.com`.

The zone file for `homer.example.com` on `homerdns.example.com` should include the lines below:

```
cluster  IN  A   10.0.0.1
         IN  A   10.0.0.2
         IN  A   10.0.0.3
```

There is now a load balanced cluster set up to be `cluster.homer.example.com` that is accessible from both the primary server (by delegation) and the internal server.

Finally, map `cluster.example.com` to the load balanced cluster. Add the following line to the `NAMED.HOSTS` equivalent file on the primary server:

```
cluster  IN  CNAME  cluster.homer.example.com.
```

The primary server now serves out the addresses from the zone file in load balanced order.

Load Request Protocol

When the DNS server finds that its load information for a host is out of date, it sends a UDP datagram to the host asking for load updates.

Each UDP request starts a timed sequence in the host. This causes the host to send updates to the DNS server at specified intervals over a set time period. When the sequence ends, the DNS server considers the information stale and sends a new request. This procedure:

- Minimizes traffic when the DNS server is heavily loaded (for example, handling more than 100 requests per second from clients).
- Is quiet when there are no requests.

The procedure does not require hosts to maintain more than a transient state on the DNS servers, since if they fail, they simply cease to respond.

If a host is part of multiple clusters, the DNS server makes the load request once and not for each separate cluster.

The host normally provides the load reply service from within NETCP. However, you can do this through a configured UDP service using the definition:

```
NETCU> ADD SERVICE METRIC UDP /ROUTINE=REPORT_TCLB_METRIC
```

LAT looks at the number of processes in COM state and uses that information to calculate its metric. LAT determines the TCP/IP cluster load balancing metric from the number of active users on the system.

Part of the metric consists of a value that is set for each host. You set this value by defining the system logical `TCPWARE_ TCLB_BIAS` with a multiplier and an addend as two values of the logical. Both are real numbers. TCPware uses the values in computing the reported load.

You can also use these values to bias a load offered to the host. For example, the following command doubles the observed load and adds 1.5 users:

```
$ DEFINE/SYSTEM TCPWARE_TCLB_BIAS "2.0","1.5"
```

A cluster might consist of four hosts with one running other tasks. This host should not receive its full share. You can set the values to cause that host to report a higher load.

TCPware re-translates the `TCPWARE_TCLB_BIAS` logical before it sends each response. This means that some other process can change it dynamically or you can set it statically.

`SET LOGIN/INTERACTIVE=0` forces TCPware cluster load balancing to report the node as an extremely high load (2147483647).

4. DHCP/BOOTP Server

Introduction

This chapter describes the DHCP/BOOTP Server. It combines the Dynamic Host Configuration Protocol (DHCP) server with the Bootstrap Protocol (BOOTP). DHCP allows hosts on a TCP/IP network to request and be assigned IP addresses, and also to discover information about the network to which they are attached. BOOTP provides similar functionality, with certain restrictions.

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

DHCP and BOOTP

In TCPware, the DHCP Server (DHCPD) is combined with the BOOTP Server (BOOTPD) to form the DHCP/BOOTP Server (DHCPD/BOOTPD).

DHCP

DHCP is the Dynamic Host Configuration Protocol. It centralizes and automates TCP/IP network configuration. The DHCP Server dynamically allocates IP addresses for hosts on the network from an available pool of addresses. In this way, new hosts or hosts that are frequently relocated can automatically get new IP addresses for a certain lease period. For this to work, the network administrator allocates address pools in each subnet and enters them into the DHCP configuration file (`DHCPD.CONF`).

DHCP is an extension of the Internet Bootstrap Protocol (BOOTP). DHCP offers a network host a temporary lease rather than an ownership of an IP address. The lease identifies the duration for which the client can safely use its dynamically assigned IP address. Lease lengths generally depend on the number of network users (crowding of the network) and the number of available IP addresses the DHCP

server can assign. The network manager sets the lease length through parameters in the DHCPD.CONF file.

BOOTP

BOOTP support is also provided by this server. BOOTP uses UDP to allow diskless systems to find their IP addresses, addresses of boot servers, and names of boot files. BOOTP can supply other client information, such as the addresses of name servers, gateways, and LPD servers.

Unlike DHCP, the BOOTP protocol does not provide a protocol for recovering dynamically assigned addresses once they are no longer needed. It is still possible to dynamically assign addresses to BOOTP clients, but some administrative process for reclaiming addresses is required. By default, leases are granted to BOOTP clients in perpetuity, although the network administrator may set an earlier cutoff date or a shorter lease length for BOOTP leases. BOOTP clients may also be served in the old standard way, which is to simply provide a declaration in the DHCPD.CONF file for each BOOTP client, permanently assigning an address to each client.

Dynamic Host Configuration Process

On startup, the DHCP server reads the DHCPD.CONF file and stores a list of available addresses on each subnet in memory. When a client requests an address using the DHCP protocol, the server allocates an address for it. Each client is assigned a lease, which expires after an amount of time chosen by the administrator (by default, one day). Before leases expire, the clients to which leases are assigned are expected to renew them in order to continue to use the addresses. Once a lease has expired, the client to which that lease was assigned is no longer permitted to use the leased IP address.

To keep track of leases across system reboots and server restarts, the DHCP server keeps a list of leases it has assigned in a lease file (DHCPD.LEASES). Before the server grants a lease to a host, it records the lease in this file and makes sure that the contents of the file are flushed to disk. This ensures that even in the event of a system crash, the server will not forget about a lease that it has assigned. On startup, after reading the configuration file, the server reads the leases file to refresh its memory about what leases have been assigned.

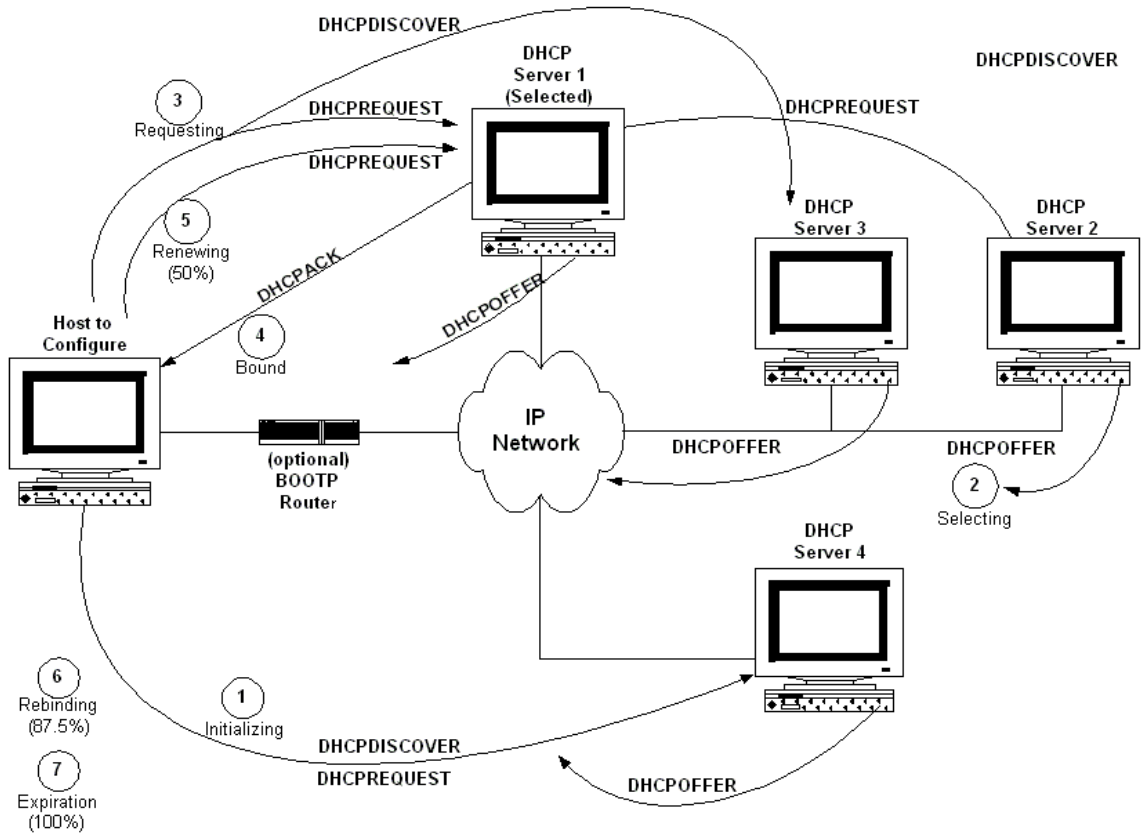
DHCP Protocol

With respect to the DHCP protocol, the DHCP server goes through an initializing, selecting, requesting, binding, renewal, rebinding, and expiration cycle when negotiating for an IP address, as shown in the below diagram. The process is basically 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 (Server 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 (acknowledgment) 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 acknowledgment message, the client must give up the IP address. It then returns to the *initializing* state and must start the address negotiation over again.

DHCP is defined in RFC 2131 and RFC 2132. Refer to them 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.



DHCP Administration

You can administer the DHCP server using the following TCPware Network Control Utility (NETCU) commands.

Command	Description
`RELEASE DHCP4 ip-address`	Forces the DHCP server to act as if it heard a DHCPRELEASE message from the client for the given IP address.
`REMOVE DHCP4 ip-address`	Synonym for `RELEASE DHCP4`
`SET DHCP4/DEBUG=value`	Sets the debug logging level to the given value.

SET DHCP4/NEWLOG	Starts a new debug log file.
SET DHCP4/PARTNERDOWN	For Failover DHCP: causes the DHCP server to transition into Partner Down state.
STOP/DHCP4	Causes the server to shut down.
SHOW DHCP4/ALL	Displays SHOW DHCP4/SUBNET output for all subnets, plus information about static assignments.
SHOW DHCP4/CLIENT_IDENTIFIER=*client-id*	Displays all lease binding and static assignment details for the given client ID.
SHOW DHCP4/CONFIGURATION	Writes all configuration and lease information to a dump file.
SHOW DHCP4/HARDWARE_ADDRESS=*hardware-address*	Displays all lease binding and static assignment details for the given hardware address.
SHOW DHCP4/IP_ADDRESS=*ip-address*	Displays lease binding details for the given IP address. Static assignments are not supported.
SHOW DHCP4/ISKNOWN *host*, *subclass*	If *host* is specified, shows whether the given hardware address or client identifier is "known", that is if there is a host declaration for that hardware address or client identifier. If *subclass* is specified, shows whether the given subclass data exists as a subclass within the given class.
SHOW DHCP4/LEASES	Displays brief information about each lease.
SHOW DHCP4/POOLS	Displays address pool availability.
SHOW DHCP4/STATUS	Checks if the DHCP server is running.
SHOW DHCP4/SUBNET=*ip-address*	Displays brief information about each IP address in the same shared network as the given IP address.

SHOW DHCP4/VERIFY	Checks the syntax of the configuration file and optionally the lease file and the update file.
SHOW DHCP4/VERSION	Displays the version of the DHCP server.
UPDATE DHCP4	Instructs the Dynamic Host Configuration Protocol (DHCP) server to process the update file and add or remove the specified host and subclass declarations.

See the *TCPware NETCU Command Reference* for details about these commands.

DHCP Service Configuration

You can configure the DHCP server using CNFNET, by typing **@TCPWARE:CNFNET DHCP4**.

You can configure the following items:

- Enable or disable the DHCP server.
- Set the debug logging level.
- Set the debug log file name.
- Include the date in the log file or not.
- Log debug messages to OPCOM or not.

See the DHCP configuration description in the *Installation and Configuration Guide.*

Verifying the DHCP Configuration

Whenever you modify the configuration file, it is good practice to verify the syntax by entering the following NETCU command:

```
$ NETCU SHOW DHCP4 /VERIFY[=(config=<config-file>)] -
_$ [/OUTPUT=<output-file>]
```

This command causes the DHCP server to run enough to read and parse the configuration file. The DHCP server displays a copyright notice and a message for each syntax error encountered. If the DHCP server displays only the copyright notice, the configuration file has no syntax errors.

The CONFIG option optionally specifies where the configuration file is. If you do not specify the CONFIG option, the DHCP server reads the default configuration file TCPWARE:DHCPD.CONF.

The /OUTPUT qualifier optionally sends command output to the specified file.

Reloading the DHCP Configuration

If you modify TCPWARE:DHCPD.CONF after starting the DHCP server, reload the DHCP configuration by restarting the DHCP server by using @TCPWARE:RESTART DHCP4. When the DHCP server restarts, it rereads the configuration file.

Upgrading to DHCP V4 from DHCP V3

Versions of TCPware prior to 6.1 provided both DHCP V4 and a legacy DHCP V3 implementation. DHCP V3 has been deprecated since TCPware 5.9, and is no longer provided. If you're still running the legacy DHCP V3 server, an easy upgrade path is available.

The DHCP V3 and V4 configuration files are substantially similar. Any changes that are needed can be made by hand, or alternatively by an automated conversion program (described below). The biggest changes between the DHCP V3 server configuration file and the DHCP V4 server configuration file are in the areas of *Dynamic DNS Updates (DDNS)* and *Failover*. This is due to the fact that in DHCP V3, DDNS and Failover are Process Software implementations, and in DHCP V4, they are the ISC implementations. Other changes in DHCP V4 are:

- Process Software's Host Name Generation function is no longer available. In DHCP V4, various evaluation functions can be used in the configuration file to ask the server to create a host name.
- The configuration file statement allow/deny ras-servers has been removed. This statement was deprecated in a previous release and has been removed altogether in DHCP V4.

The DHCP V3 server lease file is most likely not going to be able to be read by the DHCP V4 server. It may be able to be read if you are using a simple configuration. It will definitely not be able to be read if you are using DDNS or failover. In most cases, you should expect that when you upgrade to DHCP V4, you will lose the entire database of leases. In that case, all DHCP clients will have to obtain new leases. To avoid conflicts, it is recommended that prior to the upgrade, the length of leases being given out by the DHCP server be made very short, so that all leases expire during the changeover. You may also want to consider initially using a different range of addresses after the upgrade to doubly ensure that the DHCP V4 server does not attempt to give out leases for IP addresses that are still in use.

Using the DHCP3 to DHCP4 conversion program

The DHCP3 to DHCP4 conversion program (TCPWARE:DHCP3TO4.EXE) will convert information in the deprecated DHCP3 configuration to the DHCP4 format. It reads the configuration and boot file and produces a single file with an equivalent DHCP4 configuration. Comments are NOT preserved. The

program takes two parameters – the input (DHCP3) configuration file and the output (DHCP4) configuration file.

The default values for the parameters are `TCPWARE:DHCPD.CONF` and `DHCPD4.CONF` (created in the local directory, if not specified to be elsewhere). It will look for the `TCPWARE:DHCPD.BOOT` file to determine if failover is configured and convert the failover information as well. The lease file is not converted, so it is still necessary to have a quiet period when changing to DHCP4 from DHCP3.

```
$ dhcp3to4 :== $tcpware:dhcp3to4
$ dhcp3to4 [tcpware:dhcpd.conf] [dhcpd4.conf]
```

It is recommended that you check the configuration with `DHCPD4` before attempting to use it:

```
$ dhcpd4 :== $tcpware:dhcpd4
$ dhcpd4 -t -cf dhcpd4.conf
```

Introducing the Configuration File

TCPware supplies a template DHCP configuration file that contains comments and a number of examples to help you enter information for your hosts. The template DHCP configuration file is located at `TCPWARE:DHCPD4_CONF.TEMPLATE`.

Using this template as a guide, create a DHCP configuration file at `TCPWARE:DHCPD.CONF` (with any text editor) containing the entries you need for your network and hosts. The `dhcpd.conf` file is a free-form ASCII text file. The file may contain extra tabs and new lines for formatting purposes and comments may be placed anywhere within the file (except within quotation marks). Comments begin with the `#` character and end at the end of the line. Keywords in the file are case-insensitive.

> **Note:** Whenever changes are made to the `dhcpd.conf` file, the DHCP server must be restarted.
>
> The file consists of a list of statements specify which fall into two categories: parameters and declarations.

Parameters

Parameter statements always specify one of the following:

- How to do something (for example, how long a lease to offer)
- Whether to do something (for example, should the DHCP server provide addresses to unknown clients)
- What parameters to provide to the client (for example, use gateway 10.177.244.7)

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), then where possible 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:

```
option routers 10.254.239.1;
```

Note that the address is specified numerically; this is not required. If you have a different domain name for each interface on your router, it is appropriate to use the domain name for that interface instead of the numeric address. However, there may be only one domain name for all 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 (for example, 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 wherever 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

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 subnet. A `subnet` declaration is required for each subnet even if no addresses are dynamically allocated on that subnet.

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

- For clients with dynamically assigned addresses, a `range` declaration must appear within the `subnet` declaration, or within 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 the two networks in a `shared-network` declaration.

Note that even when the shared-network declaration is absent, an empty one is created by the server to contain the subnet (and any scoped parameters included in the subnet). For practical purposes, this means that "stateless" DHCP clients, which are not tied to addresses (and therefore subnets) will receive the same configuration as stateful ones.

Some sites may have departments that have clients on more than one subnet. 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 the declarations in a `group` declaration using the parameters that are common to that department.

- For clients with dynamically assigned addresses, group parameter assignments by network topology. Alternately, host declarations can provide parameters and if they have no fixed-address parameter, the clients get an address dynamically assigned. See the example below.
- `class` declarations and conditional declarations may be used may be used to group parameter assignments based on information the client sends.

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.

Each of these declarations itself appears within a lexical scope, and all declarations at less specific lexical scopes are also consulted for client option declarations. Scopes are never considered twice, and if parameters are declared in more than one scope, the parameter declared in the most specific scope is the one that is used.

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. If no such entry is found, it looks for an entry with no fixed-address parameter.

Example

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. You could have `host` declarations for each server and group them by model:

```
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; }
}
```

Dynamic 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 that IP addresses that have never been assigned are chosen over those that have previously been assigned to other clients. If an address is found that the client is permitted to have, and that has never been assigned to any client before, the address is immediately allocated to the client. If the address is available for allocation but has been previously assigned to a different client, the server will keep looking in hopes of finding an address that has never been assigned to a client.

Note also that the DHCP server generates the list of available IP addresses from a hash table. This means that the addresses are not sorted in any order. It is not possible to predict or control the order in which the DHCP server allocates IP addresses.

Renewing

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.

Fixed Addresses

There may be a host declaration matching the client's identification. If that host declaration contains a `fixed-address` declaration that lists an IP address that is valid for the network segment to which the client is connected. In this case, the DHCP server will never do dynamic address allocation. In this case, the client is required to take the address specified in the host declaration. If the client sends a DHCPREQUEST for some other address, the server will respond with a DHCPNAK.

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 be 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, if they are on the same shared network.

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.

As described in the *Dynamic Address Allocation* section, the DHCP server checks each IP address pool in sequence 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. If the client is allowed to use the pool, the server chooses an available address from that pool (if any) and tentatively assigns that address to the client.

See below for the recognized allow and deny statements. They can be used to permit or refuse access to known or unknown clients, members of a class, dynamic BOOTP clients, or all clients.

IP Address Conflict Prevention

The DHCP server checks IP addresses to see if they are in use before allocating them to clients. It does this by sending an ICMP Echo request message (ping) to the IP address being allocated. If no ICMP Echo reply is received within a second, the address is assumed to be free. This is only done for leases that have been specified in `range` statements, and only when the lease is thought by the DHCP server to be free – that is, the DHCP server or its failover peer has not listed the lease as in use.

If a response is received to an ICMP Echo request, the DHCP server assumes that there is a configuration error – that the IP address is in use by some host on the network that is not a DHCP client. It marks the address as abandoned and will not assign it to clients.

If a DHCP client tries to get an IP address, but none are available, but there are abandoned IP addresses, then the DHCP server will attempt to reclaim an abandoned IP address. It marks one IP address as free, and then does the same ICMP Echo request check described previously. If there is no answer to the ICMP Echo request, the address is assigned to the client.

The DHCP server does not cycle through abandoned IP addresses if the first IP address it tries to reclaim is free. Rather, when the next DHCPDISCOVER comes in from the client, it will attempt a new allocation using the same method described here and will typically try a new IP address.

Lease Lengths

DHCP leases can be assigned almost any length from zero seconds to infinity. What lease length makes sense for any given subnet, or for any given installation, will vary depending on the kinds of hosts being served.

For example, in an office environment where systems are added from time to time and removed from time to time, but move relatively infrequently, it might make sense to allow lease times of a month or more. In a final test environment on a manufacturing floor, it may make more sense to assign a maximum lease length of 30 minutes – enough time to go through a simple test procedure on a network appliance before packaging it up for delivery.

It is possible to specify two lease lengths: the default length that will be assigned if a client doesn't ask for any particular lease length, and a maximum lease length. These are specified as clauses to the subnet command, for example:

```
subnet 10.252.197.0 netmask 255.255.255.0 {
     range 10.252.197.10 10.252.197.107;
     default-lease-time 600;
     max-lease-time 7200;
}
```

This `subnet` declaration specifies a default lease time of 600 seconds (ten minutes), and a maximum lease time of 7200 seconds (two hours). Other common values would be 86400 (one day), 604800 (one week) and 2592000 (30 days). Note that each subnet need not have the same lease.

Reserved Leases

It's often useful to allocate a single address to a single client, in approximate perpetuity. Host statements with `fixed-address` clauses exist to a certain extent to serve this purpose, but because host statements are intended to approximate a static configuration, they suffer from not being referenced in a litany of other server services, such as dynamic DNS, failover, 'on events' and so forth.

If a standard dynamic lease, as from any `range` statement, is marked reserved, then the server will only allocate this lease to the client it is identified by (by client identifier or hardware address). In practice, this means that the lease follows the normal state engine, enters ACTIVE state when the client is bound to it, expires, or is released, and any events or services that would normally be supplied during these events are processed normally, as with any other dynamic lease. The only difference is that failover servers treat reserved leases as special when they enter the FREE or BACKUP states – each server applies the lease into the state it may allocate from – and the leases are not placed on the queue for allocation to other clients. Instead they may only be found by client identity. The result is that the lease is only offered to the returning client.

> **Note:** Care should be taken to ensure that the client only has one lease within a given subnet that it is identified by.

Leases may be set 'reserved' through the `infinite-is-reserved` configuration option. Leases marked 'reserved' are effectively treated the same as leases marked 'bootp'.

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 the *Conditional Behavior* section) 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 Windows 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.

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 pick-first-value (option dhcp-client-identifier, hardware);
}
class "allocation-class-2" {
  match pick-first-value (option dhcp-client-identifier, 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 is 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:tcpware:alphapc";
```

```
  filename "tcpware:netbsd.alphapc-diskless";
}
```

In these examples, subclassing is being used 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, you can use the value of the `vendor-class-identifier` option to determine what values to send in the `vendor-encapsulated-options` option. See the *Vendor Encapsulated Options* section.

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

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.

Spawning Classes

It is possible to declare a spawning class. A spawning class is a class that automatically produces subclasses based on what the client sends. The reason that spawning classes were created was to make it possible to create lease-limited classes on the fly. For example, 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.

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 example is given only because it is a straightforward one.

Combining Match, Match-If, and Spawn

In some cases, it may be useful to use one expression to assign a client to a particular class, and a second expression to put it into a subclass of that class. This can be done by combining the `match if` and `spawn with` statements, or the `match if` and `match` statements. For example:

```
class "jr-cable-modems" {
  match if option dhcp-vendor-identifier = "jrcm";
  spawn with option agent.circuit-id;
  lease limit 4;
}
class "dv-dsl-modems" {
  match if option dhcp-vendor-identifier = "dvdsl";
  spawn with option agent.circuit-id;
  lease limit 16;
}
```

This allows you to have two classes that both have the same `spawn with` expression without getting the clients in the two classes confused with each other.

Events

There are three kinds of events that can happen regarding a lease, and it is possible to declare statements that occur when any of these events happen. These events are:

- the `commit` event, when the server has made a commitment of a certain lease to a client
- the `release` event when the client has released the server from its commitment
- the `expiry` event when the commitment expires

To declare a set of statements to execute when an event happens, you must use the `on` statement, followed by the name of the event, followed by a series of statements to execute when the event happens, enclosed in braces.

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 the *Expressions* section 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.

Dynamic DNS Updates (DDNS)

The DHCP server can perform dynamic updates to DNS using DNS's dynamic updating functionality (DDNS). Within the configuration files, you can define how you want the updates to be done.

> **Note:** Be sure to configure your name server to allow updates from the DHCP server, see *Chapter 3, Domain Name Services*.

DDNS in DHCP

This section describes how DDNS is implemented in the DHCP server.

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

- `ddns-updates` *`flag`*`;`

- `ddns-update-style { interim | none };`
- `do-forward-updates flag;`
- `ddns-hostname name;`
- `ddns-domainname name;`
- `ddns-rev-domainname name;`
- `update-static-leases flag;`
- `allow/deny/ignore client-updates;`
- `update-conflict-detection flag;`
- `update-optimization flag;`

The DNS update scheme implemented by DHCP is called the *interim* DHCP-DNS interaction draft update mode. In future versions of ISC DHCP, an update mode will be implemented based on the standardized RFCs which came from these drafts.

Use the `ddns-update-style interim` statement to enable the interim update mode. To turn on DDNS updates, use the statement `ddns-updates on`. The `ddns-updates` statement can be at the top of the `dhcpd.conf` file or inside a shared-network or subnet declaration or other scope.

DHCP Interim Update Mode

Updating A and PTR Records

With the interim update mode, the DHCP server does not necessarily always update both the A and the PTR records. By default, forward (A record) updates are enabled. They can be disabled by setting the `do-forward-updates` parameter to `off` or `false`. In this case, the DHCP server never attempts to update the client's A record, and only attempts to update the PTR record if the client supplies an FQDN that should be placed in the PTR record using the FQDN option. If forward updates are enabled, then the DHCP server honors the `client-updates` flag.

The FQDN option sent by the client may include a flag which indicates that the client wishes to update its own A record. In that case, the server can be configured either to honor the client's intentions or ignore them. This is done with the statement `allow/deny/ignore client-updates`. By default, client updates are allowed.

If the server is configured to allow client updates, then if the client sends a fully-qualified domain name in the FQDN option, the server will use that name to update the PTR record. For example, let us say that the client is a visitor from the "radish.org" domain, whose hostname is "jdoe". The server is for the example.org domain. The DHCP client indicates in the FQDN option that its FQDN is "jdoe.radish.org.". It also indicates that it wants to update its own A record. The DHCP server therefore does not attempt to set up an A record for the client, but does set up a PTR record for the IP address that it assigns the client, pointing at jdoe.radish.org. Once the DHCP client has an IP address, it can update its own A record, if the "radish.org" DNS server will allow it to do so.

If the server is configured not to allow client updates or if the client doesn't want to do its own update, the server performs the update of the A record.

The server must determine a host name for the client. It first looks for a `ddns-hostname` configuration option and uses that if it is present. If no such option is present, the server looks for a valid hostname in the FQDN option sent by the client. If one is found, it is used. Otherwise, if the client sent a host-name option, that is used. Otherwise, if there is a `host` declaration that applies to the client, the name from that declaration is used (the name can be specified via the host-name option, or by enabling the `use-host-decl-names` parameter). If none of these applies, the server will not have a hostname for the client and will not be able to do a DNS update.

Note that in the configuration file, `ddns-hostname` or the host-name option can be defined such that the server will generate a host name. See the *Expressions* section for more information.

The server then chooses a domain name for the client. By default, the server uses its own domain name. If desired, the domain name may be specified in the configuration file by using the `option domain-name` statement, or the `ddns-domainname` parameter. For the PTR record, the domain name is by default `in-addr.arpa`. If desired, this domain name may be specified in the configuration file by using the `ddns-rev-domainname` parameter.

The domain name is appended to the host name that it chose for the client. The server then updates both the A and PTR record.

After doing the DDNS updates, if the `ignore client-updates` directive is used, then the server sends a response in the DHCP packet, using the FQDN option, that implies to the client that it should perform its own updates if it chooses to do so. With `deny client-updates`, a response is sent which indicates the client may not perform updates.

Conflict Detection

With the interim scheme, a method is used that allows more than one DHCP server to update the DNS database without accidentally deleting A records that shouldn't be deleted nor failing to add A records that should be added. The scheme works as follows:

When the DHCP server issues a client a new lease, it creates a text string that is an MD5 hash over the DHCP client's identification. The update adds an A record with the name the server chose and a TXT record containing the hashed identifier string. If this update succeeds, the server is done.

If the update fails because the A record already exists, then the DHCP server attempts to add the A record with the prerequisite that there must be a TXT record in the same name as the new A record, and that TXT record's contents must be equal to the hashed identifier. If this update succeeds, then the client has its A record and PTR record. If it fails, then the name the client has been assigned (or requested) is

in use and can't be used by the client. At this point the DHCP server gives up trying to do a DNS update for the client until the client chooses a new name.

This conflict detection can be disabled by setting the `update-conflict-detection` parameter to `off` or `false` in the configuration file. In this case, the server skips the TXT file check and instead simply tears down any previous binding to install the new binding without question.

Update Optimization

Because each DNS update involves a round trip to the DNS server, there is a cost associated with doing updates even if they do not actually modify the DNS database. As a result, the DHCP server tracks whether or not it has updated the record in the past (this information is stored on the lease) and does not attempt to update records that it thinks it has already updated.

This optimization can be disabled by setting the `update-optimization` parameter to `off` or `false` in the configuration file. If the update-optimization parameter is false for a given client, the server will attempt a DNS update for that client each time the client renews its lease. This will allow the DNS to heal from database inconsistencies more easily, but the cost is that the DHCP server must do many more DNS updates. We recommend leaving this optimization enabled, which is the default. If this parameter is not specified, or is set to `true`, the DHCP server will only update when the client information changes, the client gets a different lease, or the client's lease expires.

Static Leases

By default, the server does not do DDNS updates for static assignments – that is, if the IP address is specified in a `fixed-address` statement in a `host` declaration. The `update-static-leases` flag, if enabled, causes the DHCP server to do DNS updates for a client even if it is being given a static assignment. It is not recommended because the DHCP server has no way to tell that the update has been done, and therefore will not delete the record when it is not in use. Also, the server must attempt the update each time the client renews its lease, which could have a significant performance impact in environments that place heavy demands on the DHCP server.

DNSSEC

This section describes how DDNS security (DNSSEC) is implemented in the DHCP server.

The following statements in the DHCP V4 server configuration file are related to DNSSEC:

- `key`
- `zone`

The use of these statements is described here. They are also listed in the *DHCP Statements* section.

When you set your name server up to allow updates from the DHCP server, you may be exposing it to unauthorized updates. To avoid this, you should use TSIG signatures – a method of cryptographically signing updates using a shared secret key. If you protect the secrecy of this key, your updates should also be secure. Note, however, that the DHCP protocol itself provides no security, and that clients can therefore provide information to the DHCP server which the DHCP server will then use in its updates, with the constraints described previously.

Name Server Configuration

The name server must be configured to allow updates for any zone that the DHCP server will be updating. For example, let us say that clients in the example.org domain are assigned addresses on the 10.10.17.0/24 subnet. In that case, you need a `key` declaration for the TSIG key you will be using, and also two `zone` declarations – one for the zone containing A records that are updated and one for the zone containing PTR records. For the TCPware DNS server, you can use something like the following. Note that you may also wish to enable logging of DNS updates in your name server (not shown here).

```
key DHCP_UPDATER {
  algorithm HMAC-MD5.SIG-ALG.REG.INT;
  secret pRP5FapFoJ95JEL06sv4PQ==;
};
zone "example.org" {
     type master;
     file "example_org.hosts";
     allow-update { key DHCP_UPDATER; };
};
zone "17.10.10.in-addr.arpa" {
     type master;
     file "17_10_10_inaddr_arpa.hosts";
     allow-update { key DHCP_UPDATER; };
};
```

DHCP Server Configuration

You also must configure your DHCP server to do updates to these zones. To do so, you need to add something like the following to your `dhcpd.conf` file:

```
key DHCP_UPDATER {
  algorithm HMAC-MD5.SIG-ALG.REG.INT;
  secret pRP5FapFoJ95JEL06sv4PQ==;
};
zone EXAMPLE.ORG. {
  primary 127.0.0.1;
```

```
    key DHCP_UPDATER;
  }
  zone 17.127.10.in-addr.arpa. {
    primary 127.0.0.1;
    key DHCP_UPDATER;
  }
```

The `primary` statement specifies the IP address of the name server whose zone information is to be updated. In addition to the `primary` statement, you may also specify `secondary` statements. The secondaries provide for additional addresses for name servers to be used if the primary does not respond. The number of name servers the DDNS code will attempt to use before giving up is limited and is currently set to three.

Note that the `zone` declarations must correspond to authority records in your name server – in the above example, there must be SOA records for `example.org.` and for `17.10.10.in-addr.arpa.`. For example, if there were a subdomain foo.example.org with no separate SOA, you could not write a zone declaration for foo.example.org. Also keep in mind that zone names in your DHCP configuration should end in a dot (`.`); this is the preferred syntax but not required. (If you do not end your zone name in a dot, the DHCP server will figure it out.) Also note that in the DHCP configuration, zone names are not encapsulated in quotes the way they are in the name server configuration.

Using `dnssec-keygen`

TCPware comes with a program for generating secret keys called `dnssec-keygen`. To use `dnssec-keygen`:

```
$ keygen :== $tcpware:dnssec-keygen.exe
$ keygen -a HMAC-MD5 -b 128 -n USER DHCP_UPDATER
You must use the keyboard to create entropy, since your system is lacking
/dev/random (or equivalent)

start typing:
.......................
```

At this point, start typing at the keyboard to generate a random input. `dnssec-keygen` will display dots until it is done, at which point it will display the following:

```
stop typing.
Kdhcp_updater-157-27654
```

The last line is the name of the files that it created in your current directory which contain the generated key:

```
$ type KDHCP_UPDATER-157-27654.*
DISK:[DEV]KDHCP_UPDATER-157-27654.KEY;1
dhcp_updater. IN KEY 0 3 157 8k+pa57JEGylQUXkOv33VA==
DISK:[DEV]KDHCP_UPDATER-157-27654.PRIVATE;1
```

```
Private-key-format: v1.3
Algorithm: 157 (HMAC_MD5)
Key: 8k+pa57JEGylQUXkOv33VA==
Bits: AAA=
Created: 20140220172314
Publish: 20140220172314
Activate: 20140220172314
```

Take the key in the `Key:` line and use it in your `named.conf` and `dhcpd.conf` files as the value for the `secret` in the `key` statement.

Upgrading DDNS from DHCP V3 to DHCP V4

If you have a legacy DHCP V3 server that is configured to do Dynamic DNS updates and you are upgrading to DHCP V4, the mapping from DHCP V3 DDNS statements to DHCP V4 DDNS statements is as follows.

Note that in DHCP V3 the default is to not do DDNS updates, while in DHCP V4 the default is to do them. Similarly, the DHCP V3 default is to not create the A record, while the DHCP V4 default is to create it.

DHCP V3	**DHCP V4**
`allow/deny dynamic-update;`	`ddns-updates` *`flag`*`;` `ddns-update-style {interim|none};`
`allow/deny update-A-record;`	`do-forward-updates` *`flag`*`;` `allow/deny/ignore client-updates;`

`allow/deny name-by-client;`
`invalid-ddns-chars {fail |`
`discard | replace [“`*`chars`*`”]};`
	see above for how host names are determined

Host Name Generation

Some DHCP clients require that the server send them a host name. The DHCP server can generate a host name if it cannot get the host name in another way. 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 the *Dynamic DNS Updates (DDNS)* section.

To enable the DHCP server to generate host names, specify in the configuration file an `option host-name` statement with a value formatted in a certain way, as described here. The `option host-name` statement can be specified for example at the top level in a `subnet` statement, or in a `host` statement.

You can specify the host name to generate either in the `option host-name` statement or the `ddns-hostname` statement. You can use a combination of evaluation functions to tell the server how to generate the name. See the *Expressions* section for more information on available evaluation functions.

See the *DHCP Interim Update Mode* section for a description of when it looks at `ddns-hostname` statement and when it looks at the `option host-name` statement to determine the host name. If it does execute either statement, it will generate a host name as specified by the expressions in the value.

For example:

```
option host-name = concat("DHCP-", binary-to-ascii(10,8,"",leased-address);
ddns-hostname = concat("DHCP-", binary-to-ascii(10,8,"",leased-address);
```

These statements in `dhcpd.conf` generate a host name consisting of the string `DHCP-` followed by the ASCII version of the IP address that was leased out to the client by the DHCP server. It uses the data expressions `concat`, `binary-to-ascii`, and `leased-address`.

You can use the following keys to specify what you want the generated host name to look like. The generated host name can contain parts of the host's IP address, client ID, and/or MAC address, plus any characters that are valid for the `host-name` option. 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.
`%B`	Second byte of the host's IP address.
`%C`	Third byte of the host's IP address.

%D	Fourth byte of the host's IP address.
%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.
%-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 192.168.11.7 through 192.168.11.10 and a subnet mask of 255.255.255.0, the DHCP server generates the following host names:

```
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.

Configuration File Declarations and Parameters

This section lists and describes the declarations and parameters you can use in a configuration file.

This section does not include DHCP Failover related configuration file statements. (See the *Failover Configuration File Statements* section).

Allow and Deny Statements

Use the `allow` and `deny` and `ignore` 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 `allow` and `deny` to set up access lists for address allocation pools.
- In a non-pool context, the `ignore` keyword can be used in place of the `deny` keyword to prevent logging of denied requests.
- In other contexts, use these keywords to control general server behavior with respect to clients based on scope.

Allow and Deny Outside of Pool Declarations

These allow, deny, and ignore 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.

These are not recommended for use inside `pool` declarations. See the *Pool Permit Lists* section for an important note.

The following table lists the available `allow` and `deny` statements.

Statement	Description

`allow unknown-clients;`
`deny unknown-clients;`
`ignore unknown-clients;`
	Use the `unknown-clients` flag to tell the DHCP server whether to dynamically 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.
	
	The use of this option outside of pool declarations is now deprecated. If you are trying to restrict access on your network to known clients, you should deny unknown clients inside of your address pool.

`allow bootp;`
`deny bootp;`
`ignore bootp;`
	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 booting;`
`deny booting;`
`ignore booting;`
	Use the `booting` flag to tell the DHCP server whether to respond to queries from a particular client. This keyword only has meaning inside of a `host` declaration. 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 declines;`
`deny declines;`
`ignore declines;`
	The DHCPDECLINE message is used by DHCP clients to indicate that the lease the server has offered is not valid. When the server receives a DHCPDECLINE for a particular address, it normally abandons that address, assuming that some unauthorized system is using it. Unfortunately, a malicious or buggy client can, using DHCPDECLINE messages, completely exhaust the DHCP server's allocation pool. The server will reclaim these leases, but while the client is running through the pool, it may cause serious thrashing in DNS, and it will also cause the DHCP server to forget old DHCP client address allocations.

	The `declines` flag tells the DHCP server whether or not to honor DHCPDECLINE messages. If it is set to `deny` or `ignore` in a particular scope, the DHCP server will not respond to DHCPDECLINE messages.
`allow duplicates;` `deny duplicates;`	Host declarations can match client messages based on the DHCP Client Identifier option or based on the client's network hardware type and MAC address. If the MAC address is used, the host declaration will match any client with that MAC address – even clients with different client identifiers. This doesn't normally happen, but is possible when one computer has more than one operating system installed on it – for example, Microsoft Windows and Linux. The `duplicates` flag tells the DHCP server that if a request is received from a client that matches the MAC address of a `host` declaration, any other leases matching that MAC address should be discarded by the server, even if the UID is not the same. This is a violation of the DHCP protocol, but can prevent clients whose client identifiers change regularly from holding many leases at the same time. By default, duplicates are allowed.
`allow leasequery;` `deny leasequery;`	The `leasequery` flag tells the DHCP server whether or not to answer DHCPLEASEQUERY packets. The answer to a DHCPLEASEQUERY packet includes information about a specific lease, such as when it was issued and when it will expire. By default, the server will not respond to these packets.
`allow dhcpinform;` `deny dhcpinform;`	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 client-updates;` `deny client-updates;`	The `client-updates` flag tells the DHCP server whether or not to honor the client's intention to do its own update of its A record. This is only relevant when doing interim DNS updates.

Allow and Deny in Pool Declarations

This section lists and describes the available allow and deny statements that can be used in pool declarations.

See the *Pool Permit Lists* section for discussion, defaults, and important notes.

Statement	**Description**
`allow unknown-clients;` `deny unknown-clients;`	Use `unknown clients` to allow or prevent allocation from this pool to any client that has no host declaration.
`allow members of "`*`class-name`*`";` `deny members of "`*`class-name`*`";`	Use `members of "`*`class`*`"` to allow or prevent allocation from this pool to any client that is a member of the named class.
`allow dynamic bootp clients;` `deny dynamic bootp clients;`	Use `dynamic bootp clients` to allow or prevent allocation from this pool to any BOOTP client.
`allow all clients;` `deny all 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 after `*`time`*`;` `deny after `*`time`*`;`	If specified, this statement either allows or prevents allocation from this pool after a given date. This can be used when you want to move clients from one pool to another. The server adjusts the regular lease time so that the latest expiry time is at the given time plus *`min-lease-time`*. A short *`min-lease-time`* enforces a

	step change, whereas a longer *min-lease-time* allows for a gradual change. *time* is either seconds since epoch, or a UTC time string e.g. 2007/08/24 09:14:32 or a string with time zone offset in seconds e.g. 2007/08/24 11:14:32 -7200

DHCP Statements

This section lists and describes the remaining declarations and parameters that can be specified in the DHCP server configuration file.

Statement	**Description**
adaptive-lease-time-threshold *percentage*;	When the number of allocated leases within a pool rises above the percentage given in this parameter, the DHCP server decreases the lease length for new clients within this pool to *min-lease-time* seconds. Clients renewing already valid (long) leases get at least the remaining time from the current lease. Since the leases expire faster, the server may either recover more quickly or avoid pool exhaustion entirely. Once the number of allocated leases drop below the threshold, the server reverts back to normal lease times. Valid percentages are between 1 and 99.
add "*class-name*";	Use the add statement to add a client to the class whose name is specified in *class-name*. Because this statement executes after IP address allocation is completed, class membership caused by this statement cannot be used in the address allocation process.
always-broadcast *flag*;	Use the always-broadcast parameter 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.
`always-reply-rfc1048` *`flag`*`;`	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` parameter 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.
`authoritative;` `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).
`boot-unknown-clients` *`flag;`*	If the `boot-unknown-clients` parameter is present and has a value of false or off, then clients for which there is no `host` declaration will not be allowed to obtain IP addresses. If this statement is not present or has a value of true or on, then clients without host declarations will be allowed to obtain IP addresses, as long as those addresses are not restricted by allow and deny statements within their pool declarations.
`class "`*`class-name`*`" {[`*`statements`*`][`*`declarations`*`]}`	This declaration 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`.
`ddns-domainname` *`name`*`;`	Use the `ddns-domainname` parameter to specify the domain name to be appended to the client's host name to form a fully qualified domain name (FQDN) for DDNS. See *Dynamic DNS Updates (DDNS).*
`ddns-hostname` *`name`*`;`	Use the `ddns-hostname` parameter to specify the hostname to be used in setting up the client's A and PTR records for DDNS. If no `ddns-hostname` is specified in scope, then the server derives the hostname automatically. See *Dynamic DNS Updates (DDNS).*
`ddns-rev-domainname` *`name`*`;`	Use the `ddns-rev-domainname` parameter to specify the domain name that will be appended to the client's reversed IP address to produce a name for use in the client's PTR record. By default, this is "in-addr.arpa.", but the default can be overridden here. The reversed IP address to which this domain name is appended is always the IP address of the client, in dotted quad notation, reversed – for example, if the IP address assigned to the client is 10.17.92.74, then the reversed IP address is 74.92.17.10. So a client with that IP address would, by default, be given a PTR record of 10.17.92.74.in-addr.arpa.
`ddns-update-style [interim | none];`	Use the `ddns-update-style` parameter to turn on DDNS updates and specify the update style. The only supported update style is `interim`. To turn off DDNS updates, set `ddns-update-style` to `none` (the default). This parameter is only meaningful in the outer scope – at the top of the `DHCPD.CONF` file. There is no way to set `ddns-update-style` to different values for different clients.

`ddns-updates` *`flag`*`;`	The `ddns-updates` parameter controls whether or not the server will attempt to do a DNS update when a lease is confirmed. Set this to `off` if the server should not attempt to do updates within a certain scope. The `ddns-updates` parameter is `on` by default. To disable DNS updates in all scopes, it is preferable to use the `ddns-update-style` statement, setting the style to `none`.
`db-time-format [default | local];`	The DHCP server software outputs several timestamps when writing leases to persistent storage. The `db-time-format` parameter selects one of two output formats: default or local. The `default` format prints the day, date, and time in UTC, while the `local` format prints the system seconds-since-epoch, and provides the day and time in the system time zone in a comment
`default-lease-time` *`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 43200 seconds (12 hours). You should set the value of `default-lease-time` no larger than the value of `max-lease-time`.
`delayed-ack` *`count`*`;`	The `delayed-ack` parameter works with the `max-ack-delay` parameter. Use the `delayed-ack` parameter to specify an integer value from 0 to 65535 (default 28) representing the maximum number of replies that the server can queue up pending transmission until after a database commit event. If this number is reached, a database commit is done (representing a performance penalty), and the replies are transmitted in a batch. This preserves the RFC2131 direction that "stable storage" be updated prior to replying to clients. If it happens at any point that there are no incoming requests into the DHCP server (and so the

	server is temporarily idle), the commit is made at that time and any queued replies are transmitted.
`do-forward-updates` *`flag;`*	The `do-forward-updates` parameter instructs the DHCP server as to whether it should attempt to update a DHCP client's A record when the client acquires or renews a lease. This statement has no effect unless DNS updates are enabled and `ddns-update-style` is set to `interim`. Forward updates are enabled by default. If this statement is used to disable forward updates, the DHCP server will never attempt to update the client's A record, and will only ever attempt to update the client's PTR record if the client supplies an FQDN (fully qualified domain name) that should be placed in the PTR record, using the `fqdn` option. If forward updates are enabled, the DHCP server will still honor the setting of the `client-updates` flag.
`dynamic-bootp-lease-cutoff` *`date;`*	Use the `dynamic-bootp-lease-cutoff` parameter 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`* *`W`* is the day of the week, from 0 (Sunday) to 6 (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.
`dynamic-bootp-lease-length` *length;*	Use the `dynamic-bootp-lease-length` parameter 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 0. 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.
`filename "`*file-name*`";`	Use the `filename` parameter to specify the name of the initial boot file that is to be loaded by a client. The *file-name* should be recognizable to whatever file transfer protocol the client can be expected to use.
`fixed-address` *address* `[,…,`*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` statement. A `fixed-address` statement 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` statement. 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.
`get-lease-hostnames` *`flag;`*	Use the `get-lease-hostnames` parameter 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.
`group {[`*`parameters`*`] [`*`declarations`*`]}`	Use the `group` declaration to apply one or more parameters to a group of declarations. You can use it to group hosts, shared networks, subnets, or other groups.

`hardware` *`hardware-type hardware-address`*`;`	Use the `hardware` parameter 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 (:).
`host` *`name`* `{[`*`parameters`*`] [`*`declarations`*`]}`	The `host` declaration provides information about a particular client. *`name`* should be a unique name, 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. So, 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", and 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. 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.

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

`include "filename";`	The `include` statement is used to read in the specified file, and process the contents of the included file as if they were in the configuration file directly.
`infinite-is-reserved flag;`	If the `infinite-is-reserved` parameter is set to `on` the server automatically marks as reserved leases which are allocated to clients who requested an infinite lease time. The default is `off`.

`key key-name {`
`  algorithm algorithm;`
`  secret key;`
`};`
	The `key` statement specifies the secret key and algorithm to use for DNSSEC.

`lease-file-name name;`	The `lease-file-name` parameter in the configuration file can be used to specify the name of the lease file. The default is `TCPWARE:DHCPD.LEASES`. This parameter must appear at the outer scope – at the top of the configuration file.
`lease limit 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. 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.
`local-port port;`	The `local-port` parameter can be used to specify a different port for the DHCP server to listen on, other than the default port of 67.

`match` *`data-expression`*`;`	*`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 if` *`boolean-expression`*`;`	*`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.
`max-ack-delay` *`microseconds`*`;`	The `max-ack-delay` parameter works with the `delayed-ack` parameter (see description above). Use the `max-ack-delay` parameter to specify the length of time that replies are allowed to queue up awaiting a database commit event. The value is in microseconds, from 0 to 4,294,967,295, with a default of 250,000 (1/4 of a second).
`max-lease-time` *`time`*`;`	Use the `max-lease-time` parameter to 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). You should set the value of `max-lease-time` at least as large as `default-lease-time`.
`min-lease-time` *`time`*`;`	Use the `min-lease-time` parameter to assign the minimum length in seconds to a lease. The infinite lease value is `infinite`. The default is the smaller of 300 seconds or the value of `max-lease-time`.

	`min-lease-time` should be less than or equal to `default-lease-time` and `max-lease-time`.
`min-secs` *`seconds`*`;`	Use the `min-secs` parameter 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. 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 that this does not, by itself, permit a primary server and a secondary server to share a pool of dynamically-allocatable addresses.
`next-server` *`name`*`;`	Use the `next-server` parameter to specify the host address of the server from where the client will load the initial boot file (specified in the `filename` statement). *`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.
`one-lease-per-client` *`flag`*`;`	Use the `one-lease-per-client` parameter to have the server free any other leases the client holds when the client sends a DHCPREQUEST 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 is guaranteed or provable, so use caution in the use of this statement.
`option`	This statement specifies actual DHCP protocol options to send to the client. The `option` statement is described in the next section.
`option` *`name`* `code` *`code`* `=` *`definition`*`;`	This statement assigns a name and a type to an `option` code. See the *Defining New Options* section for more information.
`option space` *`space-name`* `[` *`options`* `];`	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`. Option space names include: `dhcp` (the default), `agent`, and `server`. 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.
`ping-check` *`flag`*`;`	When the DHCP server is considering dynamically allocating an IP address to a client, it first sends an ICMP Echo request (a ping) to the address being assigned. It waits for a second, and if no ICMP Echo response has been heard, it assigns the address. If a response is heard, the lease is abandoned, and the server does not respond to the client.

	This ping check introduces a default one-second delay in responding to DHCPDISCOVER messages, which can be a problem for some clients. The length of delay may be configured using the `ping-timeout` parameter. A value of `false` or `off` turns pinging off.
`ping-retries` *`count;`*	This parameter 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-timeout` *`time;`*	This parameter defines the time (in seconds) that ping should wait for a response. The default is 1 second.
`pool {[`*`permit list`*`][`*`range declaration`*`][`*`statements`*`]}`	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.
`range [dynamic-bootp]` *`low-address`* `[`*`high-address`*`];`	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.
remote-port *port;*	The remote-port parameter can be used to specify a different port for the DHCP server to send to clients at, other than the default port of 68. Changing the remote port is not recommended.

requested-options-only *flag;*	Use the requested-options-only parameter 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.

`server-identifier` *`hostname`*`;`	The `server-identifier` parameter is equivalent to the `dhcp-server-identifier` option. See the `dhcp-server-identifier` option for more information.
`server-name` *`name`*`;`	Use the `server-name` parameter to inform the client of the server's name from which it is booting. *`name`* should be the name provided to the client.
`shared-network {[`*`parameters`*`] [`*`declarations`*`]};`	Use this declaration 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` declaration. *`parameters`* specified in the shared-network declaration 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. It can have the syntax of a valid domain name (although it will never be used as such), or can be any arbitrary name enclosed in quotation marks.

`site-option-space` *`option-space`*`;`	The `site-option-space` parameter can be used to determine from what option space site-local options will be taken. This can be used in much the same way as the `vendor-option-space` parameter. Site-local options in DHCP are those options whose numeric codes are greater than 224. These options are intended for site-specific uses, but are frequently used by vendors of embedded hardware that contains DHCP clients. Because site-specific options are allocated on an ad-hoc basis, it is quite possible that one vendor's DHCP client might use the same option code that another vendor's client uses, for different purposes. The `site-option-space` parameter can be used to assign a different set of site-specific options for each such vendor, using conditional evaluation.
`spawn with data-expression;`	*`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.
`stash-agent-options` *`flag`*`;`	If the `stash-agent-options` parameter is true for a given client, the server will record the relay agent information options sent during the client's initial DHCPREQUEST message when the client was in the SELECTING state and behave as if those options are included in all subsequent DHCPREQUEST messages sent in the RENEWING state. This works around a problem

	with relay agent information options, which is that they usually not appear in DHCPREQUEST messages sent by the client in the RENEWING state, because such messages are unicast directly to the server and not sent through a relay agent.

`subclass "class-name" class-data;`
`subclass "class-name" class-data {`
`  [statements]`
`}`
	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.

`subnet` *`subnet-number`* `netmask` *`netmask`* `{[`*`parameters`*`][`*`declarations`*`]}`	This declaration 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 are an IP address or domain name that resolves to the *subnet-number* or the *netmask* of the subnet being described. The *subnet-number* and the *netmask* are enough to determine if any given IP address is on the specified subnet. 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.
update-conflict-detection *flag*;	DDNS: If the update-conflict-detection parameter is true, the server will perform standard DHCID multiple-client, one-name conflict detection. If the parameter has been set to false, the server will skip this check and instead simply tear down any previous bindings to install the new binding without question. The default is true.
update-optimization *flag*;	DDNS: If the update-optimization parameter is false for a given client, the server will attempt a DNS update for that client each time the client renews its lease, rather than only attempting an update when it appears to be necessary. This will allow the DNS to heal from database inconsistencies more easily, but the cost is that the DHCP server must do many more DNS updates. We recommend leaving this option enabled, which is the default. If this parameter is not specified, or is true, the DHCP server will only update when the client information changes, the client gets a different lease, or the client's lease expires.
update-static-leases *flag*;	DDNS: The update-static-leases parameter, if enabled, causes the DHCP server to do DNS updates for clients even if those clients are being assigned their IP address using a fixed-address statement – that is, the client

	is being given a static assignment. It is not recommended because the DHCP server has no way to tell that the update has been done, and therefore will not delete the record when it is not in use. Also, the server must attempt the update each time the client renews its lease, which could have a significant performance impact in environments that place heavy demands on the DHCP server.

`use-host-decl-names` *`flag`*`;`	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.example.com;`
	` }`
	`}`
	
	is equivalent to
	
	`host joe {`
	` hardware ethernet 08:00:2b:4c:29:32;`
	` fixed-address joe.example.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-lease-addr-for-default-route` *`flag`*`;`	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. The use of this feature is not recommended. 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.
`vendor-option-space` *`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.

`zone` *`zone-name`* `{`
`  primary` *`ip-address`*`;`
`  [ secondary` *`ip-address`*`; ]`
`  [ key` *`key-name`*`; ]`
`};`
	DDNS: The `zone` statement provides information to the server about the DNS zones that it might be updating via DDNS. This statement is required if you are using DNSSEC (using DNSSEC is recommended).
	
	The *`zone-name`* must correspond to the name of the actual zone to be updated, for example “example.org.” or “17.10.10.in-addr.arpa.” (without the quotes). Note that the zone specified must have an SOA record.
	
	The `primary` and `secondary` keywords specify the IP addresses of the primary and secondary name servers for the zone. Up to three name servers may be specified.
	
	The `key` keyword is used to provide the secret key to be used by DNSSEC. The *`key-name`* is the name of a `key` statement in the configuration file which contains that information.

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.

Expressions can be used to set the value of a DHCP server parameter or an option, for example based on some value that the client has sent. To do this, you can use expression evaluation. To assign the result of an evaluation to a parameter or option, use the following syntax in your configuration file:

```
parameter-or-option = expression;
```

The following example asks the DHCP server to create a host name for the client:

```
option host-name = concat("DHCP-", binary-to-ascii(10,8,"",leased-address);
ddns-hostname = concat("DHCP-", binary-to-ascii(10,8,"",leased-address);
```

These statements in `dhcpd.conf` generate a host name consisting of the string "`DHCP-`" followed by the ASCII version of the IP address that was leased out to the client by the DHCP server. It uses the data expressions `concat`, `binary-to-ascii`, and `leased-address`, described below.

Boolean Expressions

The following are the boolean expressions supported by DHCP.

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.
data-expression-1 = *data-expression-2*	The equals (=) 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.
data-expression-1 != *data-expression-2*	The not-equals (!=) operator compares the results of evaluating two data expressions, evaluating to true if they are not the same; evaluating to false if they are. 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.
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.
static	The `static` expression evaluates to true if the lease assigned to the client whose request is being processed is derived from a static address assignment.

Data Expressions

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

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 hexadecimal 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-exprN`*`)`	The data expressions *`data-expr1`* through *`data-exprN`* are evaluated and the results of each evaluation are concatenated in the sequence that the subexpressions are listed. If any subexpression evaluates to null, the result of the concatenation is null.
`config-option` *`option-name`*	The `config-option` operator returns the value for the specified option that the server has been configured to send.

`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.
`gethostname`	The *`gethostname`* function returns a data string whose contents are a character string, the results of calling `gethostname()` on the local system with a size limit of 255 bytes (not including NULL terminator).
`hardware`	The `hardware` operator returns a data string whose first element is the type of the network interface indicated in the packet being processed, and whose subsequent elements are the client's link-layer address. 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)
`host-decl-name`	The `host-decl-name` function returns the name of the `host` declaration that matched the client whose request is being processed, if any. If no host declaration matched, the result is the null value.
`pick-first-value (data-expr1 [,… data-exprN]`	The `pick-first-value` function takes any number of data expressions as its arguments. Each expression is evaluated, starting with the first in the list, until an expression is found that does not evaluate to a null value. That expression is returned, and none of the subsequent expressions are evaluated. If all expressions evaluate to a null value, the null value is returned.
`lcase (data-expr)`	The `lcase` function returns the result of evaluating *`data-expr`* converted to lower case. If *`data-expr`* evaluates to null, then the result is also null.

`leased-address`	In any context where the client 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 packet to which the server is responding.
`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 4 and *`data-expr2`* evaluates to 12 bytes of data, the `reverse` expression evaluates to 12 bytes of data constructed in the following way: 1. the last 4 bytes of the input data, 2. followed by the middle 4 bytes, 3. followed by the first 4 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`* evaluates 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* evaluates 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. The backslash (\) character is treated specially, as in C programming: • \t means TAB, • \r means carriage return • \n means newline • \b means bell • any octal value can be specified with *nnn*, where *nnn* is any positive octal number less than 0400 • any hexadecimal value can be specified with \x*nn*, where *nn* is any positive hexadecimal number less than or equal to 0xff
ucase (*data-expr*)	The ucase function returns the result of evaluating *data-expr* converted to upper case. If *data-expr* evaluates to null, then the result is also null.

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 (8, 16, or 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.

Action Expressions

`log ([`*`priority`*`,] `*`data-expr`*`)`	The `log` expression may be used to write a message (specified via *`data-expr`*) to the DHCP server debug log file and/or OPCOM (depending on how the server was configured via `TCPWARE:CNFNET`). The optional *`priority`* can be: `fatal`, `error`, `info`, or `debug`.

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 below.

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 other predefined option spaces, for example, `agent` and `server`. You can also define option spaces of your own. See the sections *Relay Agent Information Option* and *Defining New Options* in this chapter.

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 `domain-name` data type specifies a domain name, which must not be enclosed in double quotes. This data type is not used for any existing DHCP options. The domain name is stored just as if it were a text option.
- The `domain-list` data type specifies a list of domain names, enclosed in double quotes, and separated by commas. For example, `"example.com","foo.example.com"`.
- 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 the *Expressions* section). The syntax is

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

Standard DHCP Options

This section describes the standard DHCP options. Italicized items indicate user input items.

> **Note:** All options can be specified with the `dhcp` option space listed explicitly. For example:

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

Option	Description
`option all-subnets-local` *`flag`*`;`	Use this option to indicate whether 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 bcms-controller-address` *`ip-address`* `[,` *`ip-address`* `…];`	This option configures a list of IPv4 addresses for use as Broadcast and Multicast Controller Servers ("BCMS").
`option bcms-controller-names` *`domain-list`*`;`	This option contains the domain names of local Broadcast and Multicast Controller Servers ("BCMS") controllers which the client may use.
`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), section 3.2.1.3 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 default-url` *`data-string`*`;`	The format and meaning of this option is not described in any standards document, but is claimed to be in use by legacy Apple products. It is not known what clients may reasonably do if supplied with this option. Use at your own risk.
`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. Use this option with caution. Make sure that the client can accept a message of the specified size.
`option dhcp-parameter-request-list` *`uint8`*`[,`*`uint8`*`...];`	The client uses 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 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 domain-search` *`domain-list`*`;`	The `domain-search` option specifies a 'search list' of Domain Names to be used by the client to locate not-fully-qualified domain names. The difference between this option and historic use of the domain-name option for the same ends is that this option is encoded in RFC1035 compressed labels on the wire.
`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 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 or not 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 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` *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. 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. Possible node types are: `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` *data-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 netinfo-server-address` *ip-address* [, *ip-address* …];	The `netinfo-server-address` option has not been described in any RFC, but has been allocated (and is claimed to be in use) by legacy Apple products. It's hard to say if this is the correct format, or what clients might be expected to do if values were configured. Use at your own risk.
`option netinfo-server-tag` *string*;	The `netinfo-server-tag` option has not been described in any RFC, but has been allocated (and is claimed to be in use) by legacy Apple products. It's hard to say if this is the correct format, or what clients might be expected to do if values were configured. Use at your own risk.
`option nis-domain` *string*;	Use this option to specify the client's NIS (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 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 1035) servers in order of preference.
`option nwip-domain` *`data-string`*`;`	The name of the NetWare/IP domain that a NetWare/IP client should use.
`option nwip-suboptions` *`data-string`*`;`	A sequence of suboptions for NetWare/IP clients – see RFC2242 for details. Normally this option is set by specifying specific NetWare/IP suboptions – see the *NetWare/IP Suboptions* The NetWare/IP Sub-optionssection for more information.
`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 RFC 1122 for more information.
`option pop-server` *`ip-address`* `[,` *`ip-address`* `...];`	Use this option to list POP3 servers in order of preference.
`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 slp-directory-agent` *`flag`* *`ip-address`* `[,` *`ip-address`* `…];`	This option specifies two things: the IP addresses of one or more Service Location Protocol Directory Agents, and whether the use of these addresses is mandatory. If the flag is `true`, the SLP agent should just use the IP addresses given. If the value is `false`, the SLP agent may additionally do active or passive multicast discovery of SLP agents (see RFC2165 for details).

	Please note that in this option and the `slp-service-scope` option, the term "SLP Agent" is being used to refer to a Service Location Protocol agent running on a machine that is being configured using the DHCP protocol. Note that some companies may refer to SLP as NDS. If you have an NDS directory agent whose address you need to configure, the `slp-directory-agent` option should work.
`option slp-service-scope` *`flag`* `[` *`string`* `];`	The Service Location Protocol Service Scope Option specifies two things: a list of service scopes for SLP, and whether the use of this list is mandatory. If the flag is `true`, the SLP agent should only use the list of scopes provided in this option; otherwise, it may use its own static configuration in preference to the list provided in this option. The text string should be a comma-separated list of scopes that the SLP agent should use. It may be omitted, in which case the SLP Agent will use the aggregated list of scopes of all directory agents known to the SLP agent.
`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. This option cannot be used with classless IP routing since it does not include a subnet mask.
`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 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 keep-alive 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 keep-alive message on a TCP connection. The time is specified as a 32-bit unsigned integer.

	`0` (zero) means do not generate keep-alive 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 numbers 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 uap-servers` *`string`*`;`	This option specifies a list of URLs, each pointing to a user authentication service that is capable of processing authentication requests encapsulated in the User Authentication Protocol (UAP). UAP servers can accept either HTTP 1.1 or SSLv3 connections. If the list includes a URL that does not contain a port component, the normal default port is assumed (i.e., port 80 for HTTP and port 443 for HTTPS). If the list includes a URL that does not contain a path component, the path `/uap` is assumed. If more than one URL is specified in this list, the URLs are separated by spaces.

`option vendor-encapsulated-options` *`data-string`*`;`	Use this option to specify vendor-specific information. See the *Vendor Encapsulated Options* section.
`option vivso` *`data-string`*`;`	The `vivso` option can contain multiple separate options, one for each 32-bit Enterprise ID. Each Enterprise-ID discriminated option then contains additional options whose format is defined by the vendor who holds that ID. This option is usually not configured manually, but rather is configured via intervening option definitions. See the *Vendor Encapsulated Options* section.
`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.

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 the following sub-options. 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.

`option agent.circuit-id` *`data-string`*`;`	The `circuit-id` sub-option 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` *`data-string;`*	The `remote-id` sub-option 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.
`option agent.DOCSIS-device-class` *`uint32;`*	The DOCSIS-device-class sub-option is intended to convey information about the host endpoint, hardware, and software, that either the host operating system or the DHCP server may not otherwise be aware of (but the relay is able to distinguish). This is implemented as a 32-bit field (4 octets), each bit representing a flag describing the host in one of these ways. So far, only bit zero (being the least significant bit) is defined in RFC3256. If this bit is set to one, the host is considered a CPE Controlled Cable Modem (CCCM). All other bits are reserved.
`option agent.link-selection` *`ip-address;`*	The link-selection sub-option is provided by relay agents to inform servers what subnet the client is actually attached to. This is useful in those cases where the `giaddr` (where responses must be sent to the relay agent) is not on the same subnet as the client. When this option is present in a packet from a relay agent, the DHCP server will use its contents to find a subnet declared in the configuration, and from here take one step further backwards to any shared network the subnet may be defined within. The client may be given any address within that shared network, as normally appropriate.

The Client FQDN Suboptions

The client FQDN option is sent from the client to the server. Then in the response, the server constructs a reply FQDN option. Due to the complexity of the client FQDN option format, it has been implemented as a sub-option space rather than a single option. Its components should not be specified in the server

configuration file. It is constructed by the server itself based on information it has when it formats a reply to send to the client.

The configuration file can reference these options by specifying the option space name "`fqdn`", followed by a period, followed by the option name.

`option fqdn.no-client-update` *`flag`*`;`	When the client sends this, if it is true, it means the client will not attempt to update its A record. When sent by the server to the client, the server is telling the client that the client should not update its own A record.
`option fqdn.server-update` *`flag`*`;`	When the client sends this to the server, it is requesting that the server update its A record. When sent by the server, it is telling the client that the server has updated (or is about to update) the client's A record.
`option fqdn.encoded` *`flag`*`;`	If true, this indicates that the domain name included in the option is encoded in DNS wire format, rather than as plain ASCII text. The client normally sets this to false if it doesn't support DNS wire format in the FQDN option. The server should always send back the same value that the client sent.
`option fqdn.rcode1` *`uint8`*`;` `option fqdn.rcode2` *`uint8`*`;`	These options specify the result of the updates of the A and PTR records, respectively, and are only sent by the DHCP server to the DHCP client. The values of these fields are those defined in the DNS protocol specification.
`option fqdn.fqdn` *`string`*`;`	Specifies the domain name that the client wishes to use. This can be a fully-qualified domain name, or a single label. If there is no trailing dot in the name, it is not fully-qualified, and the server will generally update that name in some locally-defined domain.
`option fqdn.hostname` (never set)	This option should never be set, but it can be read back using the `option` and `config-option` operators in an expression, in which case it returns the first label in the `fqdn.fqdn suboption` - for example, if the value of

	`fqdn.fqdn` is "`foo.example.com.`", then `fqdn.hostname` will be "`foo`".
`option fqdn.domainname` (never set)	This option should never be set, but it can be read back using the `option` and `config-option` operators in an expression, in which case it returns all labels after the first label in the `fqdn.fqdn` sub-option – for example, if the value of `fqdn.fqdn` is "`foo.example.com.`", then `fqdn.domainname` will be "`example.com.`". If this sub-option value is not set, it means that an unqualified name was sent in the `fqdn` option, or that no `fqdn` option was sent at all.

The NetWare/IP Sub-options

RFC2242 defines a set of encapsulated options for Novell NetWare/IP clients. To use these options in the DHCP server, specify the option space name, "`nwip`", followed by a period, followed by the option name. The following options can be specified:

`option nwip.nsq-broadcast` *`flag;`*	If true, the client should use the NetWare Nearest Server Query to locate a NetWare/IP server. The behavior of the Novell client if this sub-option is false, or is not present, is not specified.
`option nwip.preferred-dss` *`ip-address`* `[,`*`ip-address`*`…];`	This sub-option specifies a list of up to five IP addresses, each of which should be the IP address of a NetWare Domain SAP/RIP server (DSS).
`option nwip.nearest-nwip-server` *`ip-address`* `[,`*`ip-address`* `…]`	This sub-option specifies a list of up to five IP addresses, each of which should be the IP address of a Nearest NetWare IP server.
`option nwip.autoretries` *`uint8;`*	Specifies the number of times that a NetWare/IP client should attempt to communicate with a given DSS server at startup.

`option nwip.autoretry-secs` *`uint8;`*	Specifies the number of seconds that a Netware/IP client should wait between retries when attempting to establish communications with a DSS server at startup.
`option nwip.nwip-1-1` *`uint8;`*	If true, the NetWare/IP client should support NetWare/IP version 1.1 compatibility. This is only needed if the client will be contacting Netware/IP version 1.1 servers.
`option nwip.primary-dss` *`ip-address;`*	Specifies the IP address of the Primary Domain SAP/RIP Service server (DSS) for this NetWare/IP domain. The NetWare/IP administration utility uses this value as primary DSS server when configuring a secondary DSS server.

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 cannot 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 TCPware. If an option name doesn't appear here, you can use it, but it's probably a good idea to put some kind of unique string at the beginning so you can be sure that future options don't take your name.

After choosing a name, choose a *code*. For site-local options, all codes between 224 and 254 are reserved for site-local 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.

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 224 = 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 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";

string	option *new-name* code *new-code* = string ; An option of type 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;
domain-list	option *new-name* code *new-code* = domain-list [compressed]; An option whose type is domain-list is an RFC1035-formatted (on the wire, "DNS Format") list of domain names, separated by root labels. The optional compressed keyword indicates if the option should be compressed relative to the start of the option contents (not the packet contents). When in doubt, omit the compressed keyword. When the software receives an option that is compressed and the compressed keyword is omitted, it will still decompress the option (relative to the option contents field). The keyword only controls whether or not transmitted packets are compressed.

encapsulation	option *new-name* code *new-code* = encapsulate *identifier*;
	
	An option whose type is encapsulate will encapsulate the contents of the option space specified in identifier. Examples of encapsulated options in the DHCP protocol include the vendor-encapsulated-options option, the netware-suboptions option and the relay-agent-information option. For example:
	
	option space local;
	option local.demo code 1 = text;
	option local-encapsulation code 225 = encapsulate local;
	option local.demo "demo";

`array`	Options can contain arrays of any of the above types except for the `text` and the `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 231 =`
	`{ 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 };`
	`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;`

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 all of these options is either a chunk of opaque data, or an actual option buffer just like a standard DHCP option buffer.

The DHCP protocol also defines the Vendor Identified Vendor Sub Options option (“VIVSO”).The VIVSO option differs in that it contains options that correspond to vendor Enterprise-ID numbers (assigned by IANA), which then contain options according to each vendor’s specifications. You will need to refer to your vendor’s documentation in order to form options to their specification.

You can send one of these options to clients in one of two ways.

The first way is to define the data directly, using a text string or a colon-separated list of hexadecimal values. 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;
```

The second way of setting the value of these options is to have the DHCP server generate a vendor-specific options buffer. To do this, you must do four things: 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 relevant option.

To define a new option space to store vendor options, use the `option space` statement. The syntax of the statement is:

```
option space name [ [code width n] [length width n] [hash size n] ];
```

Where the numbers following `code width`, `length width`, and `hash size` respectively identify the number of bytes used to describe option codes, option lengths, and the size in buckets of the hash tables to hold options in this space (most option spaces use 1-byte codes and lengths, which is the default).

The code and length widths are used in the DHCP protocol – you must configure these numbers to match the applicable option space you are configuring. They each default to 1. Valid values for code widths are 1, 2 or 4. Valid values for length widths are 0, 1 or 2. The hash size defaults depend upon the code width selected, and may be 254 or 1009. Valid values range between 1 and 65535. Note that the higher you configure this value, the more memory will be used.

It is considered good practice to configure a value that is slightly larger than the estimated number of options you plan to configure within the space.

The name of the option space can be used in option definitions. For example:

```
option space SUNW code width 1 length width 1 hash size 3;
option SUNW.server-address code 2 = ip-address;
option SUNW.server-name code 3 = text;
option SUNW.root-path code 4 = text;
option space ISC code width 1 length width 1 hash size 3;
option ISC.sample code 1 = text;
option vendor.ISC code 2495 = encapsulate vivso-sample;
option vendor-class.ISC code 2495 = text;
option ISC.sample "configuration text here";
option vendor-class.ISC "vendor class here";
```

Once you have defined an option space and the format of 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 in the previous example, you can send different option values to different clients based on the `vendor-class-identifier` option that the clients send, 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";
option vivso-sample.sample “Hello world!”;
subclass "vendor-classes" "SUNW.Ultra-5_10" {
  vendor-option-space SUNW;
  option SUNW.root-path "tcpware:[sparc]";
}
subclass "vendor-classes" "SUNW.i86pc" {
  vendor-option-space SUNW;
  option SUNW.root-path "tcpware:[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.

Note that the VIVSO option can have multiple vendor definitions all at once (even transmitted to the same client), so it is not necessary to configure it this way.

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.

In order to prevent lease the file from becoming arbitrarily large, from time to time the DHCP server creates a new `dhcpd.leases` file from its in-memory lease database. Once this file has been written to disk, the old file is renamed `dhcpd.leases_old`, and the new file is renamed `dhcpd.leases`. If the system crashes in the middle of this process, whichever `dhcpd.leases` file remains will contain all the lease information, so there is no need for a special crash recovery process.

Declarations

The primary 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 the duration of the lease and to whom it is assigned.

The following table describes the statements the DHCP server puts into a lease file.

If DHCP failover is in use, the lease file will also contain failover-related statements. See the failover sections of this chapter for more information.

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.
`binding state` *`state`*`;` `next binding state` *`state`*`;`	Indicates the current lease state and what state the lease will move to when the current state expires. If failover is not in use, the state will be either `active` or `free`.
`client-hostname "`*`hostname`*`";`	Records the host name if the client sends one using the host-name option.
`ddns-text;`	Records the value of the client's TXT identification record.
`ddns-fwd-name;`	Records the name the server used to update the client's A record.
`ddns-client-fqdn;`	Records the name the client said it was using to update its own A record.
`ddns-rev-name;`	Records the name the server used to update the client's PTR record. The name to which the PTR record points will be either `ddns-fwd-name` or `ddns-client-fqdn`.
`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 described in the *Lease Date Format* section.
`hardware` *`hardware-type mac-address`*`;`	Specifies the hardware type and the MAC address as a series of hexadecimal octets, separated by colons.
`on` *`events`* `{` *`statements…`* `};`	Records a list of statements to be executed if the given event occurs. Possible events are `release` and `expiry`. If multiple events are specified, they are separated by the vertical bar character.
`option agent.circuit-id` *`string`*`;` `option agent.remote-id` *`string`*`;`	Records the circuit ID and remote ID options sent by the relay agent, if the relay agent uses the relay agent information option.
`reserved;`	Indicates that the lease is reserved. See the *Reserved Leases* Reserved Leasessection.
`set` *`variable = value`*`;`	Sets the value of a variable on the lease. For example, the vendor-class-identifier variable is used to record the client-supplied Vendor Class Identifier option. Other variables that are set are related to DDNS.
`starts` *`date`*`;`	Records the start time of a lease. Lease dates are specified by the DHCP server as described in the *Lease Date Format* section.
`uid` *`client-identifier`*`;`	Records the client identifier used by the client to acquire the lease. Clients are not required to send client identifiers, and this statement only appears if the client did in fact send one. Client identifiers are frequently an ARP type (for example, 1 for ethernet) followed by the MAC address, just like in the hardware statement, but this is not required. The client identifier is recorded as a colon-separated hexadecimal list or as a quoted string. If it is recorded as

	a quoted string and it contains one or more non-printable characters, those characters are represented as octal escapes – a backslash character followed by three octal digits.

Lease Date Format

Lease file dates are specified in one of two ways, depending on the configuration value for the `db-time-format` parameter. If it is set to `default`, then the date fields appear as follows:

```
weekday  year/month/day  hour:minute:second
```

The *`weekday`* is present to make it easier for a human to tell when a lease expires – it is specified as a number from zero to six, with zero being Sunday. Weekday is ignored on input. The *`year`* is specified with the century, so it should generally be four digits except for really long leases. The *`month`* is specified as a number starting with 1 for January. The *`day`* of the month is likewise specified starting with 1. The *`hour`* is a number between 0 and 23, the *`minute`* a number between 0 and 59, and the *`second`* also a number between 0 and 59.

Default format lease times are specified in Universal Coordinated Time (UTC), not in the local time zone.

If the `db-time-format` parameter is configured to `local`, then the date fields appear as follows:

```
epoch seconds-since-epoch;
# day-name month-name day-number hour:minutes:seconds year
```

The `seconds-since-epoch` is as according to the system's local clock (often referred to as "UNIX time"). The pound symbol (#) supplies a comment that describes what actual time this is according to the system's configured time zone, at the time the value was written. It is provided only for human inspection.

If a lease will never expire, date is displayed as `never` instead of an actual date.

Working with DHCP Leases

The DHCP server requires that a lease database be present before it will start. Before starting the DHCP server for the first time, make sure there is a `TCPWARE:DHCPD.LEASES` file. If it doesn't exist, create an empty one.

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 TCPWARE: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 TCPWARE: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 TCPWARE:DHCPD.LEASES_OLD to TCPWARE:DHCPD.LEASES, restoring the old, valid lease file, and then start the DHCP server. This guarantees that a valid lease file will be restored.

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 re-abandoned.

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.

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 NETCU SHOW DHCP4 commands do not have any lease information to display for static assignments.

- For SHOW DHCP/IP_ADDRESS, statically assigned IP addresses are not supported.
- For SHOW DHCP/SUBNET and /LEASES, statically assigned IP addresses are not shown.
- For SHOW DHCP/ALL, /HARDWARE_ADDRESS, and /CLIENT_IDENTIFIER, and in the dump file produced by /CONFIGURATION, statically assigned IP addresses are identified as a static assignment and no lease information is shown.
- For SHOW DHCP/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.

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 `TCPWARE:DHCPD.UPDATES`.

The commands that can be placed into the update file are described in section *Update File Statements* below.

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 `NETCU UPDATE DHCP4` command:

```
$ netcu update dhcp4
```

A different file name can optionally be specified:

```
$ netcu update dhcp4/filename=mydir:dhcpd.updates
```

You can verify the syntax of the update file before sending it to the DHCP server by using the `NETCU SHOW DHCP4/VERIFY` command:

```
$ netcu show dhcp4/verify=(update[=filename])
```

If the update file name is not specified, the file TCPWARE:DHCPD.UPDATES is read. Note that the configuration file is read in before the update file. A different configuration file can be specified using the CONFIG option:

```
$ netcu show dhcp4/verify=(config=filename,update[=filename])
```

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 netcu commands:

```
$ netcu show dhcp4/isknown host hw-addr-or-client-id
```

```
$ netcu show dhcp4/isknown subclass class-name subclass-data
```

Update File Statements

This table describes the commands you can use in an update file.

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 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;`

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";
```

DHCP 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 in this scenario operate without any communications or data sharing between them. Each server works as a standalone server.

Process Software takes the use of multiple servers to another level by offering DHCP Failover. DHCP Failover allows a secondary DHCP server to back up the primary DHCP server with the addition of taking into account network failure. This strategy ensures that clients can reliably log into their corporate network and know they will be able to connect to corporate resources. In failover mode, the primary and the backup DHCP servers share a common IP address lease pool.

The failover protocol defines a primary server role and a secondary server role. There are some differences in how primaries and secondaries act, but most of the differences simply have to do with providing a way for each peer to behave in the opposite way from the other. One server must be configured as primary, and the other must be configured as secondary, but it doesn't matter much which one is which.

The primary and secondary DHCP servers both give out and renew leases during normal operations. Each server will have about half of the available IP addresses in the pool at any given time for allocation. If one server fails, the other server will continue to renew leases out of the pool, and will allocate new addresses out of the roughly half of available addresses that it had when communications with the other server were lost.

Configuring DHCP Failover

To configure your DHCP servers to use 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 failover messages.
3. On the Primary system:

   Add a `failover peer` declaration to the `dhcpd.conf` configuration file with the keyword `primary` and other parameters, and add failover peer references to each address pool for which you want to do failover. See *DHCP Failover Configuration Statements* for more information.

4. On the Secondary system:

   Add a `failover peer` declaration to the `dhcpd.conf` configuration file with the keyword `secondary` and other parameters, and add failover peer references to each address pool for which you want to do failover.

5. If you don't 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 failover parameters as needed (see *Failover Configuration File Statements*).

   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.

   Note that you can write the `failover peer` declaration directly in the configuration file, or you can write it in a separate file and use the `include` statement to include this file in the main configuration file. This latter method is preferred because it allows both the primary and secondary to use the same main configuration file, avoiding configuration mismatches.

6. Copy the configuration file to the `TCPWARE` directory on both the Primary and the Secondary systems.
7. Make sure that both the Primary and the Secondary systems have lease files in the `TCPWARE` directory. 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 and you're in business!

Failover Configuration File Statements

In order to configure failover in DHCP, you need to write a `failover peer` declaration that configures the failover protocol, and you need to include peer references in each pool declaration for which you want to do failover. You do not have to do failover for all pools on a given network segment. You must not tell one failover partner it's doing failover on a particular address pool and tell the other it is not. You must not have any common address pools on which you are not doing failover. A pool declaration that utilizes failover would look like this:

```
pool {
  failover peer "foo";
  pool specific parameters
};
```

For safety, we recommend that you either do failover or don't do failover, but don't do any mixed pools. Also, we recommend that you use the same master configuration file for both servers, and have a separate file for each failover partner that contains that partner's peer declaration and includes the master file. This will help you to avoid configuration mismatches.

A basic sample `dhcpd.conf` file for a primary server might look like this:

```
    failover peer "foo" {
      primary;
      address pri.example.com;
      port 519;
      peer address sec.example.com;
      peer port 520;
      max-response-delay 60;
      max-unacked-updates 10;
      mclt 3600;
      split 128;
      load balance max seconds 3;
    }
    include "tcpware:dhcpd.master";
```

The following table lists the statements that can appear in the `failover peer` declaration:

Failover Peer Statement	**Description**
`primary` or `secondary`	Determines whether the server is the primary server or the secondary server in the failover partnership. Only one may be specified on a given system.

	`primary;` `secondary;`
`address` *`ip-address`*`;`	Specifies the IP address or DNS name upon which the server should listen for connections from its failover peer. Also, the value to use for the DHCP Failover Protocol server identifier. Required.
`peer address` *`ip-address`*`;`	Specifies the IP address or DNS name to which the server should connect to reach its failover peer for failover messages.
`port` *`port-number`*`;`	Specifies the TCP port on which the server should listen for connections from its failover peer. Optional. The default is 647.
`peer port` *`port-number`*`;`	Specifies the TCP port to which the server should connect to reach its failover peer for failover messages. Optional. The default is 647.
`max-response-delay` *`seconds`*`;`	Tells the DHCP server how many seconds may pass without receiving a message from its failover peer before it assumes that connection has failed. This number should be small enough that a transient network failure that breaks the connection will not result in the servers being out of communication for a long time, but large enough that the server isn't constantly making and breaking connections. Required.
`max-unacked-updates` *`count`*`;`	Tells the failover peer how many BNDUPD (bind update) messages it can send before it receives a BNDACK (bind ack) from the local system. Required.
`mclt` *`seconds`*`;`	Defines the Maximum Client Lead Time. It is required on the primary, and optional on the secondary. This is the length of time for which a lease may be renewed by either failover peer without contacting the other. The longer you set this, the longer it will take for the running server to recover IP addresses after moving into `partner-down` state. The shorter you set it, the more load your servers will experience when they are not communicating. A value of 3600 seconds is reasonable.

`split` *`index;`*	Specifies the split between the primary and secondary for the purposes of load balancing. Whenever a client makes a DHCP request, the DHCP server runs a hash on the client identification, resulting in a value from 0 to 255. This is used as an index into a 256-bit field. If the bit at that index is set, the primary is responsible. If the bit at that index is not set, the secondary is responsible. The `split` value determines how many of the leading bits are set to one. So, in practice, higher split values will cause the primary to serve more clients than the secondary. Lower split values, the converse is true. Legal values are between 0 and 255, of which the most reasonable is 128 (a 50/50 split). Cannot be specified in the secondary. The primary must specify either `split` or `hba`, but not both.

`hba` *`colon-separated-hex-list`*	Specifies the split between the primary and secondary as a bitmap rather than a cutoff (as `split` does), which theoretically allows for finer-grained control. In practice, there is probably no need for such fine-grained control, however. An example `hba` statement:
	
	`hba ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff: 00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00;`
	
	This is equivalent to a `split 128;` statement, and identical. The following example is also equivalent to a split of 128, but is not identical:
	
	`hba aa:aa:aa:aa:aa:aa:aa:aa:aa:aa:aa:aa:aa:aa:aa:aa: aa:aa:aa:aa:aa:aa:aa:aa:aa:aa:aa:aa:aa:aa:aa:aa;`
	
	They are equivalent, because half the bits are set to 0, half are set to 1 (0xa is 1010 binary) and consequently this would roughly divide the clients equally between the servers. They are not

	identical, because the actual peers this would load balance to each server are different for each example. You must only have `split` or `hba` defined in the primary, never both. For most cases, the fine-grained control that `hba` offers isn't necessary, and `split` should be used. Cannot be specified in the secondary.
`load balance max seconds` *`seconds;`*	Allows you to configure a cutoff after which load balancing is disabled. The cutoff is based on the number of seconds since the client sent its first DHCPDISCOVER or DHCPREQUEST message, and only works with clients that correctly implement the secs field. We recommend setting this to something like 3 or 5. The effect of this is that if one of the failover peers gets into a state where it is responding to failover messages but not responding to some client requests, the other failover peer will take over its client load automatically as the clients retry.
`auto-partner-down` *`seconds;`*	Instructs the server to initiate a timed delay upon entering the `communications-interrupted` state (out of contact with the failover peer). At the conclusion of the timer, the server will automatically enter the `partner-down` state. See the *DHCP Failover Partner-Down State* section for more information. A zero value (the default) disables the auto-partner-down feature, and any positive value indicates the time in seconds to wait before automatically entering `partner-down`.
`pool balance` *`statements`*	See the *DHCP V4 Pool Balancing* section for information about pool balancing. `max-lease-misbalance` *`percentage;`* `max-lease-ownership` *`percentage;`* `min-balance` *`seconds;`* `max-balance` *`seconds;`*

DHCP Pool Balancing

The DHCP server evaluates pool balance on a schedule, rather than on demand as leases are allocated (which was the old approach).

In order to understand pool balance, some elements of its operation first need to be defined. First, there are `free` and `backup` leases. Both of these are referred to as `free-state` *leases*. The states `free` and `backup` are "the free states" for the purpose of this discussion. The difference is that only the primary may allocate from `free` leases unless under special circumstances, and only the secondary may allocate `backup` leases.

When pool balance is performed, the only plausible expectation is to provide a 50/50 split of the free-state leases between the two servers. This is because no one can predict which server will fail, regardless of the relative load placed upon the two servers, so giving each server half the leases gives both servers the same amount of failure endurance. Therefore, there is no way to configure any different behavior, outside of some very small windows described shortly.

The first thing calculated on any pool balance run is a value referred to as `lts`, or "Leases To Send". This, simply, for the primary is the difference in the count of free and backup leases, divided by two. For the secondary, it is the difference in the backup and free leases, divided by two. The resulting value is signed: if it is positive, the local server is expected to hand out leases to regain a 50/50 balance. If it is negative, the remote server would need to send leases to balance the pool. Once the `lts` value reaches zero, the pool is perfectly balanced (give or take one lease in the case of an odd number of total free-state leases).

The current approach is still something of a hybrid of the old approach, marked by the presence of the `max-lease-misbalance` parameter. This parameter configures what used to be a 10% fixed value in earlier versions: if `lts` is less than the number that are `free` plus the number that are `backup` times the `max-lease-misbalance` percentage, then the server will skip balancing a given pool (it won't bother moving any leases, even if some leases "should" be moved).

The meaning of this parameter is also somewhat overloaded, however, in that it also governs the estimation of when to attempt to balance the pool (which may then also be skipped over). The oldest leases in the free and backup states are examined. The time they have resided in their respective queues is used as an estimate to indicate how much time it is probable it would take before the leases at the top of the list would be consumed (and thus, how long it would take to use all leases in that state). This percentage is directly multiplied by this time, and fit into the schedule if it falls within the `min-balance` and `max-balance` configured values. The scheduled pool check time is only moved in a downwards direction, it is never increased. Lastly, if the `lts` is more than double this number in the negative direction, the local server will 'panic' and transmit a Failover protocol POOLREQ (pool request) message, in the hopes that the remote system will be woken up into action.

Once the `lts` value exceeds the `max-lease-misbalance` percentage of total free-state leases as described above, leases are moved to the remote server. This is done in two passes:

In the first pass, only leases whose most recent bound client would have been served by the remote server – according to the Load Balance Algorithm (see `split` and `hba` parameters above) – are given away to the peer. This first pass will happily continue to give away leases, decrementing the `lts` value by one for each, until the `lts` value has reached the negative of the total number of leases multiplied by the `max-lease-ownership` percentage. So, it is through this value that you can permit a small misbalance of the lease pools – for the purpose of giving the peer more than a 50/50 share of leases in the hopes that their clients might someday return and be allocated by the peer (operating normally). This process is referred to as "MAC Address Affinity", but this is somewhat misnamed: it applies equally to DHCP Client Identifier options. Note also that affinity is applied to leases when they enter the state `free` from `expired` or `released`. In this case also, leases will not be moved from `free` to `backup` if the secondary already has more than its share.

The second pass is only entered into if the first pass fails to reduce the `lts` underneath the total number of free-state leases multiplied by the `max-lease-ownership` percentage. In this pass, the oldest leases are given over to the peer without second thought about the Load Balance Algorithm, and this continues until the `lts` falls under this value. In this way, the local server will also happily keep to itself a small percentage of the leases that would normally load balance.

So, the `max-lease-misbalance` value acts as a behavioral gate. Smaller values will cause more leases to transition states to balance the pools over time, higher values will decrease the amount of change (but may lead to pool starvation if there's a run on leases).

The `max-lease-ownership` value permits a small (percentage) skew in the lease balance of a percentage of the total number of free-state leases.

Finally, the `min-balance` and `max-balance` parameters make certain that a scheduled rebalance event happens within a reasonable timeframe (not to be thrown off by, for example, a 7-year-old free lease).

Plausible values for the `max-lease-misbalance` and `max-lease-ownership` percentages lie between 0 and 100, inclusive, but values over 50 are indistinguishable from one another (once lts exceeds 50% of the free-state leases, one server must therefore have 100% of the leases in its respective free state). It is recommended to select a `max-lease-ownership` value that is lower than the value selected for the `max-lease-misbalance` value. `max-lease-ownership` defaults to 10, and `max-lease-misbalance` defaults to 15.

Plausible values for the `min-balance` and `max-balance` times (in seconds) range from 0 to (2^{32})-1, but default to values 60 and 3600 respectively (to place balance events between 1 minute and 1 hour).

Failover Lease File Statements

The statements listed in this section have been added to the DHCP lease file for failover. These are in addition to the DHCP lease file statements listed above.

DHCP Failover Lease Statements

Statement	Description
`tstp` *`date;`*	The time the peer has been told the lease expires.
`tsfp` *`date;`*	The lease expiry time that the peer has acknowledged.
`atsfp` *`date;`*	The actual time sent from the failover partner.
`cltt` *`date;`*	The client's last transaction time.
`binding state` *`state;`* `next binding state` *`state;`*	When the DHCP server is not configured to use Failover, the only possible binding states are `active` or `free`. With failover, additional states are added, including the `backup` state, which indicates that the lease is available for allocation by the secondary.

`failover peer state`	Records the state of the primary and secondary failover partners.
	
	`failover peer "`*`name`*`" state {`
	`  my state` *`state`* `at` *`date`*`;`
	`  partner state` *`state`* `at` *`date`*`;`
	`  mclt` *`seconds`*`;`
	`};`
	
	The states of the peer named *`name`* is recorded in this declaration. Both the state of the running server (my state) and the other failover partner (partner state) are recorded. The following states are possible:

	`unknown-state, partner-down, normal, communications-interrupted, resolution-interrupted, potential-conflict, recover, recover-done, shutdown, paused, startup.` The `mclt` records the Maximum Client Lead Time (from the configuration file).

DHCP Failover Partner-Down State

It is possible during a prolonged failure to tell the remaining DHCP failover peer that the other server is down, in which case the remaining server will (over time) reclaim all the addresses the other server had available for allocation, and begin to reuse them. This is called putting the server into the *partner-down* state.

Be careful about transitioning a failover server to *partner-down* state. The *partner-down* and *communications-interrupted* states are intentionally distinct because there do exist situations where a failover server can fail to communicate with its peer, but still has the ability to receive and reply to requests from DHCP clients. Partner-down state permits the server to allocate leases from the partner's free lease pool after the `mclt` timer expires. This can be dangerous if the partner is in fact operating at the time (the two servers will give conflicting bindings).

It is also possible to get into this kind of dangerous situation: you put one server into the *partner-down* state, and then that server goes down, and the other server comes back up. The other server does not know that the first server was in the partner-down state, and may issue addresses previously issued by the other server to different clients, resulting in IP address conflicts. Before putting a server into partner-down state, therefore, make sure that the other server will not restart automatically.

Also, the `automatic` partner-down feature should be used only with extreme caution. In general, it should be only be used at sites where the failover servers are directly connected to one another, such as by a dedicated hardwired link ("a heartbeat cable"). In other cases, the network manager should verify that the failover partner is really down and not coming back any time soon before putting a failover server into partner-down state manually.

When the down server comes back online, it automatically detects that it has been offline and requests a complete update from the server that was running in the *partner-down* state, and then both servers resume normal processing.

Transitioning to Partner-Down State

There are three ways that you can transition the DHCP server to *partner-down* state:

1. The server may transition to *partner-down* state automatically based on the `auto-partner-down` parameter of the `failover peer` statement

   If the above parameter is specified in the configuration file, the DHCP server transitions to *partner-down* state automatically after it has been in *communications-interrupted* state for the specified time.
2. You can put the DHCP server into Partner Down state using the following NETCU command:

   ```
   $ netcu set dhcp4/partnerdown
   ```
3. You can edit the DHCP lease file and specify that the server should be in *partner-down* state.

   Edit the last `failover peer` state declaration in the lease file to set `my state` to `partner-down`, for example: (note that it is only required to set the `my state` value)

   ```
   failover peer "name" state {
     my state partner-down;
     partner state state at date;
   }
   ```

   The next time the DHCP server is restarted, it will be in *partner-down* state. You can restart the DHCP server by shutting down the server with the `NETCU STOP/DHCP4` command. NETCP automatically restarts the DHCP server.

Failover Server Restart

When a server starts that has not previously communicated with its failover peer, it must establish communications with its peer and synchronize with it before it can serve clients. This can happen either because you have just configured your DHCP servers to perform failover for the first time, or because one of your failover servers has failed catastrophically and lost its database.

The initial recovery process is designed to ensure that when one failover peer loses its database and then resynchronizes, any leases that the failed server gave out before it failed will be honored. When the failed server starts up, it notices that it has no saved failover state, and attempts to contact its peer.

When it has established contact, it asks the peer for a complete copy of its peer's lease database. The peer then sends its complete database, and sends a message indicating that it is done. The failed server then waits until `mclt` has passed, and once `mclt` has passed both servers make the transition back into normal operation. This waiting period ensures that any leases the failed server may have given out while out of contact with its partner will have expired.

While the failed server is recovering, its partner remains in the *partner-down* state, which means that it is serving all clients. The failed server provides no service at all to DHCP clients until it has made the transition into normal operation.

In the case where both servers detect that they have never before communicated with their partner, they both come up in this recovery state and follow the procedure just described. In this case, no service will be provided to DHCP clients until `mclt` has expired.

Sample DHCPD.CONF File

The following is a sample `DHCPD.CONF` file.

```
#
# TCPWARE: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 don’t 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
# supersede the global options specified earlier.
subnet 192.5.5.0 netmask 255.255.255.224 {
    range 192.5.5.26 192.5.5.30;
    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 192.5.5.31;
    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";
}
# Do not allow this one to boot
host host2
    hardware ethernet aa:cc:04:00:33:11;
    deny booting;
}
# Some other examples
host host1 {
    option dhcp-client-identifier "host1";
    fixed-address 10.10.11.101, 10.11.22.101;
}
```

5. Serial Link Interfaces: PPP and SLIP

Introduction

This chapter describes the serial link interfaces available with TCPware. There are two types of serial link interfaces:

- Point-to-Point Protocol (PPP)
- Serial Line IP (SLIP)

Point-to-Point Protocol Interface

TCPware supports the Point-to-Point Protocol (PPP) so that you can send IP datagrams over serial links, including LAT or modem connections.

PPP is an enhancement to the nonstandard Serial Line IP (SLIP) interface (see *Serial Line IP Interface*). PPP provides self-contained error detection and automatically negotiated header compression. It also provides authentication through the Password Authentication Protocol (PAP) which you can set using `PPPD` command options.

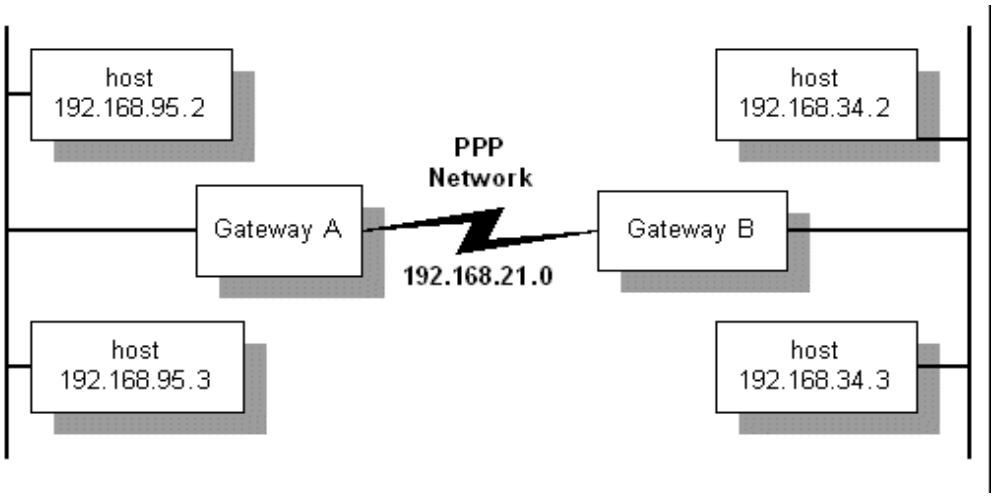
You configure PPP on the TCPware host using the `PPPD` command and its options at the DCL prompt or aggregated in an options file. You do not need to configure PPP using the configuration procedure (`CNFNET`) or Network Control Utility (`NETCU`) commands.

Implementation

PPP is a standard (as Internet STD 51, or RFC 1661) for transporting multiprotocol datagrams over serial point-to-point links. PPP is composed of three parts:

- A method to encapsulate multiprotocol datagrams over serial links.
- An extensible Link Control Protocol (LCP) to establish, configure, and test the data link connection.
- A family of Network Control Protocols (NCPs, such as the IP Control Program and the authentication protocols) to establish and configure the different network layer protocols.

You need TCP/IP and application software at each end of the PPP link. One end can be TCPware, while the other end (known as the *peer*) can be any PPP client implementation. The below diagram shows a typical PPP network.



Because PPP lines are point-to-point connections, with TCPware you have the option to configure PPP lines as network lines or as unnumbered interface lines. The advantage to unnumbered interface lines is that you do not need to assign them IP addresses.

TCPware supports both dedicated and dialup PPP lines. PPP is common with line speeds from 14.4 to 28.8 kilobits per second (Kb/s).

PPP is implemented as a single program, PPPD, the PPP daemon, which runs as a foreign command. The daemon is a process that:

- Runs between the terminal driver and IP driver.
- Negotiates PPP line configuration with a peer PPP node.
- Establishes the PPP connection between the specified serial line and TCPware.

There is one PPPD process for each physical connection.

You control PPP through PPPD command line options, which you can add to an options file (`TCPWARE:PPPOPTIONS.DAT`).

Before Configuring PPP Lines

Before you begin configuring PPP lines:

- You must set the `TTY_ALTYPAHD` system parameter larger than its default value. This avoids losing characters. The higher the line speed, the higher you should set this parameter. For most applications, 1024 is appropriate.

TTY_ALTYPAHD is not a dynamic parameter. If you use SYSGEN to change it, you must reboot the system for the change to take effect.

- The MAXBUF system parameter must be at least twice the maximum transmission unit (MTU) of the PPP line plus 134. The default MTU for PPP lines is 1500 bytes; therefore, MAXBUF must be at least 3134. Increase MAXBUF if necessary. MAXBUF is dynamic. If you use SYSGEN to change it, you do not need to reboot the system for the change in value to take effect.
- Determine the speed of your serial line or modem. You may need to specify this line speed as a PPPD command parameter.
- Make sure that the terminal device you specify is allocated to your current process.

PPPD Command

You configure PPP lines on your local system using the PPPD command at the DCL prompt and specifying the appropriate options. You start PPPD as a foreign command as follows:

```
$ PPPD :== $TCPWARE:PPPD.EXE
$ PPPD options
```

When you start PPPD, it either begins sending PPP packets to the specified terminal device to start negotiating, or waits for PPP packets to arrive, depending on the *options* specified on the command line. When negotiation is complete and IP is up, you can tell if the PPP device is configured by using the NETCU command SHOW NETWORK:

```
$ NETCU SHOW NETWORK
TCPware(R) for OpenVMS Internet Network Information:
Line   Local Address  Subnet Mask    MTU    Xmits   Errs  Recvs Errs  RBU
----   -------------  -----------    ---    -----   ----  ----- ----  ---
PPP-0  192.168.142.57 255.255.255.0  1500   1       0     1     0     0
LPB-0  127.0.0.1      255.0.0.0      64512  74      0     74    0     0
...
```

The PPP line shows up as PPP-*n*, the *n* starting with 0 and incrementing for each new line.

A PPP session is terminated in one of the following ways:

- If you enter the NETCU STOP/IP command for the PPP line
- The peer terminates the PPP session
- The serial line is hung up (for modem lines)
- The PPPD process is stopped

The PPPD process either ends or listens on the line for more incoming PPP packets, depending on the command line option used. PPP also ends when it cannot agree on the option negotiation during startup.

By default, PPPD creates a detached process to which it hands over the terminal device. To execute PPPD in foreground mode, use the -DETACH (or -NODETACH) option with the PPPD command. (Note that special privileges apply to a detached process. See the -DETACH option in the PPPD command reference for details.) The name of the detached process is PPP_*terminal-device-name*.

The TCPWARE:PPPOPTIONS.DAT file can contain any option you can specify on the PPPD command line. Options in the file have precedence over the options on the command line. The options file can contain any of the PPPD options, separated with spaces or tabs. You can specify options in multiple lines, as in the following example:

```
NETMASK 255.255.252.0 ASYNCMAP 0
NAME FLOWERS.EXAMPLE.COM
AUTH +PPP
```

You can specify command files in four different PPPD options, as described in the below table.

Option	Description
CONNECT *file*	Sets the terminal device through the specified command file
DISCONNECT *file*	Resets the terminal device through the specified command file
IPUP *file*	The specified file executes when IP is started over PPP
IPDOWN *file*	The specified file executes when IP over PPP is shut down

The command file associated with the CONNECT and DISCONNECT options must return an exit status to PPPD with the EXIT command. PPPD waits for the script to finish and terminates if the return status is not successful (the status code is an odd number). The command files are provided with the device name as the P1 parameter. You can use a dialup scripting program such as KERMIT.

The IPUP and IPDOWN options are executed asynchronously so that PPPD does not wait for their completion. They are provided with the following command parameters:

Parameter	Description
P1	Interface name (such as PPP-0)
P2	Terminal device (such as _TTA3:)

P3	Local IP address
P4	Remote IP address
P5	(Optional) String specified with the `IPPARAM` option

Configuring PPP Links

The PPPD command line parameter and options shown in the below table control the PPP link configuration and specify the basic characteristics of the PPP link.

Option	**Description**
`device-name`	Terminal device, such as TTA3
`ASYNCMAP` *`map`*	Bit map of characters to escape
`CRTSCTS`	Uses hardware flow control
`ESCAPE` *`xx,yy,...`*	Escape character definitions
`MRU` *`n`*	Maximum Receive Unit (MRU)

See the *Command Reference* section in this chapter for details on each option.

By default, PPPD starts sending out configuration requests to the peer to establish a connection, and terminates when the connection shuts down, negotiation fails, or the peer does not respond within a set timeout period. You can change this course of action using the options shown in the below table.

Option	**Description**
`PASSIVE`	Initiates negotiation but waits passively for the peer to respond if that fails
`SILENT`	Passively waits until the PPP packet arrives

PERSIST	After the PPP connection is terminated, waits for a new connection without exiting (in the case of a dial-out, reestablishes the connection by redialing)

Authentication

PPPD provides sufficient access control. You can provide legitimate users PPP access to a server machine, without fear of compromising the security of the server or the network it is on. This access control is available as a combination of the following:

- The TCPWARE:PPPOPTIONS.DAT file, where you can place options to require authentication when running PPPD.
- Password Authentication Protocol (PAP) secrets files where you can restrict the set of IP addresses for individual users

PPPD's default action is to agree to authenticate if requested, and not to require authentication from the peer. However, PPPD does not agree to authenticate itself with a particular protocol if it has no secrets it can use to do so.

You can change this behavior with the command line options shown in the below table:

Option	Description
AUTH	Peer authenticates (any authentication)
+PAP	Peer authenticates with PAP
-PAP	Do not agree to authenticate with PAP

Using the Password Authentication Protocol

The Password Authentication Protocol (PAP) provides a simple method for the peer to establish its identity. PAP uses a two-way handshake with a simple name and password combination. This handshake occurs only on establishing the initial PPP link.

PAP is not a strong authentication method. Passwords go over the circuit as clear text, and there is no protection from playback or repeated trial and error attacks.

> **Note:** You can optionally authenticate using the user/password combination in the OpenVMS UAF file by option `login`.

Using the Challenge Handshake Authentication Protocol

The Challenge Handshake Authentication Protocol (CHAP) is a stronger method than PAP of authenticating the PPP link, and is the preferred method. A CHAP secret (password) is encrypted, and you can repeat authentication periodically during the session using different challenge values.

Authentication Files

Authentication information is stored in the `TCPWARE:PPPPAP.DAT` file for PAP authentication or the `TCPWARE:PPPCHAP.DAT` file for CHAP authentication. The contents of these files are used both for authenticating incoming peer hosts and authenticating the local host to remote peers. You can use the `TCPWARE:PPPSECRET.TEMPLATE` file as a template for both, since the format for both is identical.

The following example shows a `PPP.DAT` entry that authenticates the local host to a peer:

```
# TCPWARE:PPP.DAT
#
# local/user name   server name   secret/password   [address restriction]
# ---------------   -----------   ---------------   ---------------------
  skat              lear          SecretString      198.168.142.57
```

Both files are multi-columned text files. Comment lines in `PPP.DAT` start with the pound sign (#). Each line consists of three fields with additional optional fields, as follows:

```
field1   field2   field3   optional-fields
```

White space separates the fields. You can use the asterisk (*) wildcard in the first and second fields. Optional fields can contain lists of authorized peer IP addresses. If the optional field is omitted, any IP address is allowed.

The fields have different meanings for PAP and CHAP authentication, and its direction, as given in the below table:

PAP authentication of incoming peer:

field1 local-username	field2 peer-hostname	field3 user-password	optional
PAP authentication of local host to peer:			
field1 peer-username	field2 peer-hostname	field3 user-password	optional
CHAP authentication of local host to peer:			
field1 peer-hostname	field2 local-hostname	field3 CHAP-secret-string	optional
field1 local-hostname	field2 peer-hostname	field3 CHAP-secret-string	optional

Modifying Authentication Names

You can use several PPPD command line options to modify names used in authentication. These options are shown below:

Option	Description
`DOMAIN d`	Appends the domain name *d* to the local hostname
`NAME n`	Sets the local hostname to *n*
`REMOTENAME n`	Sets the assumed remote hostname to *n*
`USEHOSTNAME`	Uses the system-specified local host name
`USER u`	Sets the username to *u* for PAP

IP Addresses

The parameter and options shown below are related to configuring IP addresses.

Parameter or option	Description
`local-IP-address:`*`remote-IP-address`*	Local and remote IP addresses; either can be omitted
`-IP`	Disables IP address negotiation
`NETMASK` *n*	Sets the interface mask to *n*
`NOIPDEFAULT`	Disables use of the local IP address as the default

It is usually not necessary to specify the IP addresses. By default, each peer uses its default IP address if it knows it.

Incoming Dialup Lines

Perform these steps to set up an incoming dialup PPP line:

1. Create a login account for the PPP site using the OpenVMS AUTHORIZE utility. This should be a captive account and must have OPER privileges.
2. Create a `LOGIN.COM` file for this account. The `TCPWARE:PPPLOGIN.TEMPLATE` file is available. Below is a sample `LOGIN.COM` file for an unnumbered interface:

```
$ ON WARNING THEN LOGOUT
$ IF (F$TRNLNM("TCPWARE_NETCP_MBX") .EQS. "") THEN GOTO NOTCPWARE
$ PPPD :== $TCPWARE:PPPD
$ TT = F$TRNLNM("TT")
$ WRITE SYS$OUTPUT "Starting PPP..."
$ DEFINE/USER SYS$ERROR NLA0:
$ DEFINE/USER SYS$OUTPUT NLA0:
$ PPPD PROXYARP :192.168.95.12 'TT' -DETACH
$ WRITE SYS$OUTPUT "Shutting down PPP..."
$ EXIT
$ NOTCPWARE:
$ WRITE SYS$OUTPUT "%PPP-F-NOTACT, TCPware not active"
$ LOGOUT
```

Once you set up the account and login file, the remote site dials the OpenVMS system and logs in as the PPP user to establish the connection. The login command file automatically configures the PPP line.

Create a separate account and LOGIN.COM file for each remote PPP site.

> **Note:** Unlike TCPware's SLIP implementation, this account can remain logged in during the PPP session. You can use ordinary OpenVMS user accounting to charge the user for the PPP connection. If you prefer to free up login sessions, you can remove the -DETACH option.

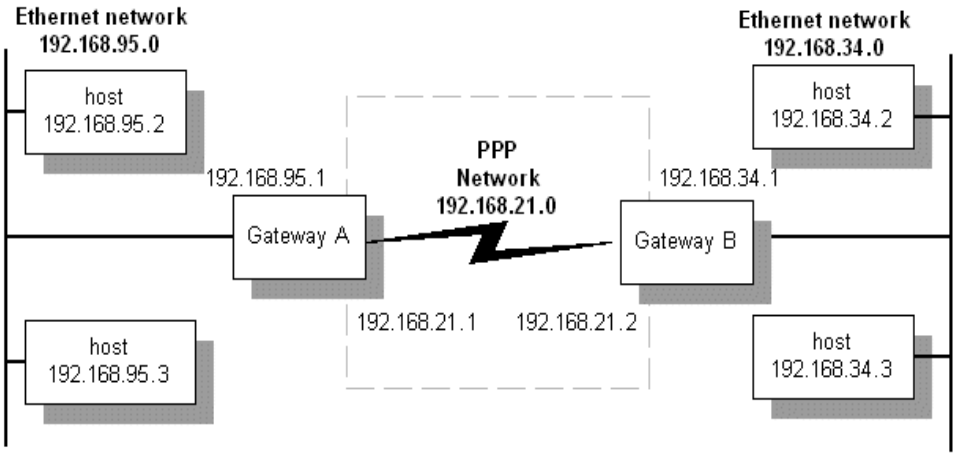
Routing

The options shown below are related to configuring routes.

Option	Description
DEFAULTROUTE	Uses the remote host as the default gateway
-IP	Disables IP address negotiation
NETMASK *n*	Sets the interface mask to *n*
PROXYARP	Starts the line as an unnumbered interface and enables proxy ARP
-PROXYARP	Disables proxy ARP

Traditional Numbered Interfaces

The below diagram shows a sample internet consisting of three networks: Ethernet network 192.168.95.0, PPP network 192.168.21.0, and Ethernet network 192.168.34.0.



Each gateway has an internet address for each network to which it connects. In this example, PPP network 192.168.21.0 is set up so that networks 192.168.95.0 and 192.168.34.0 can communicate.

Initiate the PPP link from local host 192.168.95.1 to peer 192.168.34.1 as follows:

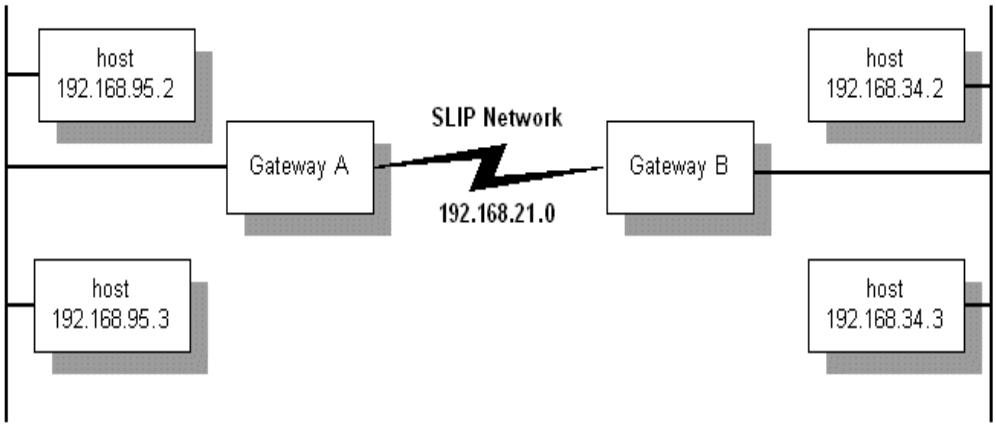
```
$ PPPD 192.168.21.1:192.168.21.2 NETMASK 255.255.255.0 TTA2
```

Perform a similar command on local host 192.168.34.1. You must also set up routing through the established connection. To do so in this setup, create a command file with the following contents for Gateway B, and use the PPPD `IPUP` option to specify the command file:

```
$ RUN TCPWARE:NETCU ADD ROUTE 192.168.34.0 192.168.21.1 /NETWORK
/GATEWAY
```

Unnumbered Interfaces

In the case of a single host connection over a PPP line to a network, you may not be able to dedicate a separate network number to the connection. This requires you to use an unnumbered interface. The below diagram shows such a scenario.



Here is how you would set up the connection on host 192.168.34.1:

```
$ PPPD 192.168.34.1:192.168.34.4 NETMASK 255.255.255.0 PROXYARP TTA2
```

The `PROXYARP` option lets host 192.168.34.1 respond to Address Resolution Protocol (ARP) requests for the remote host's address. In this way, other hosts on the 192.168.34.0 network can send any packets addressed to 192.168.34.2. This is known as "proxy ARP" and keeps you from having to add the host routes on all other hosts in network 192.168.34.0.

The `PROXYARP` option does the following:

- Starts the PPP interface as unnumbered
- Enables proxy ARP for the remote node
- Adds a host route to the remote node through the PPP interface

TCP/IP Header Compression

PPPD also provides the option to compress TCP/IP headers using the Van Jacobson (VJ) header compression algorithm. Compression is enabled unless you for some reason want to disable it. The options shown below are related to TCP/IP header compression.

Option	Description
`-VJ`	Disables VJ compression
`-VJCCOMP`	Disables VJ Connection ID compression
`VJ-MAX-SLOTS` *n*	Sets the number of VJ connection slots to *n*

Command Reference

The following pages include the command reference for the PPPD command. The *options* include parameters and options (some with arguments). Parameters and options are listed alphabetically.

Frequently used parameters and options include:

PPPD

Provides the basic Link Control Protocol (LCP), authentication support, and a Network Control Protocol (NCP) for establishing and configuring the IP Control Protocol (IPCP). Parameters and options are listed alphabetically for reference.

Requires OPER and PHY_IO privileges. Requires READALL privilege if secret files are used. Requires TMPMBX, DETACH, and SHARE privileges for running in detached mode.

Format

`PPPD [parameters | options]`

Parameters

device-name

Communicates over the named device. If no device name is given, or the name of the controlling terminal is given, PPPD uses the controlling terminal.

local-IP-address:remote-IP-address

Sets the local or remote interface IP addresses. Either one may be omitted. The IP addresses can be specified with a hostname or in decimal dot notation (such as 150.234.56.78). The default local address is the (first) IP address of the system (unless the `NOIPDEFAULT` option is given). The remote address is obtained from the peer if not specified in any option. Thus, in simple cases, this option is not required. If a local and/or remote IP address is specified with this option, PPPD does not accept a different value from the peer in the IPCP negotiation, unless the `IPCP-ACCEPT-LOCAL` or `IPCP-ACCEPT-REMOTE` options are given, respectively.

Options

-AC

Disables address/control compression negotiation (default).

-ALL

Disables requesting or allowing negotiation of any options for LCP and IPCP (uses the default values).

-AM
Disables ASYNCMAP negotiation (uses the default ASYNCMAP, which is to escape all control characters).

ASYNCMAP *map*
-AS *map*
Sets the async character map to *map*, which describes which control characters cannot be successfully received over the serial line. The peer is requested to send these characters as a two-byte escape sequence. The argument is a 32-bit hex number with each bit representing a character to escape. Bit 0 (00000001) represents the character `0x00`; bit 31 (80000000) represents the character `0x1f` (Ctrl/_). If multiple ASYNCMAP options are given, the values are OR'd together. If no ASYNCMAP option is given, no async character map will be negotiated for the receive direction; the peer should then escape all control characters.

AUTH
Requires the peer to authenticate itself before allowing network packets to be sent or received.

CONNECT *command-file*
Uses a DCL command file to set up the serial line. If used with `-D`, debug output is logged in the *command-file*`.LOG` file.

CRTSCTS
Uses hardware flow control (RTS/CTS) to control the flow of data on the serial port. If you use neither `CRTSCTS` nor `-CRTSCTS`, the hardware flow control setting for the serial port is unchanged.

-CRTSCTS
Disables hardware flow control (RTS/CTS) on the serial port. If you use neither CRTSCTS nor -CRTSCTS, the hardware flow control setting for the serial port is unchanged.

DEBUG
-D
Enables debugging.

DEFAULTROUTE
Adds a default route to the system routing tables, using the peer as the gateway, when IPCP negotiation is successfully completed. This entry is removed when the PPP connection breaks.

-DEFAULTROUTE
Disables the `defaultroute` option. If you want to prevent users from creating default routes with PPPD, place this option in the `TCPWARE:PPPDOPTIONS.DAT` file.

-DETACH
-NODETACH
Executes PPPD in foreground mode. Normally, PPPD creates a detached process to which it hands over the terminal device. To start in detached mode, PPPD requires the DETACH privilege, along with TMPMBX and SHARE privileges if you are using your login device as a PPP device. `-DETACH` and `-NODETACH` are identical.

DISCONNECT *command-file*
Runs the DCL command file after PPPD terminates the link. This command file could issue commands to the modem to hang up, if hardware modem control signals were not available. If used with `-D`, debug output is logged in the *`command-file`*`.LOG` file.

DNS *address*
Identifies the primary Domain Name System (DNS) name server. If omitted, PPPD uses the first name server specified by the `TCPWARE_NAMESERVERS` logical, set up through the DNS configuration.

DOMAIN *d*
Appends the domain name *`d`* to the local host name for authentication purposes. For example, if `gethostname()` returns the name IRIS, but the fully qualified domain name is IRIS.EXAMPLE.COM, you would use the `DOMAIN` option to set the domain to EXAMPLE.COM.

ESCAPE *xx,yy,...*
Escapes the specified characters on transmission (regardless of whether the peer requests them to be escaped with its async control character map). Specify the characters to be escaped as a list of hex numbers separated by commas. Note that you can specify almost any character for the `ESCAPE` option,

unlike the ASYNCMAP option that only lets you specify control characters. The characters that cannot be escaped are those with hex values 0x20 through 0x3f, and 0x5e.

FILE *file*
Reads options from a file.

-IP
Disables IP address negotiation. If used, you must specify the remote IP address with an option on the command line, or in the TCPWARE:PPPOPTIONS.DAT file.

IPCP-ACCEPT-LOCAL
Accepts the peer's interpretation of the local IP address, even if the local IP address was specified in an option.

IPCP-ACCEPT-REMOTE
Accepts the peer's interpretation of its (remote) IP address, even if the remote IP address was specified in an option.

IPCP-MAX-CONFIGURE *n*
Sets the maximum number of IPCP configure-request transmissions to *n* (default 10).

IPCP-MAX-FAILURE *n*
Sets the maximum number of IPCP configure-NAKs returned before starting to send configure-Rejects instead to *n* (default 10).

IPCP-MAX-TERMINATE *n*
Sets the maximum number of IPCP terminate-request transmissions to *n* (default 3).

IPCP-RESTART *n*
Sets the IPCP restart interval (retransmission timeout) to *n* seconds (default 3).

IPPARAM *string*
Provides an extra parameter to the IPUP and IPDOWN command file scripts. If used, the string supplied becomes the fifth parameter to those scripts.

IPUP *command-file*
Executes the specified command file when IP over PPP starts up, asynchronously, so that PPPD does not wait for the file's completion. Provided with the following command parameters:

Command Parameter	Description
P1	Interface name (such as PPP-0)
P2	Terminal device (such as _TTA3:)
P3	Local IP address
P4	Remote IP address
P5	(Optional) String specified with the IPPARAM option

If used with -D, debug output is logged in the *command-file*.LOG file.

IPDOWN *command-file*
Executes the specified file when IP over PPP shuts down, asynchronously, so that PPPD does not wait for the file's completion. Provided with the following command parameters:

Command Parameter	Description
P1	Interface name (such as PPP-0)
P2	Terminal device (such as _TTA3:)
P3	Local IP address
P4	Remote IP address

P5	(Optional) String specified with the `IPPARAM` option

If used with `-D`, debug output is logged in the *`command-file`*`.LOG` file.

KDEBUG *n*

Enables debugging of the low-level interface with the IP and terminal driver. The argument *n* is a number that is the sum of the following values:

Value	Description
1	Enable general debug messages
2	Request that the contents of received PPP packets be printed
4	Request that the contents of transmitted PPP packets be printed
8	Request that raw data be received from the serial device
16	Request that raw data be transmitted to the serial device

LCP-ECHO-FAILURE *n*

Presumes the peer is dead if *n* LCP echo-requests are sent without receiving a valid LCP echo-reply. If this happens, PPPD terminates the connection. Requires a non-zero value for the `LCP-ECHO-INTERVAL` parameter. Use this option to enable PPPD to terminate after the physical connection breaks (such as when the modem hangs up) in situations where no hardware modem control lines are available.

LCP-ECHO-INTERVAL *n*

Sends an LCP echo-request frame to the peer every *n* seconds. Normally the peer should respond to the echo-request by sending an echo-reply. You can use this option with the `LCP-ECHO-FAILURE` option to detect when the peer is no longer connected.

LCP-MAX-CONFIGURE *n*

Sets the maximum number of LCP configure-request transmissions to *n* (default 10).

LCP-MAX-FAILURE *n*
Sets the maximum number of LCP configure-NAKs returned before starting to send configure-Rejects instead to *n* (default 10).

LCP-MAX-TERMINATE *n*
Sets the maximum number of LCP terminate-request transmissions to *n* (default 3).

LCP-RESTART *n*
Sets the LCP restart interval (retransmission timeout) to *n* seconds (default 3).

LOGIN
Uses the system password database for authenticating the peer using PAP.

-MN
Disables magic number negotiation. With this option, PPPD cannot detect a looped-back line.

MRU *n*
Sets the MRU (Maximum Receive Unit) value to *n* for negotiation. The peer is requested to send packets of no more than *n* bytes. The minimum value is 128 and the default is 1500; 296 is recommended for slow links (40 bytes for the TCP/IP header plus 256 bytes of data).

-MRU
Disables MRU negotiation. PPPD uses the default MRU value of 1500 bytes.

MTU *n*
Sets the MTU (Maximum Transmit Unit) value to *n*. Unless the peer requests a smaller value through MRU negotiation, PPPD requests that the IP layer send data packets of no more than *n* bytes through the PPP network interface.

NAME *n*
Sets the name of the local system for authentication purposes to *n*.

NBDNS *address*
Identifies the primary NetBIOS name server.

NETMASK *n*
Sets the interface netmask to *n*, a 32-bit netmask in "decimal dot" notation (such as 255.255.252.0).

NOIPDEFAULT
Disables the default action when no local IP address is specified, which is to determine (if possible) the local IP address from the hostname. With this option, the peer must supply the local IP address during IPCP negotiation (unless you explicitly specify it on the command line, or in the TCPWARE:PPPDOPTIONS.DAT file).

+PAP
Requires the peer to authenticate itself using PAP.

-PAP
Disables authentication using PAP.

PAP-MAX-AUTHREQ *n*
Sets the maximum number of PAP authenticate-request transmissions to *n* (default 10).

PAP-RESTART *n*
Sets the PAP restart interval (retransmission timeout) to *n* seconds (default 3).

PAP-TIMEOUT *n*
Sets the maximum time that PPPD waits for the peer to authenticate itself with PAP to *n* seconds (0 means no limit).

PASSIVE
-P
Enables the "passive" option in the LCP. With this option, PPPD attempts to initiate a connection; if it does not receive a reply from the peer, it waits passively for a valid LCP packet from the peer (instead of exiting, as it does without this option).

-PC
Disables protocol field compression negotiation (the default).

PERSIST
Disables exiting after a connection is terminated. Instead, tries to reopen the connection.

PROXYARP
Adds an entry to this system's ARP (Address Resolution Protocol) table with the IP address of the peer and the Ethernet address of this system. This also starts the PPP interface as an unnumbered interface.

-PROXYARP
Disables the PROXYARP option. If you want to prevent users from creating proxy ARP entries with PPPD, place this option in the TCPWARE:PPPDOPTIONS.DAT file.

REMOTENAME *n*
Sets the assumed name of the remote system for authentication purposes to *n*.

SDNS *address*
Identifies the secondary DNS name server. If omitted, PPPD uses the first name server specified by the TCPWARE_NAMESERVERS logical, set up through Domain Name Services configuration.

SILENT
Disables transmitting LCP packets to initiate a connection until a valid LCP packet is received from the peer.

SNBDNS *address*
Identifies the secondary NetBIOS name server.

USEHOSTNAME
Enforces the use of the hostname as the name of the local system for authentication purposes (overrides the NAME option).

USER *u*
Sets the username to use to authenticate this machine with the peer using PAP to *u*.

-VJ
Disables negotiation of Van Jacobson style TCP/IP header compression (the default is to negotiate TCP/IP header compression).

-VJCCOMP
Disables the connection-ID compression option in Van Jacobson style TCP/IP header compression. With this option, PPPD does not omit the connection-ID byte from Van Jacobson compressed TCP/IP headers, nor request the peer to do so.

VJ-MAX-SLOTS *n*
Sets the number of connection slots to be used by the Van Jacobson TCP/IP header compression and decompression code to *n*, which must be between 2 and 16 (inclusive).

Troubleshooting PPPD

PPPD provides two types of debugging information:

- Trace output to SYS$OUTPUT
- OPCOM messages

By default, PPPD generates OPCOM messages for fatal errors, such as a failure to open the IP interface or insufficient privileges. In normal running operation, you should not see any OPCOM message. If you have a problem getting PPPD to work, first search for the OPCOM messages for PPPD.

You can also specify the DEBUG option. This enables the PPPD process to print out informational messages to SYS$OUTPUT. Define SYS$OUTPUT to the appropriate log file before invoking the PPPD server (or you can invoke PPPD interactively and output to the terminal). You must specify -DETACH to use this option.

When you specify the DEBUG (or -D) option, it debugs at level 5, which is to display up to warning and significant events. For more informational and debugging information, you can raise the debug level up to 7 by defining the logical TCPWARE_PPPD_DEBUG_LEVEL.

For a detached process, or if you prefer, you can also raise the message level for OPCOM messages. By default, it is set to 4 to report fatal and error messages. You may want to raise it to 5 to monitor the significant events in PPPD, or even higher for more detail by defining the logical TCPWARE_PPPD_OPCOM_LEVEL.

If you get the OPCOM messages:

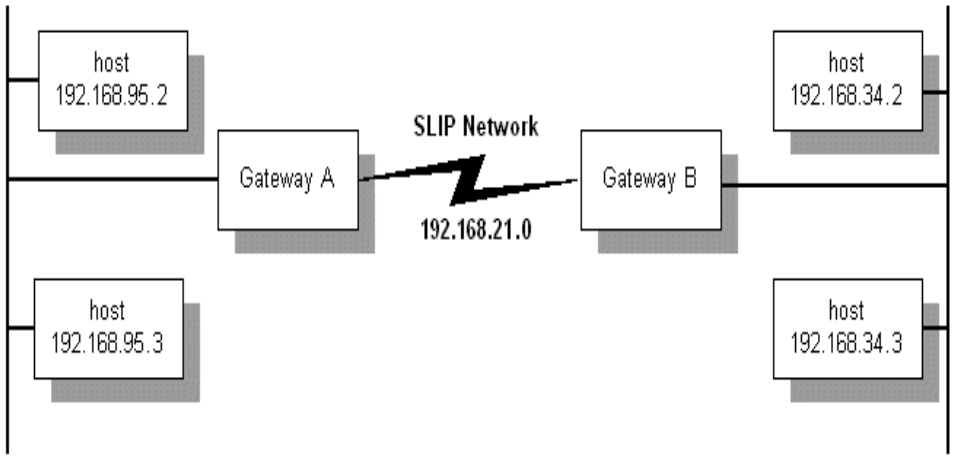
```
%TCPware_PPPD-E-setting terminal device failed with error 0x2C4
%TCPware_PPPD-E-PPP device initialization failed with error 0x2C4
%SYSTEM-F-DEVACTIVE, device is active,
```

Make sure that the *device-name* indicated on the PPPD command line is allocated to the current process before starting PPPD.

Serial Line IP Interface

Use serial Line IP (SLIP) when you need to route TCP/IP traffic over a serial line instead of an Ethernet cable. You most commonly use SLIP to connect systems on two Ethernet networks some distance apart.

You need TCP/IP and application software at each end of the SLIP link. One end can be TCPware, while the other end can be a SLIP implementation such as HP-UX or Linux. The below diagram shows a typical SLIP network.



Because SLIP lines are point-to-point connections, with TCPware you have the option to configure SLIP lines as network lines or as unnumbered interface lines.

TCPware supports both dedicated and dialup SLIP lines. Configure dedicated (hard-wired) SLIP lines during network configuration. Configure dialup SLIP lines as you need them, as described in this chapter. SLIP is common with line speeds from 1200 bits per second (bps) to 19.2 Kbps.

NETCP (not IPDRIVER) does the I/O to the terminal device to send and receive datagrams. NETCP uses the IPDRIVER External Interface to do this.

SLIP Line Identification

You can use any standard OpenVMS terminal device as a SLIP line. Unlike other line ID controller numbers, the SLIP line ID is not related to the actual device name. CNFNET prompts you for the actual device name during TCPware configuration.

The `START/IP` command *`line-specific-information`* parameter provides the OpenVMS device name for the SLIP line. If you omit this parameter, TCPware assumes that the `TCPWARE_SLIP_`*`n`* system logical (where *`n`* is the controller number) defines the device.

The maximum number of SLIP lines you can configure for one TCPware host is 256. You can define lines `SLIP-0` through `SLIP-255`. If you try to define a SLIP line with a larger number in CNFNET, the message `%TCPWARE_CNFNET-E-INVLINE, invalid line` appears.

Before Configuring SLIP Lines

Before you begin configuring SLIP lines:

- You must set the TTY_ALTYPAHD system parameter larger than its default value. This avoids losing characters. The higher the line speed, the higher you should set this parameter. For most applications, 1024 is appropriate.

  TTY_ALTYPAHD is not a dynamic parameter. If you use SYSGEN to change it, you must reboot the system for the change to take effect.

- The MAXBUF parameter must be at least twice the maximum transmission unit (MTU) of the SLIP line, plus 144. The default MTU for SLIP lines is 1006 bytes; therefore, MAXBUF must be at least 2156. Increase MAXBUF if necessary. MAXBUF is dynamic. If you use SYSGEN to change it, you do not need to reboot the system for the change in value to take effect.

Configuring SLIP Lines

To configure TCPware for SLIP:

1. If you plan to use a dedicated SLIP line, enter its line ID, host name, internet address, and terminal device name in response to the applicable prompts in CNFNET.

   You can use any valid OpenVMS terminal device as a SLIP line. CNFNET prompts you for the actual device name. Make sure that the network number portion of the SLIP line's internet address is unique if you use the /UNNUMBERED interface flag.

2. For dedicated SLIP lines, you may want to create the TCPWARE:SLIP_SETUP.COM file. The network startup command procedure (TCPWARE:STARTNET) executes this command procedure, if it exists, before using the SLIP lines.

   SLIP_SETUP.COM should contain the commands necessary to configure the terminal devices for proper operation. Typically, it would include SET TERMINAL commands to set the baud rate and other terminal characteristics.

3. For SLIP line speeds higher than 1200 bps, enable the alternate type-ahead buffer (ALTYPEAHD) characteristic for the terminal. Enter the following command for each SLIP terminal at the DCL prompt or in the SLIP_SETUP.COM file:

   ```
   SET TERMINAL /ALTYPEAHD /PERMANENT device
   ```

4. For both dedicated and dialup SLIP lines, set up routing information so that TCP/IP traffic routes properly over the SLIP link. The SLIP link should either have a unique network number or be unnumbered.

   You can give TCPware routing information either in the Network Control Utility (NETCU), by editing the TCPWARE:ROUTING.COM file to include appropriate NETCU commands, or

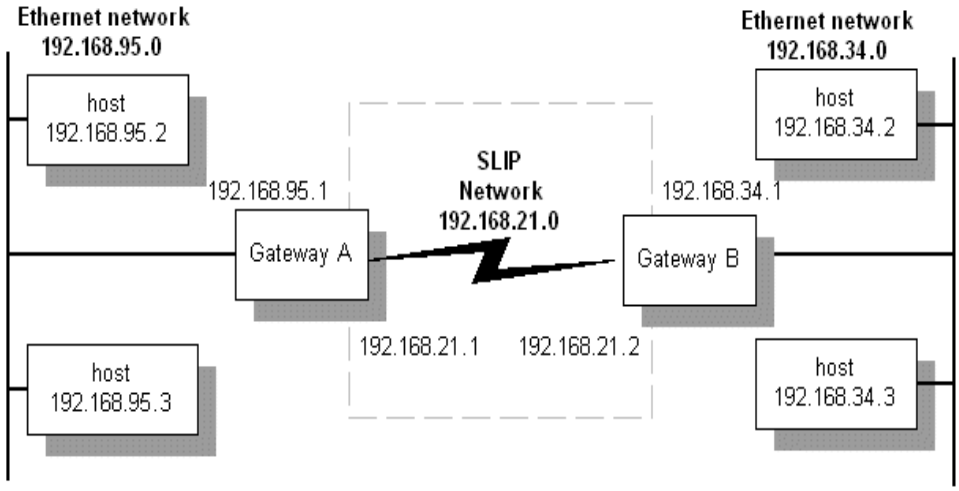
through GATED. For example, enter the following commands on each SLIP terminal at the DCL prompt or in the `SLIP_SETUP.COM` file (for dedicated lines):

```
ADD ROUTE /NETWORK network-address /GATEWAY gateway-address
ENABLE FORWARDING
```

NETCU entries remain active until TCPware shuts down. Updating the `ROUTING.COM` file with these commands makes them permanent. Do not use this method if using GateD to configure routes. To use GateD to configure routes, include a `static` statement for each of the routes in the `TCPWARE:GATED.CONF` file. (See the next section.)

Sample SLIP Link

The below diagram shows a sample internet consisting of three networks: Ethernet network 192.168.95.0, SLIP network 192.168.21.0, and Ethernet network 192.168.34.0.



Each gateway has an internet address for each network to which it connects. In this example, you can do the following to set up SLIP network 192.168.21.0 so that networks 192.168.95.0 and 192.168.34.0 can communicate:

- At each TCPware host in network 192.168.95.0 (on the Gateway A side), set the local gateway host address:

  ```
  SET GATEWAY 192.168.95.1
  ```

- Do the same for each host in network 192.168.34.0 (on the Gateway B side):

  ```
  SET GATEWAY 192.168.34.1
  ```

- At Gateway A, add the route through Gateway B's SLIP network address:

```
ADD ROUTE /NETWORK 192.168.34.0 /GATEWAY 192.168.21.2
ENABLE FORWARDING
```

- At Gateway B, add the route through Gateway A's SLIP address:

```
ADD ROUTE /NETWORK 192.168.95.0 /GATEWAY 192.168.21.1
ENABLE FORWARDING
```

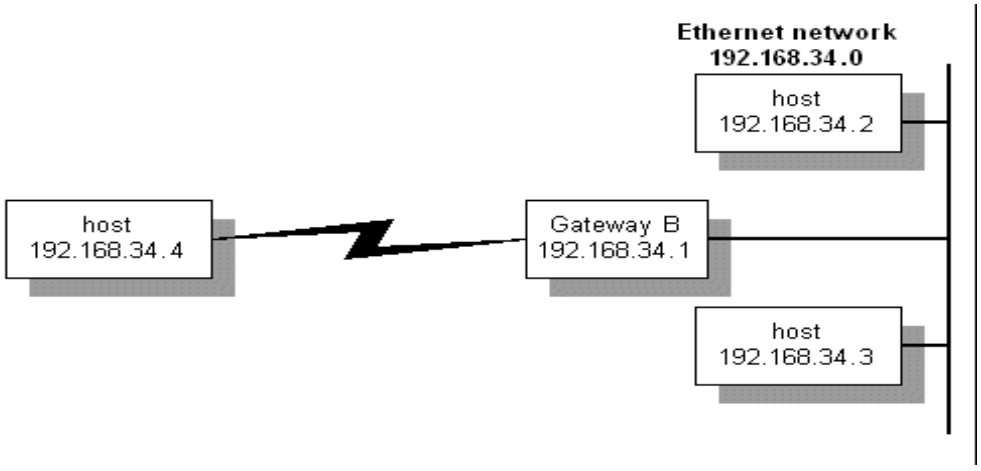
> **Note:** You can also define the default gateway by responding to prompts during the network configuration procedure. See Chapter 3, *Configuring the TCP/IP Core Environment*, in the TCPware *Installation & Configuration Guide*.

You can also configure the SLIP route using GateD. Include the following statements in the `GATED.CONF` files instead of the `ADD ROUTE` commands in `ROUTING.COM`:

```
static
{ 192.168.34.0 gateway 192.168.21.2 ;} ;
static
{ 192.168.95.0 gateway 192.168.21.1 ;} ;
```

Sample Unnumbered SLIP Link

In the case of a single host connection over a SLIP line to a network, you may not be able to dedicate a separate SLIP address to the connection on the network end. This requires you to use an unnumbered interface. The below diagram shows such a scenario.



Here is how you would set up the connection on host 192.168.34.4:

```
SET GATEWAY 192.168.34.1
```

Here is how you would set up the connection at Gateway B for host 192.168.34.2:

```
START/IP /UNNUMBERED SLIP-0 192.168.34.1
ADD ROUTE 192.168.34.4 SLIP-0
ENABLE FORWARDING /ARP
```

You need the `/UNNUMBERED` qualifier with the `START/IP` command. The `ENABLE FORWARDING` command with the `/ARP` qualifier lets host 192.168.34.1 respond to Address Resolution Protocol (ARP) requests for the remote host's address. In this way, other hosts on the 192.168.34.0 network can send any packets addressed to *`new-remote-ip-addr`* to Gateway B. This is known as "proxy ARP" and keeps you from having to add the host route on all other hosts in network 192.168.34.0.

You can also set up unnumbered interfaces on both ends of the SLIP connection. You must use `ENABLE FORWARDING`, but you cannot use proxy ARP with unnumbered interfaces, and you have to add routes on all other nodes in the network.

Incoming Dialup SLIP Lines

Perform these steps to set up an incoming dialup SLIP line:

1. Create a login account for the SLIP site using the OpenVMS `AUTHORIZE` utility. This should be a captive account and must have either OPER privilege or have been granted the `TCPWARE_CONTROL` rights identifier. `TCPWARE_CONTROL` requires the software password file, `PSW_*.DAT`, to have at least read access for the `TCPWARE_CONTROL` rights identifier.
2. Create a `LOGIN.COM` file for this account. A template file is available in `TCPWARE:SLIPLOGIN.COM`. The below example shows an unnumbered interface example.

In the example, a user is dialing in from a PC configured to use an address of 192.168.95.124. The Ethernet interface on the OpenVMS system is configured with an address of 192.168.95.12.

```
$ ON WARNING THEN LOGOUT
$ IF (F$TRNLNM("TCPWARE_NETCP_MBX") .EQS. "") THEN GOTO EXIT
$ NETCU := $TCPWARE:NETCU
$ TT = F$TRNLNM("TT")
$ DEFINE/USER SYS$ERROR NLA0:
$ DEFINE/USER SYS$OUTPUT NLA0:
$ NETCU START/IP/UNNUMBERED SLIP-1 192.168.95.12 'TT'
$ IF ($STATUS .NE. 1) THEN GOTO EXIT
$ NETCU ADD ROUTE 192.168.95.124 SLIP-1
$ NETCU ENABLE GATEWAY/ARP
$ EXIT:
$ LOGOUT
```

Once you set up the account and login file, the remote site simply dials up the OpenVMS system and logs in as the SLIP user to establish the connection. The login command file automatically configures the SLIP line.

Make sure to create a separate account and `LOGIN.COM` file for each remote SLIP site.

Outgoing Dialup SLIP Lines

To set up an outgoing dialup SLIP line:

1. Allocate the terminal device you wish to use. Enter: **ALLOCATE *terminal***
2. Set the terminal characteristics (such as the baud rate) using `SET TERMINAL` commands.
3. Use `SET HOST/DTE`, `KERMIT`, or some other utility to dial the remote system and log in as the SLIP user.
4. If the remote end successfully starts the SLIP line, exit `SET HOST/DTE` or `KERMIT`.
5. Start the outgoing SLIP line. Enter:

   ```
   $ NETCU START/IP SLIP-unit internet-address terminal
   ```

   - `SLIP-`*`unit`* - is the controller number of the SLIP line you want to assign (this number is for identification only and must be unique). You can use an asterisk (`*`) as a wildcard value, which assigns the lowest unused line ID to the SLIP interface (starting with `SLIP-0`), and also defines the `TCPWARE_LINE` (global) symbol to be that interface. (See the `START/IP` command in the *NETCU Command Reference*, Chapter 2, *NETCU Commands*.)
   - *`internet-address`* - is the internet address of the local host for the SLIP network.
   - *`terminal`* - is the terminal device name.

   You can also add any of the `NETCU START/IP` qualifiers supported for SLIP lines on the `NETCU START/IP` line.

   The following is a sample outgoing SLIP line startup command:

   ```
   NETCU START/IP SLIP-0 192.168.95.6 TXA7
   ```

6. Deallocate the terminal device. Enter: **DEALLOCATE terminal**

The SLIP line is now ready to use.

Disconnecting SLIP Lines

To disconnect a SLIP line, enter:

```
$ NETCU STOP/IP SLIP-unit
```

Note: TCPWare automatically removes the SLIP line from the network configuration if you configure the terminal device as a modem line with hang-up enabled, and you lose the phone line for any reason.

Full XON/XOFF Flow Control

The `/FLAGS=FLOWCONTROL` qualifier with the `START/IP` command configures the OpenVMS terminal device for full XON/XOFF control (`READSYNC`, `HOSTSYNC`, and `TTSYNC`). This means that you can use high speed modems that support compression and reliable data transfer modes. In addition, when OpenVMS terminal devices use full flow control, they do not need to use the alternate type-ahead buffer.

When you use `/FLAGS=FLOWCONTROL` qualifier with the `START/IP` command, TCPware configures the SLIP line to run a modified SLIP protocol. The modified SLIP protocol maps the characters shown below. Note that all numeric values are in octal.

Character name...	Has ASCII value...	With mapped character sequence...	
SLIP End of packet	300	333	334
SLIP Escape	333	333	335
XON	021	333	336
XON + 200	221	333	337
XOFF	023	333	340
XOFF + 200	223	333	341

RFC 1055, *A Nonstandard for Transmission of IP Datagrams over Serial Lines: SLIP*, defines the SLIP End-of-Packet and Escape characters, but does not define the XON/XOFF character mapping. TCPware uses this character mapping only when you specify `/FLAGS=FLOWCONTROL`.

> **Note:** Only use `/FLAGS=FLOWCONTROL` when the other end of the SLIP line connects to a system running TCPware, and if you configure it to use this option.

Qualifiers with SLIP Lines

The `START/IP` command supports a number of other qualifiers that you can use with SLIP lines. For details, see Chapter 2 in the *NETCU Command Reference*.

Compressed SLIP

Use compressed SLIP (CSLIP) to compress the TCP/IP headers only (and not the data) over the SLIP line.

You can set CSLIP options in NETCU for the serial line to either compress all TCP/IP headers or to compress them if it receives a compressed header from the peer. Use the `/FLAGS` qualifier for the `START/IP` command, as follows:

This Command...	Specifies that the serial line should compress...
`/FLAGS=COMPRESSED`	all TCP/IP headers.
`/FLAGS=AUTOENABLE`	TCP/IP headers only if the peer sends compressed TCP/IP headers.

Troubleshooting SLIP

Access error messages help by entering **`HELP TCPWARE MESSAGES [identifier]`**.

Also keep the following in mind:

- If you are not running TCPware on both ends of the SLIP connection, avoid using XON/XOFF flow control with SLIP. If you have a modem that uses XON/XOFF, disable that mechanism.

- If SLIP performance is poor, check that you configured the terminal to use the alternate type-ahead buffer (using **`SET TERMINAL /ALTYPEAHD /PERMANENT`**), and that you adjusted the `SYSGEN TTY_ALTYPAHD` parameter.

6. Cluster Alias Failover

Introduction

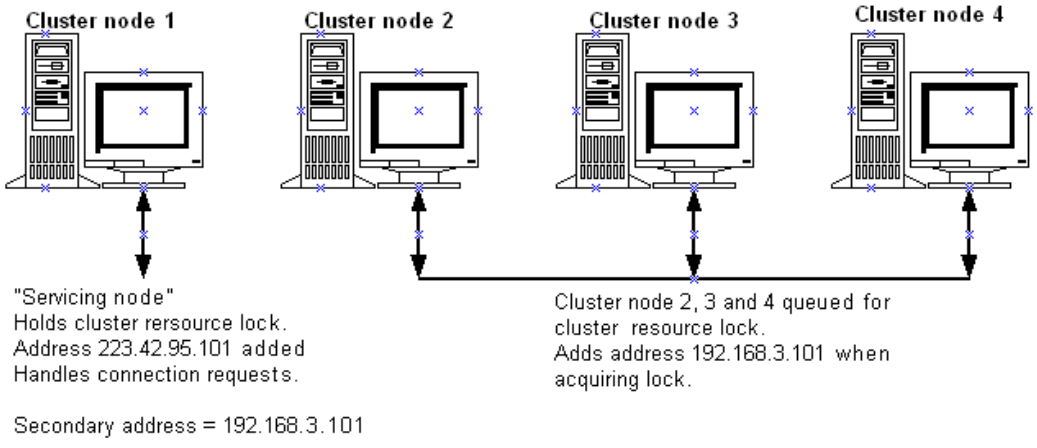
This chapter describes how to implement cluster alias failover for your VMS cluster.

Cluster alias failover allows you to set up an alias node to provide continued connection in case a system fails. The alias accepts incoming connection requests in a VMS cluster for a server if the servicing node goes down. Use cluster alias failover primarily for NFS over UDP, but you can also use it for other TCP/IP protocols such as FTP or TELNET.

For cluster alias failover to work properly, make sure that you have the same set of directories exported from each VMS cluster node. If it works effectively, cluster alias failover allows users to continue working productively if cluster nodes go down.

How It Works

Each VMS cluster node has a unique internet address. With cluster alias failover, you can assign a common secondary internet address, or alias. This alias is always an Internet address and never a name. This same secondary address is given to each VMS cluster node that handles connection requests and is recognized as a local address as shown below.



One of the nodes accepts the OpenVMS cluster-wide resource lock, adds the alias, and handles the incoming connection requests. Other VMS cluster nodes are also assigned the alias queue for the

resource lock. If the servicing node goes down (or you shut down TCPware on it), the system releases the resource lock. One of the queued VMS cluster nodes acquires the resource lock and adds the secondary address.

> **CAUTION!** Do not use cluster alias addresses as Domain Name Services addresses.

Setting It Up

Use the `ADD SECONDARY` command as described in the *NETCU Command Reference* to set up cluster failover. For example:

```
NETCU>ADD SECONDARY 192.168.3.101
```

The address 192.168.3.101 becomes the local alias address for the interface.

You can include the `/CLUSTER_LOCK` qualifier with the `ADD SECONDARY` command. This qualifier instructs the VMS cluster node to accept the OpenVMS cluster-wide resource lock before adding the secondary address. If another node in the VMS cluster holds the lock, the node queues for the lock and adds the address when it acquires it.

Use the `REMOVE SECONDARY` command as described in the *NETCU Command Reference* to remove an alias added through `ADD SECONDARY`. If the system holds a cluster lock, use the `/ABORT` qualifier to force removal of the secondary address.

Be sure to add the `ADD SECONDARY` command to the TCPware `ROUTING.COM` file so that it can take effect each time you start TCPware.

Limitations

There is no concept of a primary node with cluster alias failover. The alias address only moves to another address when the active servicing node goes down. The alias does not go back to the original servicing node when it comes back up.

You can move the alias address to a particular node by issuing the `REMOVE SECONDARY /ABORT` command in NETCU.

7. Managing SNMP Services

Introduction

This chapter explains the following Simple Network Management Protocol (SNMP) information:

- Links and traps
- Management Information Base (MIB)
- Configuring the SNMP Services
- Maintaining the SNMP configuration file
- Extendible MIB support
- SNMP Multiplexing (SMUX) peers support
- Agent X peers support
- Log file

SNMP Services allows network management stations to obtain timely information about the network activities of OpenVMS server hosts. The information describes such things as routing, line status, the volume of network traffic, and error conditions.

Links

In SNMP, network communication lines are called *links.* When counting the number of IP datagrams sent and received over most links, the SNMP agent returns the same numbers that are available through the `SHOW NETWORKS` command in TCPware's Network Control Utility (NETCU). These numbers indicate how many datagrams TCPware delivers.

Traps

A trap is an unsolicited message the SNMP agent sends to a management station to inform it that a change in the network occurred. The management station is responsible for diagnosing and monitoring any reported problems. For example, the SNMP agent sends traps to tell the management stations which communication lines are running and which are down.

The SNMP agent sends traps only to clients configured to receive traps, as defined in the SNMP agent configuration file (`SNMPD.CONF`). The SNMP agent supports all traps defined in the SNMP protocol, except EGP-Neighbor-Loss, Warm-Start, and Enterprise-Specific.

TCPware initially enables all supported traps. If for any reason you may want to disable them, you can do so by editing the SNMP agent configuration file. The changes take effect the next time you start the agent.

SNMP clients can enable or disable Authentication Failure Traps while the SNMP agent is running. These clients must have READ-WRITE community access, as described in the *MIB Access Rules* section.

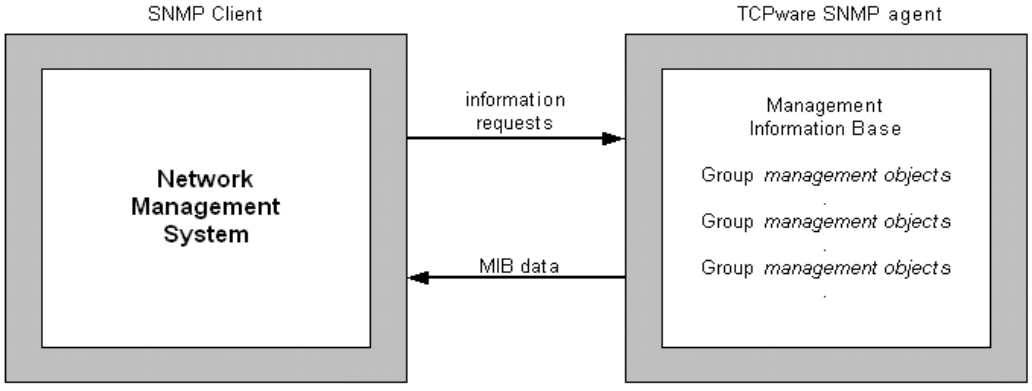
Management Information Base

A Management Information Base (MIB) is a collection of network management data residing on the SNMP agent host. The network management station reads and writes MIB data to the agent. Related types of data in the MIB are in groups. Each piece of data within a group is a *management object*.

All management objects in a MIB are coded in ASN.1. Any authorized clients can access data in the MIB by using the SNMP Get and GetNext requests.

MIB-II is the MIB version for TCP/IP implementations. The SNMP agent supports all management objects defined in MIB-II, except those in the External Gateway Protocol (EGP) Group.

The below diagram shows an SNMP client and agent exchanging MIB data.



MIB Access Rules

Two kinds of rules restrict access to the MIB:

- Community access profiles
- The access mode assigned to each management object – NONE, READ-ONLY, READ-WRITE, and WRITE-ONLY. The SNMP protocol standard determines the access mode.

The network administrator assigns each SNMP agent and client to at least one community. A community consists of SNMP agents and clients that have the same *access profile*, or collection of rules that determine whether community members can:

- Read or write MIB data
- Receive traps

You define access profiles in the SNMP agent configuration file.

Clients with READ-WRITE community access can alter the values of certain management objects in the MIB.

MIB Groups

The below table summarizes the information in each MIB group.

See also RFC 1213, *Management Information Base for Network Management of TCP/IP-based Internets: MIB-II*, for complete information on each MIB group.

Group	**Contains objects...**	**Which...**

System	`sysDescr`
	`sysUpTime`
	`sysContact`
	`sysName`
	`sysLocation`
	`sysServices`
		Provide information about the agent host, such as the domain name, geographic location, and the name of a contact person.

Interfaces	`ifNumber`
	`ifTable`
	`ifEntry`
	`ifIndex`
	`ifDescr`
	`ifType`
	`ifMtu`
	`ifSpeed`
	`ifPhysAddress`
	`ifAdminStatus`
	`ifOperStatus`
	`ifLastChange`
	`ifInOctets`
	`ifInUcastPkts`
		Provide generic information about each network interface, such as the speed, administrative status, and the maximum size of transmission units. Count the number of data errors, and the number of packets sent and received. Contain the Interfaces Table.

	ifInNUcastPkts ifInDiscards ifInErrors ifInUnknownProtos ifOutOctets ifOutUcastPkts ifOutNUcastPkts ifOutDiscards ifOutErrors ifOutQLen ifSpecific	
Address Translation (AT)	atTable atEntry atIfIndex atPhysAddress atNetAddress	Map the network (IP) address to the physical address.
IP	ipForwarding ipDefaultTTL ipInReceives ipInHdrErrors ipInAddrErrors ipForwDatagrams ipInUnknownProtos ipInDiscards ipInDelivers ipOutRequests ipOutDiscards ipOutNoRoutes ipReasmTimeout ipReasmReqds ipReasmOKs ipReasmFails ipFragOKs ipFragFails ipFragCreates ipAddrTable ipAddrEntry ipAdEntAddr ipAdEntIfIndex ipAdEntNetMask ipAdEntBcastAddr ipAdEntReasmMaxSize ipRouteTable ipRoute Entry ipRouteInfo ipRouteIfIndex ipRouteMetric1 ipRouteMetric2	Count the number of datagrams sent, received, in error, discarded, fragmented, and reassembled. Contain the IP Address Table, IP Routing Table, and IP Address Translation Table.

	ipRouteMetric3 ipRouteMetric4 ipRouteNextHop ipRouteType ipRouteProto ipRouteAge ipRouteMask ipRouteMetric5 ipRouteInfo ipNetToMediaTable ipNetToMediaEntry ipNetToMediaIFIndex ipNetToMediaPhyAddress ipNetToMediaNetAddress ipNetToMediaType ipRoutingDiscards	
ICMP	icmpInMsgs icmpInErrors icmpInDestUnreachs icmpInTimeExcds icmpInProbs icmpInSrchQuenchs icmpInRedirects icmpInEchos icmpInEchoReps icmpInTimestamps icmpInTimestampReps icmpInAddrMasks icmpInAddrMaskReps icmpOutMsgs icmpOutErrors icmpOutDestUnreachs icmpOutTimeExcds icmpOutParmProbs icmpOutSrcQuenchs icmpOutRedirects icmpOutEchos icmpOutEchoReps icmpOutTimestamps icmpOutTimestampReps icmpOutAddrMasks icmpOutAddrMaskReps	Count the number of ICMP messages sent, received, and in error. Also, count source quenches, redirects, and timestamps.
TCP	tcpRtoAlgorithm tcpRtoMin tcpRtoMax tcpMaxConn tcpActiveOpens tcpPassiveOpens	Count the number of active opens, passive opens, and failed attempts. Also, contain the TCP Connection Table.

	tcpAttemptFails tcpEstabResets tcpCurrEstab tcpInSegs tcpOutSegs tcpRetransSegs tcpConnTable tcpConnEntry tcpConnState tcpConnLocalAddress tcpConnLocalPort tcpConnRemAddress tcpConnRemPort tcpInErrs tcpOutRsts	
UDP	udpInDatagrams udpNoPorts udpInErrors udpOutDatagrams udpTable udpEntry udpLocalAddress udpLocalPort	Count the number of datagrams sent and received. Also, contain the UDP Listener Table.
SNMP	snmpInPkts snmpOutPkts snmpInBadVersions snmpInBadCommunityNames snmpInBadCommunityUses snmpInASNParseErrs snmpInTooBigs snmpInNoSuchNames snmpInBadValues snmpInReadOnlys snmpInGenErrs snmpInTotalReqVars snmpInTotalSetVars snmpInGetRequests snmpInGetNexts snmpInSetRequets snmpInGetResponses snmpInTraps snmpOutTooBigs snmpOutNoSuchNames snmpOutBadValues snmpOutGenErrs snmpOutGetRequests snmpOutGetNexts snmpOutSetRequests	Count the number of packets sent and received, invalid community names, and invalid version numbers, and SNMP errors. Also, count the number of requests, responses, and traps sent and received.

	snmpOutGetResponses snmpOutTraps snmpEnableAuthenTraps	

Configuring SNMP Services

To configure SNMP services, follow these steps:

1. Invoke the CNFNET procedure by entering the following command at the DCL prompt:

```
$ @TCPWARE:CNFNET SNMP
```

2. Edit the SNMP configuration file, as described in the next section.
3. Restart TCPware or SNMP.

Configuration File

The SNMP configuration file is `SNMPD.CONF`. The `TCPWARE_ROOT` directory includes this file.

The SNMP configuration 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.
- AgentX peer details
- SMUX peer details

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. Even though the `TRAPS` community will accept a mask, it is not currently used.

```
COMMUNITY OURNET 192.168.1.10 write !implied /32
COMMUNITY OURMGR 192.168.1.0/24 read
```

> **Note:** after editing the configuration, 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.

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 in your network (such as those running UNIX) may require information in a specific case.
- Place quotation marks (`" "`) around strings that contain spaces or that occupy more than one line in the file.
- Use a pound sign (`#`) or an exclamation point (`!`) to denote comments. SNMP ignores all information following these characters. It treats the pound sign and exclamation point like regular characters if they appear within quotation marks (`""`).

Values for MIB Objects

To define the values of several MIB objects in the SNMP configuration file, use the corresponding keywords listed in the table below.

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 you define each item.

SYSDESCR [*id-string*]

The *id-string* should include the full name of the hardware, operating system, and networking software. For example:

```
SYSDESCR "AlphaServer 8400, VMS V7.3, Process Software TCPwarefor OpenVMS"
```

If you omit the *id-string*, TCPware tries to obtain this information from your current system. If the attempt fails, the default is `System description is unknown.`

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@example.com"
```

The default is `System contact is unknown at this time.`

SYSLOCATION [*system-location*]

The *system-location* specifies the geographical location of the host. For example:

```
SYSLOCATION "959 Concord Street, Framingham, MA"
```

The default is: `System location is unknown at this time.`

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* specifies the manufacturer's name, product name, and hardware version for the interface. For example:

```
INTERFACE qna-1 10000000 "DELQA Ethernet Controller Version 1.0"
```

If you do not enter a description, TCPware tries to obtain one from your current system. If the attempt fails, the default is *xxxxxxxx*`,0,Unknown`.

SYSSERVICES *services-set-number*

The SNMP agent uses a default value of 72 for this MIB object. You can override this value in the configuration file. RFC 1213, *Management Information Base for Network Management of TCP/IP-based Internets: MIB-II*, explains how to calculate the value of `services-set-number`.

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*[*/mask*] *type*

`community-name`	Specifies the name of the community to which the client belongs. This parameter is case-sensitive.
`internet-address`	Specifies the client's internet address. If you enter 0.0.0.0, any address can use the community.
`mask`	Specifies the number of bits in the internet address that must match the specified address. The default value is 32.
`type`	Defines the access profile as one of the following: • `READ-ONLY` - The client can retrieve data from the MIB on this host • `READ-WRITE` - The client can retrieve data from and write data to the MIB on this host • `TRAPS` - The client will receive all enabled traps

`COMMUNITY public 0.0.0.0` is defined as `READ-ONLY` if no other communities are defined. The below example shows some community parameters defined in the configuration file.

```
community northeast  192.168.4.56  READ-ONLY
community northeast  192.168.220.1 READ-WRITE
community southwest  192.168.23.1  READ-WRITE
community southwest  192.168.23.1  TRAPS
```

- Client 192.168.4.56 in the `northeast` community has `READ-ONLY` access to the MIB, while client 192.168.220.1 in the same community has `READ-WRITE` access.

- Client 192.168.23.1 belongs to the `southwest` community. This community has `READ-WRITE` access to the MIB and trap information will be sent to this client.

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 below table shows how to disable 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 READ-WRITE community access.

Generating Traps

To generate an SNMP trap, define the symbol:

```
$ TRAP_GEN :== $TCPWARE: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, denoting a location in Process Software's portion of the MIB tree.
generic_trap	An integer representing the generic trap value.
specific_trap	An integer representing the specific trap value.
trap_specific_values	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 TCPWARE:SNMPD.CONF file to determine where to send the trap.

By default, traps are sent out of the first interface configured on the system. To use a different interface, use the hostid parameter in the configuration file:

HOSTID *ip-address*

```
HostId 192.168.1.5
```

The HOSTID parameter is used to specify the IP address to use when sending traps on a system with multiple interfaces. The IP address specified in the HOSTID statement is checked against the addresses configured on the system.

V2TRAPS

The SNMP agent sends SNMP v1 traps by default. To change to sending SNMP v2 traps by default include V2TRAPS in the configuration file.

Receiving Traps

TCPware also provides a program that will listen for traps and format them for display. In order to invoke this program, run TCPWARE:TRAP_LISTEN. It prompts for an optional file to log information to (default is the terminal) and the port number to listen on (default is 162).

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.

The SNMP server only accepts SMUX connections from peers listed by IP address in the `SNMPD.CONF` file.

To enable SMUX support, answer `Yes` to the appropriate question in `@TCPWARE:CNFNET SNMP`.

SMUX_PEER *ip-address [/mask*]

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. The mask specifies the number of bits in the internet address that must match the specified address. The default value is 32.

SNMP Agent Extensibility (AgentX) Peers

The AgentX 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 AgentX protocol to the subagent. The subagent passes the results of an SNMP query back to the SNMP agent.

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, answer `Yes` to the question "`Do you want to activate the SNMP AgentX service on this host?`" in `@TCPWARE:CNFNET SNMP`.

Then add `AGENTX_PEER` *`ip-address`* to the `SNMPD.CONF` file.

AGENTX_PEER *ip-address* [*/mask*]

The SNMP server only accepts AgentX connections from peers listed by IP address in the `SNMPD.CONF` file. Use the following syntax in the file:

```
AGENTX_PEER ip-address
```

If you are developing an AgentX subagent and need to debug the packets being exchanged with the SNMP Agent, then `define /system TCPWARE_SNMP_DEBUG 0%X40000` before starting SNMP.

Private MIB Application Program Interface

In addition to SMUX and AgentX, TCPware'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 TCPware'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 Chapter 10 of the *Programmer's Reference, SNMP Extensible Agent API Routines*.

SNMP Log File

When the SNMP agent starts up, it creates a log file called `TCPWARE: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.

Reloading the SNMP Configuration Without Rebooting

To reload the SNMP configuration:

```
$ @TCPWARE:RESTART SNMP
```

Performing SNMP Functions with TCPware

You can display SNMP information with the `NETCU SHOW SNMP MIB_VARIABLE` command. See the `SHOW SNMP` command in the TCPware *NETCU Command Reference* for information about this command.

Template Configuration File

SNMP services provides a `TEMPLATE_SNMPD.CONF` file in `TCPWARE_COMMON:[TCPWARE]` that you can use as a basis:

```
!
!       SNMP Agent (SNMPD) Configuration File (template)
!
! System description: sysdescr <id string>
! Typically the id string would include:
!       VAX cpu model (such as MicroVAX II, VAX 8650, etc)
!       VMS and version number
!       "Process Software, TCPware for OpenVMS Version 8.3"
!
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.
!
!interface una-0 10000000 "DEC DELUA Ethernet controller"
!
! Communities:
! community <community name><internet address><READ-ONLY|READ-
! WRITE|TRAPS>
!
community readers  1.2.3.4     READ-ONLY
community netman   223.95.45.3 READ-WRITE
community nettraps 223.49.45.3 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:
! SMUX_PEER <ip-address>
SMUX_PEER 1.2.5.4
SMUX_PEER 1.0.5.6
!
! Agent X Peers:
! AGENTX_PEER <ip-address>
AGENTX_PEER 127.0.0.1
AGENTX_PEER 192.168.1.1
```

8. X.25 Interface

CAUTION! This chapter used to document TCPware's X.25 interface for the VAX platform, which has been deprecated and is no longer available.

9. Routing and GateD

Introduction

This chapter describes TCPware's multiple gateway routing support, including how to set up routing and forwarding, and how to configure the Gateway Routing Daemon (GateD).

Multiple Gateway Support

All hosts and gateways on a network store routing information, usually including a list of default gateway addresses.

The TCPware routing table contains a list of default gateway addresses. TCPware always uses the first gateway address on the list unless it is marked as possibly being down. In this case, TCPware rotates the address of the gateway that is possibly down to the end of the list. TCPware then uses the next gateway address in the list, regardless of its state.

If all gateways are marked as being possibly down, TCPware uses all the addresses in rotation. This minimizes the number of datagrams sent to suspicious gateways and maintains stability when more than one gateway is available.

Router or Link Failure

When a router fails, the host detects that it is sending packets into a "black hole." The host detects this in approximately one minute. The host:

1. Marks that entry in the gateway address list as possibly being down.
2. Rotates that gateway address entry to the end of the list.
3. Uses the next gateway address, which is now the first entry in the list.

When a link fails, the router connected to that link redirects TCPware to use another router for that destination. TCPware does this using ICMP redirects.

Router or Link Recovery

When a router recovers, TCPware reverts to that router *only* if told to do so through a redirect for a specific destination. The acting router issues the redirect only if the original route has a better bandwidth, delay, and hop metric for the intended destination.

The system does not issue a redirect if the links between both routing paths are the same speed. In this case, TCPware continues to use the new router until:

- You reenter the gateway address using the Network Control Utility (NETCU).
- The new router fails.

When a link recovers, TCPware discards the dynamic route set by the ICMP redirect and switches back to the original router.

Static Routing

This section explains how to configure specific routes using Network Control Utility (NETCU) commands.

Routing Guidelines

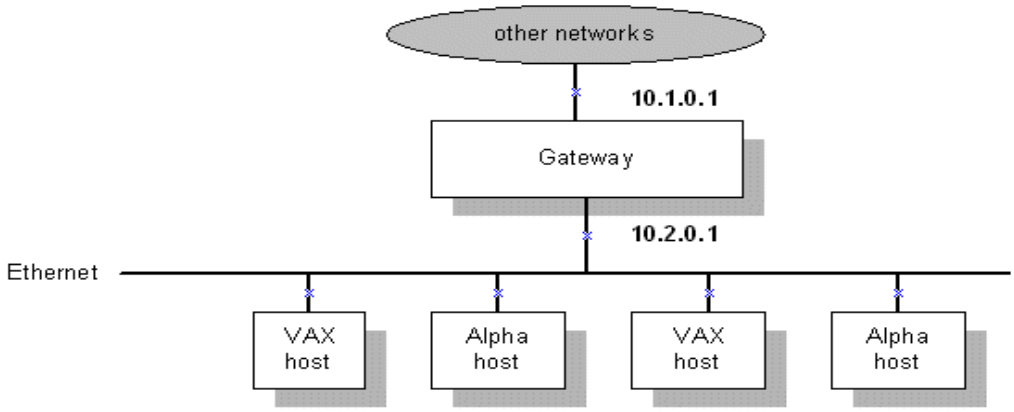
When setting up routing, consider the following guidelines:

- Most routes should be network routes rather than host routes. This prevents the routing table from becoming too large.
- Define a default gateway using the `NETCU SET GATEWAY` command (see the *NETCU Command Reference*). Use the default gateway when sending a datagram to a host that is not on a local network and for which no other route is known.
- You can set up routing so that TCPware executes your routing commands at startup. Enter the NETCU routing commands in the `TCPWARE:ROUTING.COM` file. CNFNET creates this file during network configuration (see the following sections).
- If using GateD to configure routes, use GateD exclusively. Do not combine GateD routing with static routing set up in NETCU, as with `ADD ROUTE`. Route settings in the `GATED.CONF` file may conflict with settings in the static `ROUTING.COM` file.

Example 1

The below diagram shows a local network connected to an internet through a gateway. Each host runs TCPware.

The gateway has an internet address for each network to which it connects.



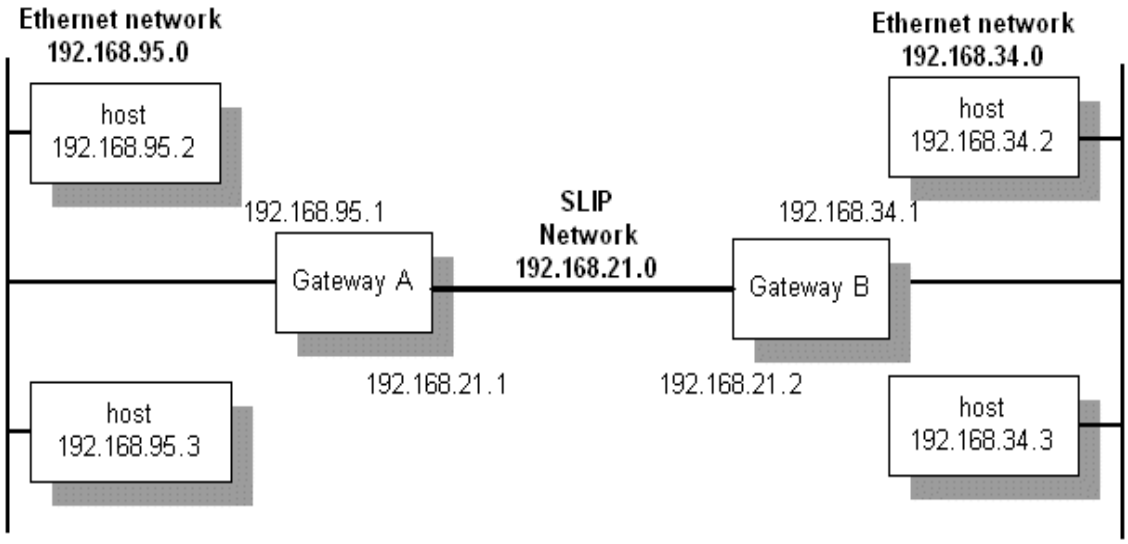
The easiest way to set up routing in this case is to define the gateway as the default gateway. To do this, perform one of the following tasks:

- Define the default gateway at each host by responding to prompts during TCPware's network configuration procedure (CNFNET).
- Enter the following NETCU command at the DCL prompt on each host:

```
$ NETCU SET GATEWAY 10.2.0.1
```

Example 2

The below diagram shows a sample internet consisting of three networks: Ethernet network 192.168.95.0, SLIP network 192.168.21.0, and Ethernet network 192.168.34.0.



Each gateway has an internet address for each network to which it connects. This is how the networks are set up:

- At each TCPware host in network 192.168.95.0, set the local gateway host address:

```
$ NETCU SET GATEWAY 192.168.95.1
```

- At each TCPware host on network 192.168.34.0, set the local gateway host address:

```
$ NETCU SET GATEWAY 192.168.34.1
```

- At Gateway A, add the route through Gateway B's SLIP network address:

```
$ NETCU ADD ROUTE 192.168.34.0 192.168.21.2 /NETWORK /GATEWAY
$ NETCU ENABLE FORWARDING
```

- At Gateway B, add the route through Gateway A's SLIP address:

```
$ NETCU ADD ROUTE 192.168.95.0 192.168.21.1 /NETWORK /GATEWAY
$ NETCU ENABLE FORWARDING
```

You can also define the default gateway by responding to prompts during the network configuration procedure (CNFNET). See Chapter 3, *Configuring the TCP/IP Core Environment*, in the *Installation & Configuration Guide*.

Forwarding

Forwarding, if enabled using `NETCU ENABLE FORWARDING`, allows IPDRIVER to route (forward) datagrams between the available networks as needed.

IPDRIVER routes datagrams between networks when you enable forwarding, and there is a known route to the datagram's destination internet address. TCPware allows fragmentation of the routed datagram.

IPDRIVER transmits an Internet Control Message Protocol (ICMP) redirect message to the source internet address of the datagram if it routes the datagram over the same source network interface.

If you enable forwarding and ARP mode, TCPware responds to ARP requests for any nonlocal internet address for which it has a defined route. This is proxy ARP. The following example shows enabling forwarding in ARP mode:

```
$ NETCU ENABLE FORWARDING/ARP
```

TCPware does not forward multicast datagrams.

Multicast Routing

When an application wants to send datagrams to a multicast internet address (Class D, 224.0.0.0 through 239.255.255.255) and the application does not specify a multicast interface, TCPware determines the interface as follows:

1. If the routing table has a host route for the multicast address, TCPware uses the host route.
2. If the routing table has a default multicast route (a network route for 224.0.0.0), TCPware uses the default multicast route.
3. If the routing table has a default route, TCPware uses the default route.
4. Otherwise, TCPware uses the first multicast-capable interface it finds.

Using GateD

The Gateway Routing Daemon (GateD) manages multiple routing protocols, including the Routing Information Protocol (RIP), Local Network Protocol (HELLO), Router Discovery Protocol, Open Shortest Path First (OSPF) protocol, Exterior Gateway Protocol (EGP), and Border Gateway Protocol (BGP).

Using GateD, the network administrator can control the flow of routing information through a configuration language. Once you start GateD, it makes routing decisions based on the data gathered by the routing protocols. If routing using GateD, use GateD exclusively.

> **Note:** If you want the system to function as a gateway, you must enable forwarding for it (using the `ENABLE FORWARDING` command in NETCU).

GateD allows you to control importing and exporting routing information by:

- Individual protocol
- Source and destination Autonomous System (AS)
- Source and destination interface
- Previous hop router
- Specific destination address

You can assign preference levels for different combinations of imported routing information by using a flexible masking capability. In TCPware, the name of the GateD process is `TCPware_GateD`.

GateD Configuration File

TCPware stores GateD configuration information in the `TCPWARE:GATED.CONF` file. You must create this file before you can use GateD. For details on GateD configuration, see *GateD Configuration Statements*.

GateD Route Selection

GateD determines the "best" route using preference values set for each protocol or peer. Each route has a single associated preference value, even though you can set preferences at many places in the `GATED.CONF` file. The last (or most specific) preference value is the one GateD uses. Some protocols have a secondary preference, sometimes called a "tie-breaker."

The factors GateD uses in determining "best" routes include:

- The route with the numerically smallest `preference` value is preferred.
- For two routes with equal preferences, the route with the numerically smallest `preference2` (the "tie-breaker") is preferred.
- A route learned from an interior gateway protocol is preferred over a route learned from an exterior gateway protocol. Least preferred is a route learned indirectly by an interior protocol from an exterior protocol.
- If Autonomous System (AS) path information is available, it helps determine the most preferred route:
    - A route with an AS path is preferred over one without an AS path.

    - If the AS paths and origins are identical, the route with the lower metric is preferred.
    - A route with an AS path origin of interior protocol is preferred over one with an origin of exterior protocol. Least preferred is an AS path with an unknown origin.
    - A route with a shorter AS path is preferred.
- If both routes are from the same protocol and AS, the one with the lower metric is preferred.
- The route with the lowest numeric next-hop address is used.

Preference values range from 0 to 255. The below table summarizes the default preference values for routes learned in various ways.

Default preference value	**Is defined by ... statement**
0	interface
10	ospf
20	gendefault (internally generated default)
30	redirect
40	kernel (routes learned using the socket route)
60	static
90	hello
100	rip
110	(point-to-point interfaces)
120	interfaces (routes to interfaces that are down)
130	aggregate/generate
150	ospf (AS external)
170	bgp

200	egp

Starting and Stopping GateD

After creating the `TCPWARE:GATED.CONF` file, you need to stop and restart GateD. Follow these steps:

1. Log in as the system manager.

2. Stop the GateD process by entering: **`@TCPWARE:SHUTNET GATED`**

3. Restart the GateD process by entering: **`@TCPWARE:STARTNET GATED`**

See the *Installation & Configuration Guide*, Chapter 6, *Starting and Testing TCPware*, for details on the `STARTNET.COM` and `SHUTNET.COM` command procedures.

GateD NETCU Commands

Use the NETCU commands in the below table to manage the GateD process. To use these commands, you need OPER or SYSPRV privilege. See the *NETCU Command Reference*, Chapter 2, *NETCU Commands*.

Command	**Description**
`CHECK GATED CONFIG`	Checks a GateD configuration file for syntax errors
`DUMP GATED STATE`	Dumps the state of the GateD process to a file
`LOAD GATED CONFIG`	Loads a GateD configuration file
`SET GATED TRACE`	Controls tracing in GateD
`SHOW GATED TRACE`	Shows tracing in GateD
`SHOW OSPF ADVERTISE`	Shows OSPF link state advertisements

SHOW OSPF AS	Shows the AS external database entries
SHOW OSPF DESTINATIONS	Shows the list of destinations and their indices
SHOW OSPF ERRORS	Shows the OSPF error log
SHOW OSPF HOPS	Shows the set of next hops for the OSPF router queried
SHOW OSPF INTERFACES	Shows all configured interfaces for OSPF
SHOW OSPF LOG	Shows the cumulative OSPF log of input/output statistics
SHOW OSPF NEIGHBORS	Shows all OSPF routing neighbors
SHOW OSPF ROUTING	Shows the OSPF routing table
SHOW OSPF STATE	Shows the link state database (except AS Externals)
SHOW RIP	Queries Routing Information Protocol (RIP) gateways
STOP/GATED	Stops the GateD process
TOGGLE GATED TRACING	Toggles tracing in GateD
UPDATE GATED INTERFACES	Rescans the GateD network interfaces

GateD Configuration Statements

The GateD configuration file is TCPWARE:GATED.CONF. This file must be present for the GateD process to run. The structure of the GateD configuration language is similar to C. The configuration file consists of statements terminated by a semicolon (;). Statements consist of tokens separated by a space. This structure simplifies identification of the associated parts of the configuration.

You can include comment lines either by beginning them with a pound sign (#) or delimiting them with slash asterisk (/*) and asterisk slash (*/). The configuration file consists of the following sections, which reflect the order in which the statements, if used, must appear:

Directives	(%directory, %include)
Statements	traceoptions options interfaces
Definitions	autonomous-system routeid martians
Protocols	rip help redirect router-discovery server/client bgp ospf
Static routes	static
Control	import export aggregate generate

Directives

Directive statements include:

```
%directory
%include
```

Directive statements provide special instructions to the parser. They do not relate to the protocol configuration and can occur anywhere in `GATED.CONF`. They also end in a new line instead of a semicolon (;) like the other statements.

Format

%directory "*directory*"

Defines the directory where the include files go if you do not fully specify directory as part of the filename in the `%include` statement. Does not actually change the current directory, but simply applies the directory prefix.

%include "*filename*"

Identifies an include file. GateD includes the contents of the file in `GATED.CONF` at the point where the `%include` appears. If you do not fully specify the filename, it is relative to the directory defined in `%directory`. The `%include` directive causes GateD to parse the specified file completely before resuming. You can nest up to ten levels of include files.

---

traceoptions

The `traceoptions` statement controls tracing options. You can configure GateD tracing options at many levels. These include file specifications, control options, and global and protocol-specific tracing options.

Lower levels of statements inherit tracing options from the next higher level, unless overridden.

Format

```
traceoptions [ "tracefile" [replace] [size size[k | m] files files]]
                      [nostamp] traceoptions [except traceoptions] | none ;
```

Options and Parameters

"*tracefile*"

File to receive tracing information. If this filename is not fully specified, GateD creates it in the directory where you started GateD.

replace

Replaces an existing file. The default is to append to an existing file.

size *size[*k | m*]* files *files*

Limits the maximum size, in `k` or `m` or the files indicated, of the trace file (the minimum is 10k). When the file reaches `size`, GateD creates a new version.

nostamp

Control option which means not to prepend a timestamp to all trace lines. The default is to prepend a timestamp.

traceoptions

Specific to each protocol statement. Note that these global options may not apply to all protocols.

except *traceoptions*

Disables more specific trace options after enabling broader ones.

`none`
Turns off all tracing for the protocol or peer.

Option	Description
`adv`	For debugging: traces the allocation and freeing of policy blocks.
`all`	Turns on the `general`, `normal`, `policy`, `route`, `state`, `task`, and `timer` options.
`general`	Shorthand for specifying both the `normal` and `route` options.
`iflist`	Traces reading of the kernel interface. Useful to specify this with the `-t` option on the command line since the first interface scan occurs before reading the configuration file.
`normal`	Traces normal protocol occurrences (abnormal protocol occurrences are always traced).
`parse`	For debugging: traces the lexical analyzer and parser.
`policy`	Traces how protocol and user-specified policy apply to routes imported and exported.
`route`	Traces routing table changes for routes installed by the protocol or peer.
`state`	Traces state machine transitions in the protocols.
`symbols`	Traces symbols read from the kernel at startup. The only useful way to specify this level of tracing is to use the `-t` option on the command line, since GateD reads the symbols from the kernel before parsing the configuration file.
`task`	Traces system interface and processing associated with the protocol or peer.
`timer`	Traces timer usage by the protocol or peer.

options

The options statements let you specify some global options. If used, options must appear before any other type of configuration statement in `GATED.CONF`.

Format

```
options [nosend]
        [noresolv]
        [gendefault [preference value][gateway host] ]
        [syslog [upto] loglevel]
        [mark time] ;
```

Options and Parameters

`nosend`

Does not send packets. Makes it possible to run GateD on a live network to test protocol interactions, without actually participating in the routing protocols. You can examine the packet traces in the GateD log to verify that GateD functions properly. Most useful for RIP and HELLO. Does not yet apply to BGP, and not useful with EGP and OSPF.

`noresolv`

Does not resolve symbolic names into IP addresses. By default, GateD uses the `gethostbyname()` and `getnetbyname()` library calls that usually use the Domain Name System (DNS) instead of the host's local host and network tables. If there is insufficient routing information to send DNS queries, GateD deadlocks during startup. Use this option to prevent these calls.

> **Note:** When you use this option, symbolic names cause configuration file errors.

`gendefault [preference value] [gateway host]`
`nogendefault`

Creates a default route with the special protocol default when a BGP or EGP neighbor is up. You can disable this for each BGP/EGP group with the `nogendefault` option. By default, this route has a `preference` value of `20`. This route is normally not installed in the kernel forwarding table; it is only

present for announcement to other protocols. The `gateway` option installs the default route in the kernel forwarding table with a next hop of the gateway defined.

> **Note:** Using more general options is preferred to using `gendefault`. (See *aggregate* for details on the `generate` statement.)

`syslog [upto] loglevel`
Amount of data GateD logs to OPCOM. OpenVMS systems map UNIX `syslog` logging levels to OPCOM severity levels. The default is `syslog upto info`. The mapping of `syslog` to OPCOM logging levels appears in *Mapping of UNIX syslog Levels to OpenVMS OPCOM Severity Levels*.

`mark time`
GateD sends a message to the trace log at the specified `time` interval. Can be one method of determining if GateD is still running.

syslog log level	Is equivalent to OPCOM level...
`emerg`	FATAL
`alert`	FATAL
`crit`	FATAL
`err`	ERROR
`warning`	WARNING
`notice`	INFORMATIONAL
`info` (default)	INFORMATIONAL
`debug`	INFORMATIONAL

Example

```
#  generate a default route when peering with an EGP or BGP neighbor:
#
options gendefault ;
```

interfaces

An interface is the connection between a router and one of its attached networks. Specify a physical interface by interface name, IP address, or domain name. Multiple reference levels in the configuration language let you identify interfaces using wildcards (only the device driver part of the name, to match any unit number), interface type names, or addresses.

Format

```
Interfaces
{
  options
       [strictinterfaces]
       [scaninterval time] ;
  interface list
       [preference value]
       [down preference value]
       [passive]
       [simplex]
  define address
       [broadcast address] | [pointtopoint address]
       [netmask mask]
       [multicast] ;
};
```

Options Clause

```
options
       [strictinterfaces]
       [scaninterval time] ;
```

strictinterfaces

Makes it a fatal error to use reference interfaces not present when you start GateD or that are not part of the `define` parameter. Normally, GateD issues a warning message and continues.

scaninterval *time*

Sets how often GateD scans the kernel interface list for changes. The default is every 15 seconds on most systems, and 60 seconds on systems that pass interface status changes through the routing socket (such as BSD 4.4).

Interface Clause

Sets interface options on the specified interfaces. A *list* can consist of interface names, domain names, numeric addresses, or the value `all`. Include one or more interface names, including wildcard names (without a number) and those that can specify more than one interface or address.

There are three ways to reference an interface:

By wildcard	Only the device driver part of the name, to match any unit number.
By name	Combined device driver and unit number of an interface.
By address	IP address or domain name (if resolving to one address only).

There are four types of interfaces allowed:

Loopback	Must have the address 127.0.0.1. Packets from a loopback interface go back to the originator. Also used for reject and blackhole routes (**not supported in TCPware**). The interface ignores any net mask. It is useful to assign an additional address to the loopback interface that is the same as the OSPF or BGP router ID; this allows routing to a system based on router ID that works if some interfaces are down.
Broadcast	Multiaccess interface capable of physical level broadcast, such as Ethernet, Token-Ring, and FDDI. A broadcast interface has an associated subnet mask and broadcast address. The interface route to a broadcast network is a route to the complete subnet.
Point-to-point	Tunnel to another host, usually on some sort of serial link. A point-to-point interface has a local address and a remote address. The remote address must be unique among the interface addresses on a given router. Many point-to-point interfaces and up to one non point-to-point interface must share the local address. This conserves subnets as you do not need any when using this technique. If you use a subnet mask on a point-to-point interface, only RIP version 1 and HELLO use it to determine which subnets propagate to the router on the other side of the point-to-point interface.

Nonbroadcast multiaccess (NBMA)	Multiaccess but not capable of broadcast, such as frame relay and X.25. This type of interface has a local address and a subnet mask.

preference *value*
Sets the preference for routes to this interface when it is up and GateD determines it to function properly. The default preference `value` is `0`. While the `preference` statement is optional, it is strongly recommended that you set an explicit preference value if you do use it.

down preference *value*
Sets the preference for routes to this interface when GateD determines that it does not function properly, but the kernel does not indicate that it is down. The default down preference `value` is `120`.

passive
Does not change the preference of the route to the interface if determined not to function properly from lack of routing information. GateD checks this only if the interface actively participates in a routing protocol.

simplex
The interface does not recognize its own broadcast packets. Some systems define an interface as simplex with the `IFF_SIMPLEX` flag. On others, the configuration defines it. On simplex interfaces, packets from the local host are assumed to have been looped back in software and are not used to indicate that the interface functions properly.

Define Clause

```
Interfaces
{
  define address
    [broadcast address] | [pointtopoint address]
    [netmask mask]
    [multicast] ;
} ;
```

Defines interfaces not present when starting GateD so that the configuration file can reference them when using `options strictinterfaces`.

broadcast *address*
Makes the interface broadcast-capable (for Ethernet or Token-Ring) and specifies the broadcast address.

pointtopoint *address*
Makes the interface point-to-point (such as SLIP or PPP) and specifies the address on the local side of the interface. The first address in the `define` statement references the host on the remote end of the interface.

An interface not defined as `broadcast` or `pointtopoint` must be nonbroadcast multiaccess (NBMA), such as for an X.25 network.

netmask *mask*
Subnet mask to use on the interface. Ignored on point-to-point interfaces.

multicast
Makes the interface multicast-capable.

Examples

1. This example sets the interface as passive.

```
#  do not mark interface 192.168.95.41 as down,
#  even if there is no traffic:
#
interfaces{
  interface 192.168.95.41 passive ;
} ;
```

2. This example shows the interface statements used with the **rip** statement (see the `rip` description). Users would receive RIP packets only from interfaces `sva-0` and `sva-1`, but not from `fza-0`, and `sva-1` would be the only one that could send them.

```
rip yes {
  interface all noripin noripout ;
  interface sva ripin
;
```

```
  interface sva-1 ripout ;
  } ;
```

Definition Statements

Definition statements include:

```
autonomoussystem
routerid
martians
```

Definition statements are general configuration statements that relate to all of GateD or at least to more than one protocol. You must use these statements for any protocol statements in the configuration file.

Format

autonomoussystem *ASnumber* [loops *number*];

An autonomous system (AS) is a set of routers under a single technical administration, using an internal protocol and common metrics to route packets within the AS, and an external protocol to route packets to other ASs. The Network Information Center (NIC) assigns AS numbers.

The `autonomoussystem` statement sets the AS number of the router. You require this option if using BGP or EGP. The `loops` option is only for protocols supporting AS paths, such as BGP. It controls the number of times this AS can appear in an AS path, and defaults to `1`.

routerid *host* ;

A router ID is an IP address used as a unique identifier assigned to represent a specific router, usually the address of an attached interface. The `routerid` statement sets the router ID for the BGP and OSPF protocols. The default is the address of the first interface GateD encounters. The address of a non-point-to-point interface is preferred over the local address of a point-to-point interface, and an address on a loopback interface that is not the loopback address (127.0.0.1) is most preferred.

```
martians
{
  host host [allow] ;
  network [allow] ;
  network mask mask [allow] ;
  network masklen number [allow] ;
  default [allow] ;
} ;
```

The `martians` statement defines a list of invalid addresses, called *martians*, that the routing software ignores. Sometimes a misconfigured system sends out obviously invalid destination addresses. The

statement allows additions to the list of `martian` addresses. (See *Route Filtering* for details on specifying ranges.)

You can also use the `allow` parameter to explicitly allow a subset of an otherwise disallowed range.

Example

This example shows the use of all three definition statements, `autonomoussystem`, `routerid`, and `martians`.

```
#  use AS number 249:
#
autonomoussystem 249 ;
#
#  set the router
ID number:
#
routerid 192.168.95.41 ;
#
#  prevent routes to
0.0.0.26 from ever being accepted:
#
martians {
host 0.0.0.26 ;
};
```

Route Filtering

You can filter routes by matching a certain set of routes by destination, or by destination and mask. Use route filters on `martians`, `import`, and `export` statements.

The action taken when no match is found depends on the context. For example, import and export route filters assume an **`all`** `reject ;` at the end of a list. A route matches the most specific filter that applies. Specifying more than one filter with the same destination, mask, and modifiers generates an error.

Format

network *[*exact | refines | allow*]*
network mask *mask* *[*exact | refines*]*
network masklen *number* *[*exact | refines*]*
all
default
host *host*

Options and Parameters

network

Destination network IP address. You can use one of the following options:

exact	Destination mask must match the supplied mask exactly. Used to match a network, but no subnets or hosts of that network.
refines	Destination mask must be more specified (longer) than the filter mask. Used to match subnets or hosts of a network, but not the network.
allow	See the `martians` definition statement.

mask *mask*

Destination network mask.

masklen *number*

Length of the destination network mask.

all
Entry matches anything. Equivalent to `0.0.0.0 mask 0.0.0.0`.

default
Matches the default route. To match, the address must be the default address and the mask must be all zeros. Equivalent to `0.0.0.0 mask 0.0.0.0 exact`. (Not valid for `martians` statements.)

host *host*
Matches the specific host. To match, the address must match exactly the specified host, and the network mask must be a host mask (all 1s). Equivalent to `host mask 255.255.255 exact`. (Not valid for `martians` statements.)

---

rip

GateD supports the Routing Information Protocol (RIP). RIP is a distance-vector protocol for distributing routing information at the local network level of the Internet. In distance-vector routing, each router transmits destination addresses and costs to its neighbors (computers communicating over RIP).

RIP versions 1 and 2 are the most commonly used interior protocol. RIP selects the route with the lowest metric as the best route. The metric is a hop count representing the number of gateways through which data must pass to reach its destination. The longest path that RIP accepts is 15 hops. If the metric is greater than 15, a destination is considered unreachable and GateD discards the route. RIP assumes the best route uses the fewest gateways, that is, the shortest path, not taking into account congestion or delay along the way.

RIP uses two types of packets: requests and responses.

Requests. A request asks for information about specific destinations or for all destinations. RIP can send requests when a router:

- Comes up
- Receives timed-out information about a destination

If a request fails to specify a destination, RIP assumes the router requests information about all destinations.

Responses. Responses contain destination and cost pairs. RIP sends responses under the following three conditions:

- In response to a request
- When information changes; for example, cost information
- At set intervals; for example, reporting the destination to each neighbor every 30 seconds

RIP discards the destination and cost information if a neighbor fails to report the distance to a destination after a certain time interval.

RIP IP Addresses. RIP version 1 contains no provision for passing around a mask. RIP infers the mask based on whether the address is class A, B, or C. Sometimes there are special cases when the inferred mask differs from class A, B, or C. For example:

- When you use RIP with a subnet (in this case the routers must know the subnet mask for a particular network number)
- When the system updates RIP with an address reported as 0.0.0.0, RIP considers this address as a default destination with a mask of 0.0.0.0

- When the system updates RIP with bits set in the host portion of the address, RIP assumes the address refers to a host with a mask of 255.255.255.255

With RIP version 2, you can specify the network mask with each network in a packet.

Configuring RIP. You configure RIP in the `GATED.CONF` file using a GateD protocol statement that enables or disables RIP. The syntax of the `rip` statement is as follows, with the parameters described next:

Format

```
rip  yes | no | on | off
[{
     [no]broadcast ;
     nocheckzero ;
     preference value ;
     defaultmetric metric ;
     query authentication [ none | [ [simple | md5] password ]];
     interface list
       [[no]ripin ] [ [no]ripout ]
       [metricin metric]
       [metricout metric] ;
       [version 1] | [ version 2 [multicast | broadcast] ]
       [ [secondary] authentication [none | [[simple | md5] password ]]];
     trustedgateways list ;
     sourcegateways list ;
     traceoptions options ;
}] ;
```

Options and Parameters

yes | on (default)
no | off
When enabled on a host, RIP listens in the background to routing updates. When enabled on a gateway, RIP supplies routing updates. Enabled by default.

broadcast ;
Broadcasts RIP packets regardless of the number of interfaces present. Useful when propagating static routes or routes learned from another protocol into RIP. In some cases, using **broadcast** when only one network interface is present can cause data packets to traverse a single network twice. The default for more than one interface.

nobroadcast ;
Does not broadcast RIP packets on attached interfaces even if there is more than one. If you use the `sourcegateways` parameter, routes are still unicast directly to that gateway. The default for a single interface.

nocheckzero ;
Does not make sure that reserved fields in incoming RIP version 1 packets are zero. Normally RIP rejects packets whose reserved fields are zero.

preference *value* ;
Sets the preference for routes learned from RIP. A preference specified in import policy can override this. The default preference `value` is **100**.

defaultmetric *metric* ;
Metric used when advertising routes learned from other protocols. Choice of values requires that you explicitly specify a metric in order to export routes from other protocols into RIP. A metric specified in export policy can override this. The default *`metric`* is `16`.

query authentication ;
Authentication required of query packets that do not originate from routers. The default is `none`.

Interface Clause

```
rip  yes | no | on | off
[{
    [no]broadcast ;
    nocheckzero ;
    preference value ;
    defaultmetric metric ;
    query authentication [ none | [ [simple | md5] password ] ] ;
    interface list
      [ [no]ripin ] [ [no]ripout ]
      [metricin metric]
      [metricout metric] ;
      [version 1] | [ version 2 [multicast | broadcast] ]
      [ [secondary] authentication [none | [ [simple | md5] password] ] ] ;
      trustedgateways list ;
      sourcegateways list ;
      traceoptions options ;
}] ;
```

Controls various attributes of sending RIP on specific interfaces. (See the interfaces statement for a description of `list`.) Note that if there are multiple interfaces configured on the same subnet, only the first one on which RIP output is configured sends the RIP updates. This limitation is required because of the way the UNIX kernel operates. A future GateD release will hopefully remove this limitation. The default `list` value is `all`.

`ripin` (default)
`noripin`
Use `ripin` explicitly when using `noripin` on a wildcard interface descriptor. The `noripin` option ignores RIP packets received over the specified interfaces.

`ripout` (default)
`noripout`
Use `ripin` explicitly when using `noripout` on a wildcard interface descriptor. The `noripin` does not send RIP packets over the specified interfaces.

`metricin` ***`metric`***
RIP metric to add to incoming routes before they are installed in the routing table. Makes the router prefer RIP routes learned using the specified interfaces less than those learned from other interfaces. The default is the kernel interface metric plus 1. If using this as the absolute value, the kernel metric is not added.

metricout *metric*
RIP metric to add to routes sent over the specified interface(s). Makes other routers prefer other sources of RIP routes over this router. The default `metric` value is 0.

version 1 (default)
Sends RIP version 1 packets over the specified interface(s).

version 2 *[*multicast | broadcast*]*
Sends RIP version 2 packets over the specified interfaces. If IP multicasting support is available on this interface, the default is to send full version 2 packets. If multicasting is not available, version 1 compatible version 2 packets are sent. Options include:

`multicast`	Multicasts RIP version 2 packets over this interface. (Default)
`broadcast`	Broadcasts RIP version 1 compatible version 2 packets over this interface even if IP multicasting is available

[*secondary*]* authentication *[*none | *[[*simple | md5*]* password*]]
Authentication type to use. Applies only to RIP version 2 and is ignored for RIP-1 packets. If you specify a *password*, the authentication type defaults to **`simple`**. The password should be a quoted string with 0 to 16 characters. If you specify **`secondary`**, this defines the secondary authentication. The default is `authentication none`.

trustedgateways *list*
List of gateways from which RIP accepts updates (host names or IP addresses). If used, only updates from the gateways in the list are accepted. The default `list` value is `all`.

sourcegateways *list*
List of routers to which RIP sends packets directly, not through multicasting or broadcasting. If used, only updates from the gateways in the list are accepted. The default `list` value is `all`.

traceoptions *options*

RIP-specific trace options:

`packets`	All RIP packets, or packets `[detail] send` or `[detail] recv` (`detail` provides a more verbose format to provide more details; if used, `detail` must come before `send` or `recv`)
`request`	RIP information request packets, such as `REQUEST`, `POLL` and `POLLENTRY`
`response`	RIP `RESPONSE` packets that actually contain routing information

hello

GateD supports the HELLO protocol. HELLO is an interior protocol that uses delay as the deciding factor when selecting the best route. Delay is the round trip time between source and destination. HELLO is not as widely used as when it was the interior protocol of the original 56-Kb/sec NSFNET backbone and used between LSI-11 ("fuzzball") routers. Because of this, HELLO is disabled by default.

By default, HELLO, like RIP, uses the kernel interface metric set by the `ifconfig` command to influence metrics added to routes as they are installed in the routing table (`metricin`). Since the kernel interface metric is in hops, it must be translated into HELLO's millisecond metric. For the translation scheme, see the HELLO Hops-to-Metrics Translation table below.

This many Hops	Translate to this HELLO metric	This many Hops	Translate to this HELLO metric	This many Hops	Translate to this HELLO metric
0	0	6	713	12	75522
1	100	7	1057	13	11190
2	148	8	1567	14	16579
3	219	9	2322	15	24564
4	325	10	3440	16	3000
5	481	11	5097		

You configure HELLO in the GATED.CONF file using a GateD protocol statement that enables or disables HELLO.

When enabled, HELLO assumes `nobroadcast` when only one interface exists. HELLO assumes broadcast when more than one interface exists.

Format

```
hello  yes | no | on | off
[{
```

```
    [no]broadcast ;
    preference value ;
    defaultmetric metric ;
    interface list
      [ [no]helloin ]
      [ [no]helloout ]
      [metricin metric]
      [metricout metric] ;
      trustedgateways list ;
      sourcegateways list ;
      traceoptions options ;
}] ;
```

Options and Parameters

yes | on or no | off (default)
When enabled on a host, HELLO listens in the background for routing updates. When enabled on a gateway, HELLO supplies routing updates. Disabled by default.

broadcast ;
nobroadcast ;
The `broadcast` option broadcasts HELLO packets regardless of the number of interfaces present. Useful when propagating static routes or routes learned from another protocol into HELLO. In some cases, using `broadcast` when only one network interface is present can cause data packets to traverse a single network twice. The default for more than one interface.

The `nobroadcast` option does not broadcast HELLO packets on attached interfaces, even if there is more than one. If you use the `sourcegateways` parameter, routes are still unicast directly to that gateway. The default for a single interface.

preference *value* ;
Preference for routes learned from HELLO. A preference specified in import policy can override this. The default preference `value` is `90`.

defaultmetric *metric* ;
Metric used when advertising routes learned from other protocols. Requires you to explicitly specify a metric in order to export routes from other protocols into HELLO. A metric specified in export policy can override this. The default `metric` is 30000.

interface *list*
[[no]helloin]
[[no]helloout]
[metricin *metric]*
[metricout *metric]* ;

Controls various attributes of sending HELLO on specific interfaces. (See interfaces statement for a description of `list`.) Note that if there are multiple interfaces configured on the same subnet, only the first interface that has HELLO output configured sends the HELLO updates. This limitation is required because of the way the UNIX kernel operates. A future GateD release will hopefully remove this limitation. The default interface `list` value is `all`.

helloin (default)
nohelloin

Use `helloin` explicitly when using `nohelloin` on a wildcard interface descriptor. The `nohelloin` option ignores HELLO packets received over the specified interfaces.

helloout (default)
nohelloout

Use `helloout` explicitly when using `nohelloout` on a wildcard interface descriptor. The `nohelloout` option does not send HELLO packets over the specified interfaces.

metricin *metric*

HELLO metric to add to incoming routes before GateD installs them in the routing table. Makes this router prefer HELLO routes learned from other interfaces over those from the specified interface(s). The default is the kernel interface metric plus one. If using this as the absolute value, GateD does not add the kernel metric to the routing table.

metricout *metric*

HELLO metric to add to routes that are sent over the specified interface(s). Makes other routers prefer other sources of HELLO routes over this router. The default metric out `metric` value is `0`.

trustedgateways *list*

List of gateways from which HELLO accepts updates (host names or IP addresses). If used, HELLO accepts only updates from the gateways in the list. The default `list` value is `all`.

sourcegateways *list*
List of routers to which HELLO sends packets directly, not through multicasting or broadcasting. If used, HELLO accepts only updates from the gateways in the list. The default `list` value is `all`.

traceoptions packets
All HELLO packets, or packets `[detail] send` or `[detail] recv` (`detail` provides a more verbose format to provide more details; if used, `detail` must come before `send` or `recv`).

---

icmp

On systems without the BSD routing socket, GateD listens to ICMP messages received by the system. Processing of ICMP redirect messages is handled by the **`redirect`** statement.

Currently the only reason to specify the `icmp` statement is to be able to trace the ICMP messages that GateD receives.

Format

```
icmp { traceoptions options ; }
```

Options and Parameters

`traceoptions options ;`

ICMP tracing options (which you can modify with `detail` and `recv`) are as follows:

`packets`	All ICMP packets received
`redirect`	Only ICMP Redirect packets received
`routerdiscovery`	Only ICMP Router Discovery packets received
`info`	Only ICMP informational packets, which include mask request/response, info request/response, echo request/response and timestamp request/response
`error`	Only ICMP error packets, which include time exceeded, parameter problem, unreachable and source quench

redirect

GateD controls whether ICMP redirect messages can modify the kernel routing table. If disabled, GateD only prevents a system from listening to ICMP redirects. By default, ICMP redirects are enabled on hosts, and disabled on gateways that run as RIP or HELLO suppliers.

You configure ICMP redirect handling in the `GATED.CONF` file using a GateD protocol statement.

Format

```
redirect  yes | no | on | off
[{
    preference value ;
    interface list [ [no]redirects ] ;
    trustedgateways list ;
}];
```

Options and Parameters

yes | on
no | off
Enabled by default on hosts. Disabled by default on gateways running as RIP or HELLO suppliers.

preference *value*
Preference for routes learned from a redirect. The default preference `value` is `30`.

interface *list [[no]redirects]*
Enables and disables redirects interface by interface. (See *interfaces* for a description of *`list`*.) The default interface *`list`* value is `all`. The possible parameters are:

`redirects`	May be necessary when you use `noredirects` on a wildcard interface descriptor. (Default)
`noredirects`	Ignores redirects received over the specified interface(s). The default is to accept redirects on all interfaces.

trustedgateways *list*

List of gateways from which redirects are accepted (host names or addresses). By default, all routers on the shared network(s) are trusted to supply redirects. If used, only redirects from the gateways in the list are accepted. The default `list` value is `all`.

---

routerdiscovery server

The Router Discovery Protocol is an IETF standard protocol used to inform hosts of the existence of routers without having hosts wiretap routing protocols such as RIP. Use it in place of, or in addition to, statically configured default routes in hosts.

The protocol is in two parts, the server that runs on routers and the client that runs on hosts (see the next statement). GateD treats these much like two separate protocols that you can enable only one at a time.

The Router Discovery Server runs on routers and announces their existence to hosts. It does this by periodically multicasting or broadcasting a Router Advertisement to each interface on which it is enabled. These Router Advertisements contain a list of all router addresses on a given interface and their preference for use as a default router.

Initially these Router Advertisements occur every few seconds, then fall back to occurring every few minutes. In addition, a host may send a Router Solicitation to which the router will respond with a unicast Router Advertisement (unless a multicast or broadcast advertisement is due momentarily).

Each Router Advertisement contains an Advertisement Lifetime field indicating how long the advertised addresses are valid. This lifetime is configured such that another Router Advertisement is sent before the lifetime expires. A lifetime of zero indicates that one or more addresses are no longer valid.

On systems supporting IP multicasting, the Router Advertisements are sent to the all-hosts multicast address 224.0.0.1 by default. However, you can specify `broadcast`. When Router Advertisements are being sent to the all-hosts multicast address, or an interface is configured for the limited-broadcast address 255.255.255.255, all IP addresses configured on the physical interface are included in the Router Advertisement. When the Router advertisements are being sent to a net or subnet broadcast, only the address associated with that net or subnet is included.

> **Note:** Do not mix `routerdiscovery server` and `routerdiscovery client` statements in the `GATED.CONF` file or you may get unintended results. You should also include `preference` statements in the `interfaces` and `routerdiscovery` statements whenever possible.

Format

```
routerdiscovery server  yes | no | on | off
[{
    traceoptions state ;
    interface list
```

```
      [minadvinterval time]
      [maxadvinterval time]
      [lifetime time] ;
    address list
      [advertise] | [ignore]
      [broadcast] | [multicast]
      [ineligible] | [preference value] ;
}];
```

> **Note:** Interface *must* be mentioned in the “Interface” directive.

Options and Parameters

`yes | on`
`no | off`
Enables or disables Router Discovery Protocol Server.

`traceoptions state`
The **`state`** is the only trace option, which traces the state transitions. The Router Discovery Server does not directly support packet tracing options; tracing of router discovery packets is enabled through the `icmp` statement described in the *icmp* statement section.

`interface list`
Parameters that apply to physical interfaces. Note a slight difference in convention from the rest of GateD: `interface` specifies just physical interfaces, while `address` specifies protocol (in this case, IP) addresses.

`maxadvinterval time`
Maximum time allowed between sending broadcast or multicast Router Advertisements from the interface. Must be no less than `4` and no more than `30:00` (30 minutes). The default is `10:00` (10 minutes).

`minadvinterval time`

Minimum time allowed between sending unsolicited broadcast or multicast Router Advertisements from the interface. Must be no less than 3 seconds and no greater than `maxadvinterval`. The default is `0.75 X maxadvinterval`.

lifetime *time*
Lifetime of addresses in a Router Advertisement. Must be no less than `maxadvinterval` and no greater than `2:30:00` (two hours, thirty minutes). The default is 3 X `maxadvinterval`.

address *list*
Parameters that apply to the specified set of addresses on this physical interface. Note a slight difference in convention from the rest of GateD: `interface` specifies just physical interfaces while `address` is protocol (in this case, IP) addresses.

advertise (default)
ignore
The `advertise` keyword includes the specified addresses in Router Advertisements. The `ignore` keyword does not.

broadcast
multicast
The `broadcast` keyword includes the given addresses in a broadcast Router Advertisement because this system does not support IP multicasting, or some hosts on an attached network do not support IP multicasting. It is possible to mix addresses on a physical interface such that some are included in a broadcast Router Advertisement and some are included in a multicast Router Advertisement. This is the default if the router does not support IP multicasting.

The `multicast` keyword includes the given addresses in a multicast Router Advertisement. If the system does not support IP multicasting, the address(es) is not included. If the system supports IP multicasting, the default is to include the addresses in a multicast Router Advertisement if the given interface supports IP multicasting. If not, the addresses are included in a broadcast Router Advertisement.

preference *value*
ineligible

The `preference` keyword sets the preferability of the addresses as a default router address, relative to other router addresses on the same subnet. A 32-bit, signed, two's complement integer, with higher values meaning more preferable. Note that hex `80000000` may only be specified as ineligible. The default value is `0`. Use a `preference` statement whenever possible.

The `ineligible` keyword assigns the given addresses a preference of hex `80000000`, which means that it is not eligible to be the default route for any hosts. This is useful when the addresses should not be used as a default route, but are given as the next hop in an ICMP Redirect. This allows the hosts to verify that the given addresses are up and available.

---

routerdiscovery client

A host listens for Router Advertisements through the all-hosts multicast address (224.0.0.2) if IP multicasting is available and enabled, or on the interface's broadcast address. When starting up, or when reconfigured, a host may send a few Router Solicitations to the all-routers multicast address, 224.0.0.2, or the interface's broadcast address.

When a Router Advertisement with a non-zero lifetime is received, the host installs a default route to each of the advertised addresses. If the preference is ineligible, or the address is not on an attached interface, the route is marked unusable but retained. If the preference is usable, the metric is set as a function of the preference such that the route with the best preference is used. If more than one address with the same preference is received, the one with the lowest IP address will be used. These default routes are not exportable to other protocols.

When a Router Advertisement with a zero lifetime is received, the host deletes all routes with next hop addresses learned from that router. In addition, any routers learned from ICMP Redirects pointing to these addresses will be deleted. The same happens when a Router Advertisement is not received to refresh these routes before the lifetime expires.

> **Note:** Do not mix `routerdiscovery server` and `routerdiscovery client` statements in the `GATED.CONF` file or you may get unintended results. You should also include `preference` statements in the `interfaces` and `routerdiscovery` statements whenever possible.

Format

```
routerdiscovery client  yes | no | on | off
[{
    traceoptions state ;
    preference value ;
    interface list
      [enable] | [disable]
      [broadcast] | [multicast]
      [quiet] | [solicit] ;
}] ;
```

Options and Parameters

`yes | on`

no | off
Enables or disables the Router Discovery Protocol Client.

traceoptions state *;*
The `state` is the only trace option, which traces the state transitions. The Router Discovery Server does not directly support packet tracing options; tracing of router discovery packets is enabled through the `icmp` statement described in the *icmp* statement section.

preference *value ;*
Preference of all Router Discovery default routes. Use a `preference` statement whenever possible. Default is 55.

interface *list*
Parameters that apply to physical interfaces. Note a slight difference in convention from the rest of GateD: `interface` specifies just physical interfaces. The Router Discovery Client has no parameters that apply only to interface addresses.

enable (default)
disable
Either performs or does not perform Router Discovery on the specified interfaces.

broadcast
multicast
The `broadcast` keyword broadcasts Router Solicitations on the specified interfaces. This is the default if IP multicast support is not available on this host or interface.

The `multicast` keyword multicasts Router Solicitations on the specified interfaces. If IP multicast is not available on this host and interface, no solicitation is performed. The default is to multicast Router Solicitations if the host and interface support it, otherwise Router Solicitations are broadcast.

solicit (default)
quiet
Either sends or does not send Router Solicitations on this interface, even though Router Discovery is performed.

egp

GateD supports the Exterior Gateway Protocol (EGP). EGP is an exterior routing protocol that moves routing information between Autonomous Systems (ASs). Unlike interior protocols, EGP propagates only reachability indications, not true metrics. EGP updates contain metrics, called distances, which range from 0 to 255. GateD only compares EGP distances learned from the same AS. EGP currently has limited usage. By default, EGP is disabled.

Before EGP sends routing information to a remote router, it must establish an adjacency with that router. This occurs by exchanging Hello and I Heard You (I-H-U) messages with that router. (Hello should not to be confused with the HELLO protocol, or OSPF HELLO messages.) Computers communicating over EGP are called EGP neighbors, and the exchange of Hello and I-H-U messages is known as acquiring a neighbor.

Once you acquire a neighbor, the system polls it for routing information. The neighbor responds by sending an update containing routing information. If the system receives a poll from its neighbor, it responds with its own update packet. When the system receives an update, it includes routes from the update into its routing database. If the neighbor fails to respond to three consecutive polls, GateD assumes that the neighbor is down and removes the neighbor's routes from its database.

You configure EGP in the `GATED.CONF` file using a GateD protocol statement.

Format

```
egp  yes | no | on | off
[{
    preference value ;
    defaultmetric metric ;
    packetsize max ;
    traceoptions options ;
    group
      [peeras ASnumber]
      [localas ASnumber]
      [maxup number
      {
        neighbor host
        [metricout metric]
        [preference value]
        [preference2 value]
        [ttl ttl]
        [nogendefault]
        [importdefault]
        [exportdefault]
        [gateway gateway]
        [lcladdr local-address]
        [sourcenet network]
```

```
        [minhello | p1 time]
        [minpoll | p2  time]
        [traceoptions options] ;
      };
}] ;
```

Options and Parameters

yes | on
no | off (default)
Enables or disables EGP support. Disabled by default.

preference *value* ;
Preference for routes learned from EGP. A preference specified on the `group` or `neighbor` statements or by import policy can override this. The default preference `value` is `200`.

defaultmetric *metric* ;
Metric used when advertising routes over EGP. This choice of values requires you to explicitly specify a metric when exporting routes to EGP neighbors. A metric specified on the `neighbor` or `group` statements or in export policy can override this. The default `metric` is `255`.

packetsize *max* ;
Maximum size of a packet that EGP expects to receive from this neighbor. If EGP receives a larger packet, it is incomplete and EGP discards it. EGP notes the length of this packet and increases the expected size to be able to receive a packet of this size. Specifying the parameter prevents the first packet from being dropped. All packet sizes are rounded up to a multiple of the system page size. The default packet size `max` value is `8192`.

traceoptions *options* ;
Tracing options for EGP (can be overridden on a group or neighbor basis):

`packets`	All EGP packets, or packets `[detail] send` or `[detail] recv` (`detail` provides a more verbose format to provide more details; if used, `detail` must come before `send` or `recv`)

`hello`	EGP HELLO/I-HEARD-U packets used to determine neighbor reachability
`acquire`	EGP ACQUIRE/CEASE packets used to initiate and terminate EGP sessions
`update`	EGP POLL/UPDATE packets used to request and receive reachability updates

Group Clause

```
Group [peeras ASnumber] [localas ASnumber] [maxup number
{
  neighbor host
    [metricout metric]
    [preference value]
    [preference2 value]
    [ttl ttl]
    [nogendefault]
    [importdefault]
    [exportdefault]
    [gateway gateway]
    [lcladdr local-address]
    [sourcenet network]
    [minhello | p1 time]
    [minpoll | p2  time]
    [traceoptions options] ;
};
```

EGP neighbors must be members of a group, which groups all neighbors in one AS. Parameters specified in the `group` clause apply to all the subsidiary neighbors, unless explicitly overridden on a `neighbor` clause. Any number of `group` clauses can specify any number of `neighbor` clauses. You can specify any parameters from the `neighbor` subclause on the `group` clause to provide defaults for the whole group (which you can override for individual neighbors).

The `group` clause is the only place to set the following attributes:

`peeras ASnumber`
AS number expected from peers in the group. Learned dynamically.

`localas` *`ASnumber`*
AS that GateD represents to the group. Usually only used when masquerading as another AS. Use is discouraged. Set globally in `autonomoussystem`.

maxup *number*
Number of neighbors GateD should acquire from this group. GateD attempts to acquire the first `maxup` neighbors in the order listed. If one of the first neighbors is not available, it acquires one farther down the list. If after startup, GateD does manage to acquire the more desirable neighbor, it drops the less desirable one. By default, GateD acquires all neighbors in the group.

Group Neighbor Clause

```
egp  yes | no | on | off
[{
    preference value ;
    defaultmetric metric ;
    packetsize max ;
    traceoptions options ;
    group
      [peeras ASnumber]
      [localas ASnumber]
      [maxup number
      {
        neighbor host
        [metricout metric]
        [preference value]
        [preference2 value]
        [ttl ttl]
        [nogendefault]
        [importdefault]
        [exportdefault]
        [gateway gateway]
        [lcladdr local-address]
        [sourcenet network]
        [p1  time | minhello]
        [p2  time | minpoll]
        [traceoptions options] ;
      };
}] ;
```

Each `neighbor` subclause defines one EGP neighbor within a group. The only required part of the subclause is the host argument, the symbolic host name or IP address of the neighbor.

metricout *metric*
Metric used for all routes sent to this neighbor. Overrides the default metric set in the `egp` statement and any metrics specified by export policy, but only for this specific neighbor or group of neighbors.

preference *value*
Preference used for routes learned from these neighbors. Can differ from the default EGP preference set in the `egp` statement, so that GateD can prefer routes from one neighbor, or group of neighbors, over another. Import policy can explicitly override this.

preference2 *value*
Tie-breaker, in the case of a preference tie. The default `value` is `0`.

ttl *ttl*
IPL time-to-live. Provided when attempting to communicate with improperly functioning routers that ignore packets sent with a TTL 1. The default `ttl` for local neighbors is `1`; the default for nonlocal neighbors is `255`.

nogendefault
Does not generate a default route when EGP receives a valid update from its neighbor. The default route is only generated when you enable the `gendefault` option.

importdefault
Accepts the default route (0.0.0.0) if included in a received EGP update. For efficiency, some networks have external routers announce a default route to avoid sending large EGP update packets. The default route in the EGP update is ignored.

exportdefault
Includes the default route (0.0.0.0) in EGP updates sent to this EGP neighbor. Allows the system to advertise the default route using EGP. Normally a default route is not included in EGP updates.

gateway *gateway*
Router on an attached network used as the next hop router for routes received from this neighbor if a network is not shared with a neighbor. Rarely used.

lcladdr local-address

Address used on the local end of the connection with the neighbor. The local address must be on an interface shared with the neighbor, or with the neighbor's gateway when using the `gateway` option. A session only opens when an interface with the appropriate local address (through which the neighbor or gateway address is directly reachable) is operating.

`sourcenet` *`network`*
Network queried in the EGP Poll packets. If there is no network shared with the neighbor, specify one of the networks attached to the neighbor. Also use to specify a network shared with the neighbor, other than the one on which the EGP packets are sent. Normally not needed. The default is the network shared with the neighbor's address.

`p1 time` or **`minhello`**
Minimum acceptable interval between the transmission of EGP HELLO packets. If the neighbor fails to respond to three hello packets, GateD stops trying to acquire the neighbor. Setting a larger interval gives the neighbor a better chance to respond. The `minhello` is an alias for the `p1` value defined in the EGP specification. The default `time` value is `30`.

`p2 time` or **`minpoll`**
Time interval between polls to the neighbor. If three polls are sent without a response, the neighbor is declared "down" and all routes learned from that neighbor are removed from the routing database. A longer polling interval supports a more stable routing database but is not as responsive to routing changes. The `minpoll` is an alias for the `p2` value defined in the EGP specification. The default time `value` is `120`.

`traceoptions` *`options`*
Tracing options for this EGP neighbor, which are:

`packets`	All EGP packets, or packets `[detail] send` or `[detail] recv` (`detail` provides a more verbose format to provide more details; if used, `detail` must come before `send` or `recv`)
`hello`	EGP HELLO/I-HEARD-U packets used to determine neighbor reachability
`acquire`	EGP ACQUIRE/CEASE packets used to initiate and terminate EGP sessions

`update`	EGP POLL/UPDATE packets used to request and receive reachability updates

bgp

The Border Gateway Protocol (BGP) is an exterior routing protocol used to exchange routing information between multiple transit Autonomous Systems (ASs) as well as between transit and stub ASs. BGP is related to EGP but operates with more capability, greater flexibility, and less bandwidth required. BGP uses path attributes to provide more information about each route. It maintains an AS path, which includes the AS number of each AS the route transits, providing information sufficient to prevent routing loops in an arbitrary topology. You can also use path attributes to distinguish between groups of routes to determine administrative preferences. This allows greater flexibility in determining route preference to achieve a variety of administrative ends.

BGP supports two basic types of sessions between neighbors—internal (sometimes called IBGP) and external. Internal sessions run between routers in the same AS, while external sessions run between routers in different ASs. When sending routes to an external peer, the local AS number is prepended to the AS path. Hence routes received from an external peer are guaranteed to have the AS number of that peer at the start of the path. Routes received from an internal neighbor do not generally have the local AS number prepended to the AS path. Hence, these routes generally have the same AS path the route had when the originating internal neighbor received the route from an external peer. Routes with no AS numbers in the path may be legitimately received from internal neighbors; these indicate that the received route should be considered internal to your own AS.

The BGP implementation supports three versions of the BGP protocol—versions 2, 3 and 4. BGP versions 2 and 3 are similar in capability and function. They only propagate classed network routes, and the AS path is a simple array of AS numbers. BGP version 4 propagates fully general address-and-mask routes, and the AS path has some structure to represent the results of aggregating dissimilar routes.

External BGP sessions may or may not include a single metric, which BGP calls the Multi-Exit Discriminator (MED), in the path attributes. For BGP versions 2 and 3 this metric is a 16-bit unsigned integer; for BGP version 4 it is a 32-bit unsigned integer. In either case, smaller values of the metric are preferred. Currently this metric only breaks ties between routes with equal preference from the same neighbor AS. Internal BGP sessions carry at least one metric in the path attributes, which BGP calls the LocalPref. The size of the metric is identical to the MED. For BGP versions 2 and 3, this metric is better when its value is smaller; for version 4 it is better when it is larger. BGP version 4 sessions optionally carry a second metric on internal sessions, this being an internal version of the MED. The use of these metrics depends on the type of internal protocol processing specified.

BGP collapses routes with similar path attributes into a single update for advertisement. Routes received in a single update are readvertised in a single update. The churn caused by the loss of a neighbor is minimized, and the initial advertisement sent during peer establishment is maximally compressed. BGP does not read information from the kernel message by message, but fills the input buffer. It processes all complete messages in the buffer before reading again. BGP also does multiple reads to clear all

incoming data queued on the socket. This feature may cause other protocols to be blocked for prolonged intervals by a busy peer connection.

All unreachable messages are collected into a single message and sent prior to reachable routes during a flash update. For these unreachable announcements, the next hop is set to the local address on the connection, no metric is sent, and the path origin is set to incomplete. On external connections the AS path in unreachable announcements is set to the local AS; on internal connections the AS path is set to zero length.

BGP implementation expects external peers to be directly attached to a shared subnet, and expects those peers to advertise next hops that are host addresses on that subnet (although this constraint can be relaxed by configuration for testing). For groups of internal peers, however, there are several alternatives that can be selected by specifying the group type. Type internal groups expect all peers to be directly attached to a shared subnet so that, like external peers, the next hops received in BGP advertisements may be used directly for forwarding. Type routing groups instead determine the immediate next hops for routes, by using the next hop received with a route from a peer as a forwarding address, and using this to look up an immediate next hop in an IGP's routes. Such groups support distant peers, but need to be informed of the IGP whose routes they use to determine immediate next hops. Finally, type IGP groups expect routes from the group peers not to be used for forwarding at all. Instead, they expect that copies of the BGP routes are also received through an IGP, and that the BGP routes are only used to determine the path attributes associated with the IGP routes. Such groups also support distant peers and also need to be informed of the IGP with which they are running.

For internal BGP group types (and for test groups), where possible, a single outgoing message is built for all group peers based on the common policy. A copy of the message is sent to every peer in the group, with possible adjustments to the next hop field as appropriate to each peer. This minimizes the computational load of running large numbers of peers in these types of groups. BGP allows unconfigured peers to connect if an appropriate group was configured with an `allow` clause.

Format

```
bgp  yes | no | on | off
[{
    preference value ;
    defaultmetric metric ;
    traceoptions options ;
    group type
    external peeras ASnumber
              | internal peeras ASnumber
              | igp peeras ASnumber proto proto
              | routing peeras ASnumber proto proto interface list
              | test peeras ASnumber
    {
```

```
        allow
        {
          network
          network mask mask
          network masklen number
          all
          host host
        } ;
        peer host
          [metricout metric]
          [localas ASnumber]
          [nogendefault]
          [gateway gateway]
          [preference value]
          [preference2 value]
          [lcladdr local-address]
          [holdtime time]
          [version number]
          [passive]
          [sendbuffer number]
          [recvbuffer number]
          [indelay time]
          [outdelay time]
          [keep [all | none] ]
          [analretentive]
          [noauthcheck]
          [noaggregatorid]
          [keepalivesalways]
          [v3asloopokay]
          [nov4asloop]
          [logupdown]
          [ttl ttl]
          [traceoptions options] ;
    };
}];
```

Options and Parameters

`yes | on`
`no | off` (default)
Enables or disables BGP support. Disabled by default.

`preference value ;`
Preference for routes learned from BGP. A preference specified on the `group` or `peer` statements, or by import policy, can override this. The default preference `value` is `170`.

defaultmetric *metric ;*
Metric used when advertising routes over BGP. A metric specified on the `group` or `peer` statements, or in export policy, can override this. The default `metric` is `65535`.

traceoptions *options ;*
Tracing options for BGP. May be overridden on a group or peer basis. The trace `options` are:

`packets`	All BGP packets, or packets `[detail] send` or `[detail] recv [detail]` provides a more verbose format to provide more details; if used, `detail` must come before `send` or `recv`).
`open`	BGP OPEN packets used to establish a peer relationship
`update`	BGP UPDATE packets used to pass network reachability information
`keepalive`	BGP KEEPALIVE packets used to verify peer reachability

Group Type Clause

peeras
For `group type`, specify one of the following `peeras` options:

`external peeras` *`ASnumber`*	In the classic external BGP group, full policy checking is applied to all incoming and outgoing advertisements. The external neighbors must be directly reachable through one of the machine's local interfaces. No metric included in external advertisements and the next hop is computed with respect to the shared interface.
`internal peeras` *`ASnumber`*	Internal group operating where there is no IP-level IGP; for example, an SMDS network or MILNET. All neighbors in this group must be directly reachable over a single interface. All next-hop information is computed with respect to this interface. Import and export policy may be applied to group advertisements. Routes received from external BGP or EGP neighbors are readvertised with the received metric.

`igp peeras` *`ASnumber`*	Internal group that runs in association with an interior protocol. The IGP group examines routes the IGP exports, and sends an advertisement only if the path attributes could not be entirely represented in the IGP tag mechanism. Only the AS path, path origin, and transitive optional attributes are sent with routes. No metric is sent, and the next hop is set to the local address the connection uses. Received internal BGP routes are not used or readvertised. Instead, the AS path information is attached to the corresponding IGP route and the latter is used for readvertisement. Since internal IGP peers are sent only a subset of the routes the IGP exports, the export policy used is the IGP's. There is no need to implement the "don't route from peers in the same group" constraint, since the advertised routes are routes that IGP already exports.
`routing peeras` *`ASnumber`*	Internal group that uses the routes of an interior protocol to resolve forwarding addresses. A type routing group propagates external routes between routers not directly connected, and computes immediate next hops for these routes by using the BGP next hop that arrived with the route as a forwarding address to be resolved using an internal protocol's routing information. In essence, internal BGP is used to carry AS external routes, while the IGP is expected to only carry AS internal routes, and the latter is used to find immediate next hops for the former. The next hop in BGP routes advertised to the type routing peers are set to local address on BGP connection to those peers, as it is assumed a route to this address is propagated over IGP. • `proto proto` - Interior protocol used to resolve BGP route next hops, and can be the name of any IGP in the configuration. • `interface list` - Optionally provides a list of interfaces whose routes are carried over the IGP for which third party next hops can be used instead.

`test peeras` *`ASnumber`*	Extension to external BGP that implements a fixed policy using test peers. Fixed policy and special case code make test peers relatively inexpensive to maintain. Test peers do not need to be on a directly attached network. If GateD and the peer are on the same (directly attached) subnet, the advertised next hop is computed with respect to that network; otherwise the next hop is the local machine's current next hop. All routing information advertised by and received from a test peer is discarded, and all BGP advertisable routes are sent back to the test peer. Metrics from EGP- and BGP-derived routes are forwarded in the advertisement; otherwise no metric is included.

Group Type Allow Clause

Allows peer connections from any addresses in the specified range of network and mask pairs. Configure all parameters for these peers on the group clause. The internal peer structures are created when an incoming open request is received, and destroyed when the connection is broken. (For details on specifying the network/mask pairs, see *Route Filtering*.)

Group Type Peer Clause

Configures an individual peer. Each peer inherits all parameters specified on a group as defaults. You can override these defaults using parameters explicitly specified in the `peer` subclause. Allows the following parameters:

`metricout` ***`metric`***

Primary metric on all routes sent to the specified peer(s). Overrides the default metric, a metric specified on the group, and any metric specified by export policy.

`localas` ***`ASnumber`***

AS that GateD represents to this group of peers. `ASnumber` is set globally in `autonomoussystem`.

`nogendefault`

Does not generate a default route when EGP receives a valid update from its neighbor. The default route is generated only when enabling the `gendefault` option.

gateway *gateway*
If a network is not shared with a peer, specifies a router on an attached network used as the next hop router for routes received from this neighbor. Not needed in most cases.

preference *value*
Preference used for routes learned from these peers. Can differ from the default BGP preference set in the `bgp` statement, so that GateD can prefer routes from one peer, or group of peers, over others. Import policy can explicitly override this.

preference2 *value*
In the case of a preference tie, can break the tie.

lcladdr local-address
Address used on the local end of the TCP connection with the peer. For external peers, the local address must be on an interface shared with the peer or with the peer's gateway when using the `gateway` parameter. A session with an external peer only opens when an interface with the appropriate local address (through which the peer or gateway address is directly reachable) is operating. For other types of peers, a peer session is maintained when any interface with the specified local address is operating. In either case, incoming connections are only recognized as matching a configured peer if they are addressed to the configured local address.

holdtime *time*
BGP holdtime value to use when negotiating the connection with this peer, in seconds. According to BGP, if GateD does not receive a keepalive, update, or notification message within the period specified in the Hold Time field of the BGP Open message, the BGP connection is closed. The value must be either 0 (no keepalives are sent) or at least 3.

version *number*
Version of the BGP protocol to use with this peer. If specified, only the specified version is offered during negotiation. Currently supported versions are `2`, `3`, and 4. By default, the highest supported version is used first, and version negotiation is attempted.

passive
Does not attempt active OPENs to this peer. GateD should wait for the peer to issue an open. By default, all explicitly configured peers are active.

sendbuffer *number* and **recvbuffer *number***
Controls the amount of send and receive buffering asked of the kernel. The maximum `number` supported is 65535 bytes, although many kernels have a lower limit. Not needed on normally functioning systems. By default, the maximum supported is configured.

indelay *time* and **outdelay *time***
Dampens route fluctuations. The `indelay` is the amount of time a route learned from a BGP peer must be stable before it is accepted into the GateD routing database. The `outdelay` is the amount of time a route must be present in the GateD routing database before it is exported to BGP. Default `time` in both cases is 0.

keep all
Retains routes learned from a peer even if the routes' AS paths contain one of our exported AS numbers.

analretentive
Issues warning messages when receiving questionable BGP updates such as duplicate routes and/or deletions of nonexistent routes. Normally these events are silently ignored.

noauthcheck
Communicates with an implementation that uses some form of authentication other than the normal authentication field of all ones.

noaggregatorid
GateD should specify the routerid in the `aggregator` attribute as zero (instead of its routerid) in order to prevent different routers in an AS from creating aggregate routes with different AS paths.

keepalivesalways

GateD should always send keepalives, even when an update could have correctly substituted for one. Allows interoperability with routers that do not completely obey the protocol specifications on this point.

v3asloopokay
By default, GateD does not advertise routes whose AS path is looped (that have an AS appearing more than once in the path) to version 3 external peers. Setting this flag removes this constraint. Ignored when set on internal groups or peers.

nov4asloop
Does not advertise routes with looped AS paths to version 4 external peers. Can be useful to avoid advertising such routes to peer which would incorrectly forward the routes on to version 3 neighbors.

logupdown
Logs a message using syslog whenever a BGP peer enters or leaves ESTABLISHED state.

ttl *ttl*
Provided when attempting to communicate with improperly functioning routers that ignore packets sent with a TTL 1. Not all kernels allow the TTL to be specified for TCP connections. The default ttl for local neighbors is 1; the default for nonlocal neighbors is 255.

traceoptions *options ;*
Tracing options for this BGP neighbor include:

`packets`	All BGP packets, or packets `[detail] send` or `[detail] recv` (`detail` provides a more verbose format to provide more details; if used, `detail` must come before `send` or `recv`)
`open`	BGP OPEN packets used to establish a peer relationship
`update`	BGP UPDATE packets used to pass network reachability information
`keepalive`	BGP KEEPALIVE packets used to verify peer reachability

ospf

Open Shortest Path First (OSPF) routing is a shortest-path-first (SPF) or link-state protocol. OSPF is an interior gateway protocol that distributes routing information between routers in a single Autonomous System (AS). OSPF chooses the least cost path as the best path. Suitable for complex networks with many routers, OSPF provides equal cost multipath routing where packets to a single destination can be sent over more than one interface simultaneously. In a link-state protocol, each router maintains a database describing the entire AS topology, which it builds out of the collected link state advertisements of all routers. Each participating router distributes its local state (that is, the router's usable interfaces and reachable neighbors) throughout the AS by flooding.

Each multiaccess network with at least two attached routers has a designated router and a backup designated router. The designated router floods a link state advertisement for the multiaccess network and has other special responsibilities. The designated router concept reduces the number of adjacencies required on a multiaccess network.

OSPF lets you group networks into areas. Routing information passed between areas is abstracted, which can significantly reduce routing traffic. OSPF uses four different types of routes, listed in order of preference—intra-area, inter-area, type 1 external, and type 2 external. Intra-area paths have destinations within the same area, while inter-area paths have destinations in other OSPF areas. AS External (ASE) routes are routes to destinations external to the AS. Routes imported into OSPF as type 1 routes are supposed to be from IGPs whose external metrics are directly comparable to OSPF metrics.

When making a routing decision, OSPF adds the internal cost of the AS Border router to the external metric. Type 2 ASEs are used for EGPs whose metrics are not comparable to OSPF metrics. In this case, GateD uses only the internal OSPF cost of the AS Border router in the routing decision.

From the topology database, each router constructs a tree of the shortest paths with itself as the root. This shortest-path tree gives the route to each destination in the AS. Externally derived routing information appears on the tree as leaves. The link-state advertisement format distinguishes between information acquired from external sources and from internal routers, so that there is no ambiguity about the source or reliability of routes. Externally derived routing information (for example, routes learned from EGP or BGP) passes transparently through the AS and is separate from OSPF's internally derived data. Each external route can also be tagged by the advertising router, enabling a passing of additional information between routers on the borders of the AS.

OSPF optionally includes type of service (TOS) routing and allows administrators to install multiple routes to a given destination for each type of service (such as for low delay or high throughput.) A router running OSPF uses the destination address and the TOS to choose the best route to the destination.

OSPF intra- and inter-area routes are always imported into the GateD routing database with a preference of 10. It would be a violation of the protocol if an OSPF router did not participate fully in the area's

OSPF, so it is not possible to override this. Although it is possible to give other routes lower preference values explicitly, it is ill-advised to do so.

Hardware multicast capabilities are also used where possible to deliver link-status messages.

OSPF areas are connected by the backbone area, the area with identifier 0.0.0.0. All areas must be logically contiguous and the backbone is no exception. To permit maximum flexibility, OSPF allows the configuration of virtual links to enable the backbone area to appear contiguous when they are actually not.

All routers in an area must agree on that area's parameters. A separate copy of the link-state algorithm is run for each area. Because of this, most configuration parameters are defined on a per area basis. All routers belonging to an area must agree on that area's configuration. Misconfiguration leads to adjacencies not forming between neighbors, and routing information might not flow, or even loop.

Authentication. You can authenticate OSPF protocol exchanges. Authentication guarantees that routing information is imported only from trusted routers, to protect the Internet and its users. There are two authentication schemes available. The first uses a simple authentication key of up to eight characters and is standardized. The second is still experimental and uses the MD5 algorithm and an authentication key of up to 16 characters.

The simple password provides very little protection, because in many cases it is possible to easily capture packets from the network and learn the authentication key. The experimental MD5 algorithm provides much more protection, as it does not include the authentication key in the packet.

The OSPF specification currently specifies that you configure the authentication type per area with the ability to configure separate passwords per interface. This was extended to allow configuration of different authentication types and keys per interface. Also, you can specify both a primary and a secondary authentication type and key on each interface. Outgoing packets use the primary authentication type, but incoming packets may match either the primary or secondary authentication type and key.

You configure OSPF in the `TCPWARE:GATED.CONF` file using a GateD protocol statement.

Format

```
ospf  yes | no | on | off
[{
    defaults
    {
      preference value ;
      cost cost ;
      tag [as] tag ;
      type 1 | type 2 ;
    } ;
    exportlimit routes ;
```

```
    exportinterval time ;
    traceoptions options;
    monitorauthkey key ;
    monitorauth  none | [simple | md5] authkey ;
    backbone | area area
    {
      authtype 0 | authtype 1 | none | simple ;
      stub [cost cost] ;
      networks
      {
        network [restrict] ;
        network mask mask [restrict] ;
        network masklen number [restrict] ;
        host host [restrict] ;
      } ;
      stubhosts
      {    host cost cost ; } ;
      interface list [cost cost]
      {   interface-parameters } ;
      interface list nonbroadcast [cost cost]
      {
        pollinterval time ;
        routers
        {
          gateway [eligible] ;
        };
        interface-parameters
      } ;
      /* Backbone only: */
      virtuallink neighborid router-id transitarea area
      {   interface-parameters } ;
    } ;
}] ;
```

Options and Parameters

yes | on
no | off
Enables or disables OSPF support.

defaults
Defaults used when importing OSPF ASE routes into the GateD routing table, and exporting routes from the GateD routing table into OSPF ASEs, including:

`preference value;`	How OSPF routes compete with routes from other protocols in the GateD routing table. The default preference `value` is 150.

`cost` *`cost ;`*	Used when exporting a non-OSPF route from the GateD routing table into OSPF as an ASE. Export policy can explicitly override this. The default `cost` is 1.
`tag` *`[as] tag ;`*	OSPF ASE routes have a 32-bit tag field that the OSPF protocol does not use, but export policy can use it to filter routes. When OSPF interacts with an EGP, you can use the tag field to propagate AS path information. In this case you would specify the `as` keyword and the tag is limited to 12 bits of information. The default `tag` value is 0.
`type 1 or 2 ;`	Export policy can explicitly change and override the default here. The default is `type 1`.

`exportlimit` ***`routes ;`***
How many ASEs are generated and flooded in each batch. The default export limits `routes` value is 100.

`exportinterval` ***`time ;`***
How often a batch of ASE link state advertisements are generated and flooded into OSPF. The default export interval `time` value is 1 (once per second).

`traceoptions` ***`options ;`***
In addition to the following OSPF specific trace flags, OSPF supports the state which traces interface and neighbor state machine transitions:

`lsabuild`	Link State Advertisement creation
`spf`	Shortest Path First (SPF) calculations
`lsatransmit`	Link State Advertisement (LSA) transmission
`lsareceive`	LSA reception
`state`	State transitions

Packet tracing options (which you can modify with `detail`, `send`, and `recv`):

`hello`	OSPF HELLO packets used to determine neighbor reachability
`dd`	OSPF Database Description packets used in synchronizing OSPF databases
`request`	OSPF Link State Request packets used in synchronizing OSPF databases
`lsu`	OSPF Link State Update packets used in synchronizing OSPF databases
`ack`	OSPF Link State Ack packets used in synchronizing OSPF databases

`monitorauthkey key ;`
`monitorauth none | [simple | md5] authkey ;`
You can query the OSPF state using the `ospf_monitor` utility, which sends nonstandard OSPF packets that generate a text response from OSPF. If you configure an authentication key, the incoming requests must match the specified authentication key. These packets cannot change OSPF state, but the act of querying OSPF can expend system resources. Not authenticated by default.

Backbone/Area Clause Options and Parameters

`backbone` or **`area area`**
Configures each OSPF router into at least one OSPF area. If you configure more than one area, at least one must be the backbone. Configure the backbone using the `backbone` keyword only; you cannot specify it as area 0. The backbone interface can be a `virtuallink`.

Further parameters include:

`authtype 0 or 1 or none or simple`	OSPF specifies an authentication scheme per area. Each interface in the area must use this same authentication scheme, although it can use a different authentication key. 0 is the same as `none`; 1 is the same as `simple`.
`stub [cost cost]`	A stub area is one in which there are no ASE routes. Use cost to inject a default route into the area with the specified cost.

networks { *network [*restrict*]* ; *network* mask *mask [*restrict*]* ; *network* masklen *number* *[*restrict*];* host *host [*restrict*]* ; } ;	The networks list describes the scope of an area. Intra-area LSAs that fall within the specified ranges are not advertised into other areas as inter-area routes. Instead, the specified ranges are advertised as summary network LSAs. If you specify restrict, the summary network LSAs are not advertised. Intra-area LSAs that do not fall into any range are also advertised as summary network LSAs. This option is very useful on well-designed networks in reducing the amount of routing information propagated between areas. The entries in this list are either networks, or a subnetwork/mask pair.
stubhosts { *host* cost *cost* ; }	The stubhosts list specifies directly attached hosts that should be advertised as reachable from this router, and the costs with which they should be advertised. Specify point-to-point interfaces here on which it is not desirable to run OSPF. It is also useful to assign an additional address to the loopback interface (one not on the 127 network) and advertise it as a stub host. If this address is the same one used as the router ID, it enables routing to OSPF routers by router ID, instead of by interface address. This is more reliable than routing to one of the router's interface addresses, which may not always be reachable.
interface list cost cost {interface-parameters}	Use this form of the interface clause (with the optional cost value, and immediately followed by the interface-parameters) to configure a broadcast (which requires IP multicast support) or a point-to-point interface. (See the interfaces statement for a description of list.) Each interface has a cost. The costs of all the interfaces a packet must cross to reach a

	destination are summed to get the cost to that destination. The `cost` can be any non-zero value (the default is 1).

The following are the `interface-parameters`. You can specify them on any class of interface:

```
enable | disable ;
retransmitinterval time ;
transitdelay time ;
priority value ;
hellointerval time ;
routerdeadinterval time ;
authkey key ;
```

`retransmitinterval` *time*	Number of seconds between link state advertisement retransmissions for adjacencies belonging to this interface.
`transitdelay` *time*	Estimated number of seconds required to transmit a link state update over this interface. Takes into account transmission and propagation delays and must be greater than 0.
`priority` *value*	Number between 0 and 255 specifying the priority for becoming the designated router on this interface. When two routers attached to a network both attempt to become designated router, the one with the highest priority prevails. A router whose router priority is 0 is ineligible to become designated router.
`hellointerval` *time*	Length of time, in seconds, between Hello packets that the router sends on the interface.
`routerdeadinterval` *time*	Number of seconds not hearing a router's Hello packets before the router's neighbors will declare it down.
`authkey` *key*	Used by OSPF authentication to generate and verify the authentication field in the OSPF header. You can configure the authentication key on a per-interface basis. Specify it using one to eight decimal digits separated by periods, a one to eight byte hexadecimal string preceded by `0x`, or a one to eight character string in double quotes.

The form of the `interface` clause with the `nobroadcast` option is for point-to-point interfaces only. By default, OSPF packets to neighbors on point-to-point interfaces are sent using the IP multicast mechanism. GateD detects this condition and falls back to using sending unicast OSPF packets to this point-to-point neighbor.

If you do not want IP multicasting, because the remote neighbor does not support it, specify `nobroadcast` to force the use of unicast OSPF packets. You can also use this option to eliminate warnings when GateD detects the bug mentioned previously. (See the previous page for the `interface-parameters`.)

Use this form of the **`interface`** clause to specify a nonbroadcast interface on a nonbroadcast multiaccess (NBMA) media. Since an OSPF broadcast media must support IP multicasting, you must configure a broadcast-capable media, such as Ethernet, that does not support IP multicasting as a nonbroadcast interface. A nonbroadcast interface supports any of the standard interface clauses listed previously, plus the following two that are specific to nonbroadcast interfaces:

`pollinterval` *`time`*	Before adjacency is established with a neighbor, OSPF packets are sent periodically at the specified poll interval.
`routers` *`gateway`*	By definition, it is not possible to send broadcast packets to discover OSPF neighbors on a nonbroadcast, so you must configure all neighbors. The list includes one or more neighbors and an indication of their eligibility to become a designated router.

`virtuallink neighborid` ***`routerid`*** **`transitarea`** ***`area`***
`{` ***`interface-parameters`*** **`}`** ***`;`***
For backbone only: Virtual links are used to establish or increase connectivity of the backbone area. The `neighborid` is the router-ID of the other end of the virtual link. The transit area specified must also be configured on this system. You can specify all standard interface parameters defined by the interface clause previously described on a virtual link. (See the previous page for the `interface-parameters`.)

static

The `static` statements define the static routes GateD uses. A single `static` statement can specify any number of routes. These statements must occur after **`protocol`** statements and before `control` statements in `GATED.CONF`. Specify any number of `static` statements, each containing any number of static route definitions. You can override these routes with ones with better preference values.

Format

```
static
{
  host host  gateway list
           | network [mask mask  | masklen number] gateway list
           | default gateway list
                     [interface list]
                     [preference value]
                     [retain]
                     [reject]
                     [blackhole]
                     [noinstall]
                     ;
             network [mask mask  |  masklen number]
                      interface interface
                     [preference value]
                     [retain]
                     [noinstall]
                     ;
} ;
```

Options and Parameters

`host...gateway list` or **`default gateway list`**

Most general form of the static statement. Defines a static route through one or more gateways. Static routes are installed when one or more of the gateways listed are available on directly attached interfaces. If more than one eligible gateway is available, they are limited by the number of multipath destinations supported.

The second form of the `network mask...` clause farther down in the statement is for primitive support of multiple network addresses on one interface.

`interface list`

Gateways are valid only when they are on one of these interfaces.

preference *value*
Preference of this static route. Controls how this route competes with routes from other protocols. The default `value` is 60.

retain
Prevent specific static routes from being removed. Normally GateD removes all routes except interface routes from the kernel forwarding table during a graceful shutdown. Useful for ensuring that some routing is available when GateD is down.

noinstall
Do not install the route in the kernel forwarding table when active, but make it still exportable to other protocols. Normally the route with the lowest preference is installed there and is the route exported to other protocols.

---

import

The control statements are:

```
import
export
aggregate
generate
```

Format

```
import [restrict | preference value]
```

The `import` statements control importing routes from routing protocols, and installing the routes in GateD's routing database. The format of an `import` statement varies depending on the source protocol. In all cases, you can specify one of two keywords to control how routes compete with other protocols:

`restrict`	Restrict the routes from the routing table. In some cases this means that the routes are not installed in the routing table. In others, it means that they are installed with a negative preference; this prevents them from becoming active so that they will not be installed in the forwarding table or exported to other protocols.
`preference` *value*	Preference value used when comparing this route to other routes from other protocols. The route with the lowest preference available at any given route becomes the active route, is installed in the forwarding table, and can be exported to other protocols. The individual protocols configure the default preferences.

Importing Routes from BGP and EGP

You can control EGP importation by AS. Note that EGP and BGP versions 2 and 3 only support propagating natural networks, so the host and default route filters are meaningless. BGP version 4 supports propagating any destination along with a contiguous network mask.

EGP and BGP both store any routes rejected implicitly by their not being mentioned in a route filter, or explicitly if `restrict` appears in the routing table with a negative preference. A negative preference prevents a route from becoming active, which prevents it from being installed in the forwarding table or exported to other protocols. This removes the need to break and reestablish a session on reconfiguring if changing the importation policy.

The syntax of the `import` statement for importing routes from BGP or EGP is any of the following:

```
import proto  bgp | egp  autonomoussystem ASnumber restrict ;
import proto  bgp | egp  autonomoussystem ASnumber
  [preference value]  {
  route-filter [restrict  |  preference value] ; } ;
import proto bgp aspath ASpathregexp
  origin   any | [igp] [egp] [incomplete]   restrict ;
import proto bgp aspath ASpathregexp
  origin   any | [igp] [egp] [incomplete]
                            [preference value] {
                            routefilter [restrict | preference value] ; } ;
```

The third and fourth variation of the `import` statements is for BGP only and supports controlling propagation by using AS path regular expressions. An AS path is a list of ASs that routing information passes through to get to a router, and an indicator of the origin of the AS path. Use this information to set the preference of one path to a destination network over another. You do this by listing patterns applied to AS paths when importing and exporting routes. Each AS that a route passes through prepends its AS number to the beginning of the AS path.

Aspath Clause

The following `aspath` clause in the **`import`** statement indicates that an AS matching the `ASpathregexp` with the specified origin is matched. The parameters follow:

```
aspath ASpathregexp origin  any | [igp] [egp] [incomplete]
```

Aspath Clause Regular Expression

`ASpathregexp`

Regular expression, with the alphabet as the set of AS numbers, consisting of one or more AS path expressions, which are terms and operators. An AS path term (`ASpathterm`) consists of the following:

`ASnumber`	Any valid AS system number, from 1 through 65534.
`.`	Matches any AS number.
`(ASpathregexp)`	Parentheses group sub-expressions. An operator such as asterisk (*) or question mark (?) works on a single element or on a regular expression enclosed in parentheses.

Aspath Clause Operators

AS path operators consists of the following:

`ASpathterm {`*`m`*`}`	Exactly `m` repetitions, where *`m`* is a positive integer.
`ASpathterm {`*`m`*`,}`	*`m`* or more repetitions, where *`m`* is a positive integer.
`ASpathterm {`*`m`*`,`*`n`*`}`	At least *`m`* and at most `n` repetitions, where `m` and `n` are both nonnegative integers and *`m`* <= *`n`*.
`ASpathterm *`	Zero or more repetitions (shorthand for `{0,}`).
`ASpathterm +`	One or more repetitions (shorthand for `{1,}`).
`ASpathterm ?`	Zero or one repetition (shorthand for `{0,1}`).
`ASpathterm | ASpathterm`	Matches either term.

Remaining Import Statement Options

`origin  any | [igp] [egp] [incomplete]`

Details the completeness of AS path information. An origin of `igp` indicates that the route was learned from an interior routing protocol and is most likely complete. An origin of `egp` indicates that the route was learned from an exterior routing protocol that does not support AS paths (EGP for example), and that the path is most likely not complete. When the path information is definitely not complete, use `incomplete`.

Importing Routes from RIP, HELLO, and Redirects

You can control importing RIP, HELLO, and Redirect routes by any protocol, source interface, or source gateway. If using more than one, they are processed from most general (protocol) to most specific (gateway). RIP and HELLO do not support preferences to choose between routes of the same protocol; they use metrics instead. They also do not save rejected routes since they have short update intervals.

The syntax of the **`import`** statement for importing routes from RIP, HELLO, or redirects is either of the following:

```
import proto  rip | hello | redirect
       [interface list  |  gateway list]
       restrict ;

import proto  rip | hello | redirect
       [interface list  |  gateway list]
       [preference value]
      {   routefilter [restrict  |  preference value] ; } ;
```

Importing Routes from OSPF

You can only control importing AS External (ASE) routes. OSPF intra- and inter-area routes are always imported into the GateD routing table with a **`preference`** of 10. If using an `ospftag`, the import clause only applies to routes with the specified tag.

You can only restrict importing OSPF ASE routes if functioning as an AS border router. Do this by specifying an `export ospfase` clause. Specifying an empty export clause can restrict importing ASEs, when no ASEs are exported.

Like the other interior protocols, you cannot use **`preference`** to choose between OSPF ASE routes; OSPF costs accomplish this. Routes rejected by policy go into the table with a negative preference.

The syntax of the `import` statement for importing routes from OSPF is either of the following:

```
import proto ospfase [tag ospftag] restrict ;

import proto ospfase [tag ospftag]
         [preference value]
        { routefilter [restrict  | preference value] ; } ;
```

export

Format

`export [restrict | metric metric]`

The `export` statement controls which routes GateD advertises to other systems. Like `import`, the `export` syntax varies slightly for each protocol. Both syntaxes are similar and the meanings of many of the parameters are the same. The main difference is that while source information controls importing routes, both destination and source information control exporting routes.

The outer portion of a given `export` statement specifies the destination of the routing information you control. The middle portion restricts the sources. The innermost portion is a route filter used to select individual routes.

One thing that applies in all cases is the specification of a metric. All protocols define a default metric for routes exported. In most cases, this can be overridden at several levels of the export statement. The most specific specification of a metric is the one applied to the route exported. The values you can specify for a metric depend on the destination protocol the **`export`** statement references:

`restrict`	Do not export anything. If specified on the destination portion of the **`export`** statement, it means not to export anything to this destination. If specified on the source portion, it means not to export anything from this source. If specified as part of a route filter, it means not to export the routes matching that filter.
`metric` *`metric`*	Metric used when exporting to the specified destination.

Exporting to EGP and BGP

The AS controls exporting to EGP and BGP, the same policy applied to all routers in the AS. EGP metrics range from 0 through 255, with 0 the most attractive. BGP metrics are 16-bit unsigned quantities (that range from 0 through 65535, inclusive with 0 the most attractive). While BGP version 4 actually supports 32-bit unsigned quantities, GateD does not yet support this.

If you do not specify an export policy, only routes to attached interfaces are exported. If you specify any policy, the defaults are overridden; you should explicitly specify everything you want exported. (Note that EGP and BGP versions 2 and 3 only support the propagation of natural networks, so the host and default route filters are meaningless. BGP version 4 supports the propagation of any destination along with a contiguous network mask.)

The syntax of the `export` statement for exporting routes to EGP or BGP is either of the following:

```
export proto  bgp | egp  as ASnumber  restrict ;
export proto  bgp | egp  as ASnumber  [metric metric]
    {  exportlist ; } ;
```

Exporting to RIP and HELLO

Any protocol, interface, or gateway can control exporting to RIP and HELLO. If you specify more than one, they are processed from most general (protocol) to most specific (gateway). It is not possible to set metrics for exporting RIP routes into RIP, or exporting HELLO routes into HELLO. Attempts to do this are silently ignored.

If you do not specify an export policy, RIP and interface routes are exported into RIP and HELLO, and interface routes are exported into HELLO. If you specify any policy, the defaults are overridden; it is necessary to explicitly specify everything that should be exported.

RIP version 1 and HELLO assume that all subnets of the shared network have the same subnet mask, so they are only able to propagate subnets of that network. RIP version 2 is capable of propagating all routes, when not sending version 1 compatible updates.

To announce routes that specify a next hop of the loopback interface (static and internally generated default routes) over RIP or HELLO, specify the metric at some level in the export clause. Just setting a default metric is not sufficient. This is a safeguard to verify that the announcement is intended.

The syntax of the **`export`** statement for exporting routes to RIP or HELLO is either of the following:

```
export proto  rip | hello
       [interface list  |  gateway list] restrict ;

export proto  rip | hello
       [interface list  |  gateway list]  [metric metric]
    {  exportlist ; } ;
```

Exporting to OSPF

It is not possible to create OSPF intra- or inter-area routes by exporting routes from the GateD routing table into OSPF. It is only possible to export from the GateD routing table into OSPF ASE routes. It is also not possible to control the propagation of OSPF routes within the OSPF protocol.

There are two types of OSPF ASE routes, type 1 and type 2 (see the OSPF protocol configuration for details on the two types). Specify the default type using the `defaults` subclause of the `ospf` clause. You can override this with the `export` statement.

OSPF ASE routes also have the provision to carry a tag. This is an arbitrary 32-bit number you can use on OSPF routers to filter routing information. (See the OSPF protocol configuration for details on OSPF tags.) You can override the default tag specified by the `ospf defaults` clause with a tag specified on the `export` statement.

The syntax of the `export` statement for exporting routes to OSPF is either of the following:

```
export proto osfpase [type 1 | 2] [tag ospf-tag]  restrict ;

export proto osfpase [type 1 | 2] [tag ospf-tag]
       [metric metric]
      { exportlist ; } ;
```

Exporting BGP and EGP Routes

You can specify BGP and EGP routes by source AS. You can export all routes by AS path. The syntax of the **`proto`** statement for exporting BGP or EGP routes is either of the following:

```
proto  bgp | egp  autonomoussystem ASnumber  restrict ;
proto  bgp | egp  autonomoussystem ASnumber  [metric metric]
     {  routefilter [restrict  |  metric metric] ; } ;
```

Exporting RIP and HELLO Routes

You can export RIP and HELLO routes by protocol, source interface, or source gateway. The syntax of the **`proto`** statement for exporting RIP or HELLO routes is either of the following:

```
proto  rip | hello
    [interface list  |  gateway list]  restrict ;
proto  rip | hello
    [interface list  |  gateway list]  [metric metric]
    {  routefilter [restrict  |  metric metric] ; } ;
```

Exporting OSPF Routes

You can export both OSPF and OSPF ASE routes into other protocols. The syntax of the **`proto`** statement for exporting OSPF routes is either of the following:

```
proto  ospfase | ospfase  restrict ;
proto  ospfase | ospfase [metric metric]
    { routefilter [restrict  |  metric metric] ; } ;
```

Exporting Routes from Nonrouting Protocols with Interface

If you want GateD to export direct or static routes, or routes learned from the kernel, use the protocol statement or interface statement along with the interface of the next hop in the GateD configuration file. The syntax of the **proto** statement for exporting routes from nonrouting protocols with an interface is either of the following:

```
proto  direct | static | kernel
    [interface list] restrict ;

proto  direct | static | kernel
    [interface list]  [metric metric]
    { routefilter [restrict  |  metric metric] ; } ;
```

The `proto` statement parameters include:

`direct`	Routes to directly attached interfaces.
`static`	Static routes specified in a static clause.
`kernel`	On systems with the routing socket, routes learned from the routing socket are installed in the GateD routing table with a protocol of `kernel`. You can export these routes by referencing this protocol. This is useful when it is desirable to have a script install routes with the `route` command and propagate them to other routing protocols.

Exporting Routes from Nonrouting Protocols by Protocol

If you want GateD to export default or aggregate routes, use the protocol statement in the GateD configuration file. The syntax of the **proto** statement for exporting routes from nonrouting protocols by protocol is either of the following:

```
proto default | aggregate  restrict ;
proto default | aggregate
       [metric metric]
       { routefilter [restrict  |  metric metric] ; } ;
```

The `proto` statement parameters include:

`default`	Routes created by the `gendefault` option. Use route generation instead.
`aggregate`	Routes synthesized from other routes when using the `aggregate` and `generate` statements.

Exporting by AS Path

When configuring BGP, all routes get an AS path when added to the routing table. For all interior routes, this AS path specifies IGP as the origin and no ASEs in the AS path (the current AS is added when the route is exported). For EGP routes, this AS path specifies EGP as the origin and the source AS as the AS path. For BGP routes, the AS path is stored as learned from BGP. (The AS path regular expression syntax appears in the *Importing Routes from BGP and EGP* subsection.)

The syntax of the **proto** statement for exporting by AS path is either of the following:

```
proto  proto | all  aspath ASpathregexp
  origin  any | [igp] [egp] [incomplete]  restrict ;
proto  proto | all  aspath ASpathregexp
  origin  any | [igp] [egp] [incomplete]  [metric metric]
       { routefilter [restrict  |  metric metric] ; } ;
```

Exporting by Route Tag

Both OSPF and RIP version 2 currently support tags. All other protocols always have a tag of zero. You can select the source of exported routes based on this tag. This is useful when classifying routes by tag when exporting them into a given routing protocol. The syntax of the **proto** statement for exporting by route tag is either of the following:

proto *proto* | all all tag *tag* restrict ;
proto *proto* | all all tag *tag*
*[*metric *metric]*
{ *routefilter [*restrict | metric *metric]* ; } ;

aggregate

Use route aggregation to generate a more general route from a specific one. Use it, for example, at an AS border to generate a route to a network to be advertised through EGP, given the presence of one or more subnets of that network learned through RIP. Regional and national networks also use route aggregation to reduce routing information. By carefully allocating network addresses to clients, regional networks can just announce one route to regional networks instead of hundreds. No aggregation occurs unless explicitly requested in an **aggregate** statement.

Aggregate routes are not actually used for packet forwarding by the originator of the aggregate route, only by the receiver (if it wishes). A router, receiving a packet that does not match one of the component routes that led to the generation of an aggregate route, is supposed to respond with an ICMP `network unreachable` message. This prevents packets for unknown component routes from following a default route into another network where they would be continuously forwarded back to the border router, until their TTL expires. Sending an unreachable message for a missing piece of an aggregate is only possible on systems that support reject routes, which TCPware does not.

Format

```
aggregate  default | network [mask mask  |  masklen number]
     [preference value] [brief]
     { proto [all | direct | static | kernel | aggregate | proto]
           [as AS  |  tag tag  |  aspath ASpathregexp] restrict ;
      proto [all | direct | static | kernel | aggregate | proto]
           [as AS  |  tag tag  |  aspath ASpathregexp]  [preference value]
           { routefilter [restrict  |  preference value] ; } ;
     } ;
```

Options and Parameters

preference *value*

The default preference `value` is `130`.

brief

Truncate the AS path to the longest common AS path. The default is to build an AS path consisting of SETs and SEQUENCEs of all contributing AS paths.

proto *proto*

In addition to the special protocols listed, you can select the contributing protocol from among those currently configured in GateD.

as *AS*
Restrict selection of routes to those learned from the specified AS.

tag *tag*
Restrict selection of routes to those with the specified tag.

aspath ASpathregexp
Restrict selection of routes to those that match the specified AS path.

restrict
Restrict certain routes from contributing to the specified aggregate.

A route can only contribute to an aggregate route that is more general than itself; it must match the aggregate under its mask. Any given route can only contribute to one aggregate route, which will be the most specific configured, but an aggregate route can contribute to a more general aggregate.

---

generate

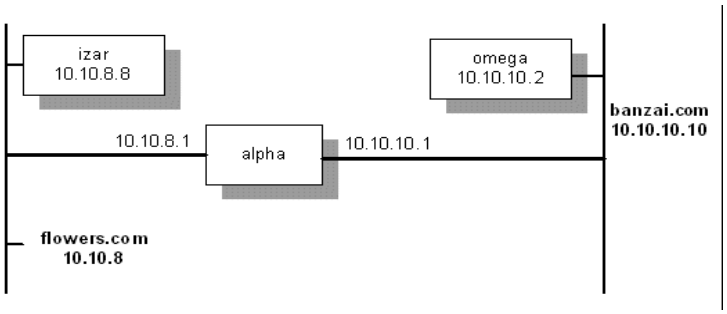
A slight variation on aggregation is generating a route based on certain conditions. This is sometimes known as the "route of last resort." This route inherits the next hops and AS path from the contributor specified with the lowest (most favorable) preference. The most common usage is to generate a default based on the presence of a route from a peer on a neighboring backbone.

Format

```
generate  default | network [mask mask  |  masklen number]
       [preference value] [brief]
            { [as AS  |  tag tag  |  aspath ASpathregexp]
            restrict ;
       proto [all | direct | static | kernel | aggregate | proto]
            [as AS  |  tag tag  |  aspath ASpathregexp]
            [preference value]
            { routefilter [restrict  |  preference value] ; } ;
       } ;
```

Sample GateD Configurations

The below diagram shows two networks connected within an AS using RIP. The following configuration files show the RIP statements on each end host and gateway `alpha`, which has IP forwarding enabled. All systems are running GateD.



Configuration File for izar

```
#  turn on RIP and listen for updates.
#
rip on;
```

GateD Configuration File for alpha

```
#  turn on RIP.
#
rip yes;
#
#  use RIP to pass routing information to the banzai network.
#
export proto rip interface 10.10.10.1
    {
    #  we know about the flowers network, so announce it.
    #
    proto direct {
      10.10.8.0 mask 255.255.255.0;
      };
    #  use RIP to announce all routes learned from flowers.
    #
    proto rip interface 10.10.8.0 {
       all;
       };
     };
```

GateD Configuration File for omega

```
#  turn on RIP and listen for updates.
#
rip on;
```

Below is a sample RIP statement where the gateway announces a default route to the backbone, and announces all of the individual subnet routes to the outside world.

```
#  enable RIP:
#
rip yes;

#  using RIP, announce all local subnets via interface 192.168.12.3:
#
export proto rip interface 192.168.12.3 metric 3
    {
       proto rip interface 192.168.1.5
       {
           all;
       };
     };
#
#  Using RIP, announce default via interface 192.168.1.5:
#
export proto rip interface 192.168.3.1
     {
        proto rip interface 192.168.1.5
         {
            default;
         };
      };
```

Below is a configuration for AS 283 that enables RIP and OSPF, which you can use to test both.

```
# this interface is passive:
#
interfaces {
        interface SVA-0 passive;
        };
#
#  this Autonomous System number is 283:
#
autonomoussystem 283;
#
#  turn on RIP:
#    packets are to be broadcast.
#    metric for routes learned via other protocols is 5.
#    multicast RIP V2 packets on SVA-0.
```

```
#
rip yes {
     broadcast;

                        defaultmetric 5;
                        interface SVA-0 version 2 multicast;
                        };
#
# turn on OSPF:
#    Trace Link State Advertisement creation and
#    Shortest Path First calculations
#    use authentication key "ZZZZZZZZ" when handling OSPF queries.
#    this system is on the backbone.
#      use simple password authentication for this area.
#      make this system very unlikely to be a designated router.
#      set the OSPF header authentication key to "YYYYYYYY" for
#      packets going out on SVA-0.
#
ospf yes {
       traceoptions lsabuild spf;
       monauthkey "ZZZZZZZZ";
       backbone {
           authtype simple;
           interface all {
              priority 2;
              };
           interface SVA-0 {
              authkey "YYYYYYYY";
              };
           };
      };
```

Below is a configuration for a static route.

```
#
# in this example our host's address is 192.168.1.42
#
static {
         192.168.2.0 masklen 24 interface 192.168.1.42 retain;
        default gateway 192.168.1.1 ;
       ;
```

10. Network Time Protocol (NTP)

Introduction

This chapter describes how to configure and manage the Network Time Protocol (NTP) to synchronize timekeeping among a set of distributed time servers and clients. The synchronization is totally transparent to users.

TCPware's NTP also includes two standard query programs, `NTPQ` and `NTPDC`, the utility `NTPTRACE`, and the `NTPDATE` program.

TCPware's NTP implementation is based on Network Time Protocol Version 4.2.

Overview of NTP

The standard timescale used by most nations of the world is Coordinated Universal Time (UTC), which is based on the Earth's rotation about its axis, and the Gregorian Calendar, which is based on the Earth's rotation about the Sun. UTC time is disseminated by various means, including radio and satellite navigation systems, telephone modems and portable clocks. For reasons of cost and convenience, it is not possible to equip every computer with a method of receiving these time signals directly. However, it is possible to equip some number of computers acting as primary time servers to synchronize a much larger number of secondary servers and clients connected by a common network. In order to do this, a distributed network clock synchronization protocol is required which can transmit an accurate reading to one or more clients and adjust each client clock as required. This is what the Network Time Protocol is for.

The synchronization protocol determines the time offset of the server clock relative to the client clock. On request, the server sends a message including the time the request arrived and the time the response was returned. The client included the time it sent the request in the request message and records the time the response arrived back from the server as well. With these four values, or *timestamps*, the client can determine the server-client propagation delay (by assuming that this is half the round-trip time) and subtract this from the time difference between client and server time settings to determine its clock offset relative to the server. In general, this is a useful approximation; however, in the Internet of today, network paths and the associated delays can differ significantly due to the individual service providers,

and this can contribute to error in determining offset. A stable, symmetrical (in terms of propagation delay) and reliable connection to the time server is important in minimizing this type of error.

NTP attempts to compensate for the problem of network instability by allowing the use of several servers as time sources and determining which of them is most reliable through statistical means that compare them to each other. All sources are assumed to have correct times, but those that differ markedly from the group are eventually ignored as having unreliable connections or being otherwise poor sources of correct time information. This tends to limit malicious activities as well, where a server that reports false times is inserted in a network, as well as bad time servers that result from hardware failure that can have much the same effect.

Clock errors can be due to other causes than variations in network delay. Other causes include latencies in computer hardware and software (jitter), as well as clock oscillator instability (wander). Despite these sources of error, NTP can, over many updates, discipline a clock to stay remarkably close to the actual time, even when a time server is not available for some period.

Programs and Files

There are several programs and files that make up NTP in TCPware. These are described in more detail later in this chapter.

Program Files

The following programs make up the NTP implementation in TCPware:

NTPD	The NTP server process used to maintain the system clock and to pass time information to lower stratum clients and servers. While this program runs as a server process, it also functions as a client in requesting time data from other servers on the network.
NTPDATE	An interactive "one shot" time setting utility. It is useful for setting the initial time on a system, perhaps at boot time, to minimize the correction necessary with NTPD. If NTPD is not desired for some reason, NTPDATE used in a recurring batch job can be used to maintain a system's clock accurately.
NTPDC	NTPDC is used to query the NTPD server about its current state and to request changes in that state. Extensive state and statistics information is available through the NTPDC interface. In addition, nearly all the configuration options which can be specified at startup using NTPD's configuration file may also be specified at run time using NTPDC.

NTPQ	The NTPQ utility program is used to query NTP servers about current state and to request changes in that state. Requests to read and write arbitrary variables can be assembled, with raw and pretty-printed output options being available. NTPQ can also obtain and print a list of peers in a common format by sending multiple queries to the server. NTPQ and NTPDC perform similar functions, but use different protocols to communicate with NTPD.
NTPTRACE	NTPTRACE determines where a given NTPD server gets its time, and follows the chain of NTP servers back to their master time source.

Configuration Files

NTP uses the following configuration files:

NTP.CONF	TCPWARE:NTP.CONF is used to specify servers from which time information is requested as well as many other aspects of NTPD behavior. See the description of NTPD for a list of options and their definitions. NTP.CONF is read only at NTPD startup, so if you make changes you will need to restart NTPD.
NTP.KEYS	TCPWARE:NTP.KEYS is used to define security information used in authorization operations. (The keys used by the NTPQ and NTPDC programs are checked against passwords requested by the programs and entered by hand.)
TIMEZONES.DAT	A default set of timezone rules is compiled into NTP. You can use the TCPWARE SET/TIMEZONE command in conjunction with this file to add other time zone rules. See the *Time Zone* section for more information about time zones and the use of this file.

Other Files

NTP uses the following files:

NTPD.LOG	The NTPD server outputs progress and error information to the TCPWARE:NTPD.LOG file. (See *Troubleshooting Tips*.)

NTP.DRIFT	The TCPWARE:NTP.DRIFT file consists of data maintained by NTPD and used to speed up clock frequency adjustments when NTPD is restarted. You should not modify this file.

Configuration

NTP Network Design

NTP does not attempt to synchronize clocks to each other. Rather, each server attempts to synchronize to Universal Coordinated Time (UTC) using the best available sources and available transmission paths to those sources. This is a fine point which is worth understanding. A group of NTP-synchronized clocks may be close to each other in time, but this is not a consequence of the clocks in the group having synchronized to each other, but rather because each clock has synchronized closely to UTC via the best source it has access to.

The most important factor in providing accurate, reliable time is the selection of modes and servers to be used in the configuration file. An NTP network should consist of a multiply redundant hierarchy of servers and clients, with each level in the hierarchy identified by stratum number. Primary servers operate at stratum one and provide synchronization to secondary servers operating at stratum two and so on to higher strata. In this hierarchy, clients are simply servers that have no dependents.

Determine which list of peers/servers you want to include in the configuration file. Include at least one (but preferably two) peer or server hosts that you are assured:

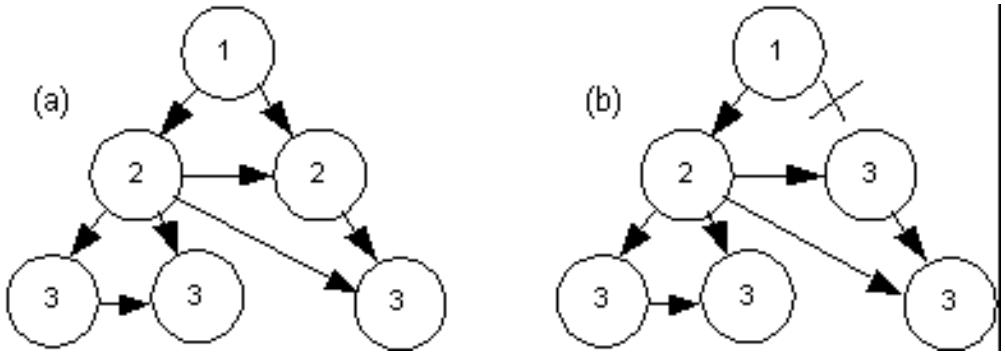
- Are running NTP.
- Provide accurate time.
- Synchronize to Internet Time Servers (if they are not themselves ITSs).

Two hosts provide reliability in case one goes down. You do not need to identify what stratum each host is. NTP determines this through the reference information it sends in its data exchanges.

NTP data is exchanged periodically between hosts as encapsulated in UDP datagrams, and adjustments are made based on an NTP algorithm. The frequency of exchange is related to the server's experience of time corrections. The more accurate the local clock becomes over time and after many adjustments, the less often the NTP server checks the need for corrections. The frequency of exchange is rarely intrusive to normal network operation. Also, the unreliability of UDP has no measurable impact on the process, and the process does not depend on any such reliability.

Primary servers are servers with reliable time sources, such as GPS receivers or atomic clocks, and can be found on the public internet, or set up within an intranet. Stratum numbers equate to the number of intermediate servers (or *hops*) between a given host and the Stratum 1 server it is ultimately referencing. Stratum numbers are not assigned statically, but change as server connections change. NTP servers can be (and often are) other types of systems running NTP, not just OpenVMS systems.

The stratum method allows for backup timekeeping in case a node or connection goes down, and stratum numbers may change as a consequence. In diagram A below, each node has a stratum number based on hop count, with the ITS at the top of the pyramid. The solid arrows are the active synchronization paths and direction of timing information flow; the lighter arrows are background synchronization paths where timing information is exchanged but not necessarily used for synchronization. Diagram B below shows the same network with one of the connections broken — note that the stratum for the affected peer increases from 2 to 3.



NTP makes local system time adjustments by either *slewing* or *stepping* the clock. Slewing runs the clock faster or slower than its normal frequency. Stepping sets the clock immediately to the correct time. Stepping occurs infrequently, only when there is a large time offset to adjust, such as when starting NTPD or when making daylight savings time (DST) changes.

Under some circumstances it can be disruptive to step the clock, such as when running database software that journals transactions. Such software can become very confused when a transaction is completed prior to the time at which it began, such as can happen when a clock is stepped backwards during a transaction's lifetime. In such cases the `slewalways` configuration option can be used to turn off stepping of the clock and force all adjustments to be made by slewing. For large time changes, such as DST changeovers, the adjustment can take a long time (several hours) to complete, and during this time the system's time will not be correct. For this reason it is not wise to allow a system set for `slewalways` to act as a server to another system.

In determining your NTP network design, keep in mind the way that the NTP protocol works, and how NTPD will determine the correct time. There should be several time servers in the configuration for each node, with good, reliable, and non-congested paths between them. Nodes that will act only as

clients can use the `slewalways` option, but nodes used as time sources by other nodes should generally allow stepping of the time so that inaccurate times are not reported for extended periods at Daylight Savings Time (DST) changeovers. See the *NTP.CONF* section of this chapter for more information on the `slewalways` option and the *Timezone* section for more information on DST handling.

Authentication

NTP implements a general-purpose address- and mask-based restriction list (see the `restrict` configuration option). While this is not adequate to prevent hacking attacks, it can be useful to lock out a malfunctioning server that is disrupting normal operations. See the *Access Control Commands* section for more information.

The NTP standard specifies an extension which provides cryptographic authentication of received NTP packets. This is implemented in NTPD using the MD5 algorithm to compute a digital signature, or message digest. See the *Authentication Using a Keys File* section for more information.

Finding Servers

In many large organizations there is an administrator for the organization's networks which handles management of various network services. This person or department can usually provide information on local NTP servers, and often suggest configurations that are known to work.

There are also several publicly available time servers on the internet. A list of such servers can be accessed via the web at http://www.eecis.udel.edu/~ntp These data are updated on a regular basis using information provided voluntarily by various site administrators.

NTP.CONF

The `NTP.CONF` file is used to specify the initial configuration of the NTPD server. It contains information about servers and peers, modes of operation, non-default file names, and other configuration data. See the table for a list of available options, and a brief description of what each does.

`peer`	`peer [ address ] [ version 4 ] [key 0] [minpoll 6] [maxpoll 10]` Specifies that the server is to operate in symmetric active mode with the specified remote server. In this mode, the local server can be synchronized to the remote server and, in addition, the remote server can be synchronized by the local server. This is useful in a network of servers where, depending on

	various failure scenarios, either the local or remote server may be the better source of time. The `address` can be a domain name or an IP address in dotted quad notation The `key` specifies that all packets sent to an address are to include authentication fields encrypted using the specified key identifier, which is an unsigned 32-bit integer. The default is to not include an encryption field. *`version`* specifies the protocol version number to be used for outgoing NTP packets. Versions 1, 2, 3 and 4 are the choices, with version 4 the default.
`server`	`server [ address ] [ version 4 ] [key 0] [minpoll 6] [maxpoll 10]` Specifies that the local server is to operate in client mode with the specified remote server. In this mode, the local server can be synchronized to the remote server, but the remote server can never be synchronized to the local server. The *`address`* can be a domain name or an IP address in dotted quad notation The `key` specifies that all packets sent to an address are to include authentication fields encrypted using the specified key identifier, which is an unsigned 32-bit integer. The default is to not include an encryption field. *`version`* specifies the protocol version number to be used for outgoing NTP packets. Versions 1, 2, 3 and 4 are the choices, with version 4 the default.
`broadcast`	`broadcast [ address ] [ version 4 ] [ key 0 ] [ ttl 1 ]`

	Specifies broadcast mode, where the local server sends periodic broadcast messages to a client population at the broadcast/multicast address specified. This specification applies only to the local server operating as a sender. For operation as a broadcast client, see the `broadcastclient` option that follows. In this mode `address` is usually the broadcast address on (one of) the local networks. The `key` specifies that all packets sent to an address are to include authentication fields encrypted using the specified key identifier, which is an unsigned 32-bit integer. The default is to not include an encryption field. *`version`* specifies the protocol version number to be used for outgoing NTP packets. Versions 1, 2, 3 and 4 are the choices, with version 4 the default. `ttl` specifies the number of routers to pass through before the packet is discarded. The default is 127 routers.
`broadcastclient`	`broadcastclient` This command directs the local server to listen for broadcast messages at the broadcast address of the local network. Upon hearing a broadcast message for the first time, the local server measures the nominal network delay using a brief client/server exchange with the remote server, then enters the `broadcastclient` mode, in which it listens for and synchronizes to succeeding broadcast messages. **Note:** In order to avoid accidental or malicious disruption in this mode, both the local and remote servers should operate using authentication and the same trusted key and key identifier.
`multicastclient`	`multicastclient [ address ]`

	Specifies that this host is a multicast client for multicasts to the specified multicast address.
`manycastclient`	`manycastclient [address] [version 4] [key 0] [minpoll 6] [maxpoll 10]` Specify that this host is a manycast client and provide relevant settings.
`manycastserver`	`manycastserver [ address ]` Specify that this host is a manycast server for the given address.
`broadcastdelay`	`broadcastdelay 0.004` The broadcast and multicast modes require a special calibration to determine the network delay between the local and remote servers. Ordinarily, this is done automatically by the initial protocol exchanges between the client and server. In some cases, the calibration procedure may fail due to network or server access controls, for example. This command specifies the default delay to be used under these circumstances. Typically (for Ethernet), a number between 0.003 and 0.007 seconds is appropriate. The default when this command is not used is 0.004 seconds.
`restrict`	`restrict [ address ] [ mask 255.255.255.0 ] ignore|noserve|notrust|noquery` Restrict access from and to the specified address for the specified types of access. The `address` argument, expressed in dotted quad form, is the address of a host or network. The `mask` argument, also expressed in dotted quad form, defaults to 255.255.255.255, meaning that the `address` is treated as the address of an individual host. A default entry (address 0.0.0.0, mask 0.0.0.0) is always included and, given the sort algorithm, is always the first entry in the list.

	Note! While `numeric-address` is normally given in dotted-quad format, the text string default, with no mask option, can be used to indicate the default entry. `ignore` - Ignores all packets from hosts which match this entry. If this flag is specified, neither queries nor time server polls are responded to. `noquery` - ignores all NTP mode 6 and 7 packets (information queries and configuration requests generated by NTPQ and NTPDC) from the source. Time service is not affected. `noserve` - Ignores NTP packets whose mode is other than 6 or 7. In effect, time service is denied, though queries may still be permitted. `notrust` - Treats these hosts normally in other respects, but never uses them as synchronization sources.
`driftfile`	`driftfile file_name` Specify the name of the drift file. The default is `MULTINET:NTP.DRIFT` if this option is not used. The drift file is used to record the frequency offset of the local clock oscillator. If the file exists, it is read at startup in order to set the initial frequency offset and then updated once per hour with the current frequency offset computed by the daemon. If the file does not exist or this command is not given, the initial frequency offset is assumed zero. In this case, it may take some hours for the frequency to stabilize and the residual timing errors to subside.
`local-master`	`local-master` *`stratum`*

	Indicates that the system is to act as an authoritative time source for other systems at a stratum level lower than the specified *stratum*.
keys	keys file_name Specify the name of the keys file. The default is MULTINET:NTP.KEYS if this option is not used.
statsdir	statsdir path Indicates the full path of a directory where statistics files should be created. This keyword allows the (otherwise constant) filegen filename prefix to be modified for file generation sets, which is useful for handling statistics logs.
filegen	filegen *[* file *filename]* *[* type *typename]* *[* enable | disable *]* Configures the generation fileset name. Generation filesets provide a means for handling files that are continuously growing during the lifetime of a server. Server statistics are a typical example for such files. At most one element of the set is being written to at any one time. The type given specifies when and how data is directed to a new element of the set. filename - This string is directly concatenated to the directory MULTINET: or the directory prefix specified using the statsdir option. The suffix for this filename is generated according to the type of a fileset. typename - A file generation set is characterized the following typenames: • none - One element of the fileset is used for each, NTPD server. • day - One file generation set element is created per day. A day is defined as the period between 00:00 and 24:00 UTC. The fileset member suffix consists of a dot (.) and a day specification in the form

	YYYYMMDD. YYYY is a 4-digit year number (such as 2003). MM is a two digit month number. DD is a two digit day number. Thus, all information written at 10 December 2002 would end up in a file named `prefix filename.20021210`. • `week` - Any fileset member contains data related to a certain week of a year. The term week is defined by computing day-of-year modulo 7. Elements of such a file generation set are distinguished by appending the following suffix to the fileset filename base: a dot, a 4-digit year number, the letter **`w`**, and a 2-digit week number. For example, information from January 10th, 2003 would end up in a file with suffix `.2003W1`. • `month` - One generation fileset element is generated per month. The filename suffix consists of a dot, a 4-digit year number, and a 2-digit month. • `year` - One generation file element is generated per year. The filename suffix consists of a dot and a 4-digit year number. • `age` - This type of file generation sets changes to a new element of the fileset every 24 hours of server operation. The filename suffix consists of a dot, the letter **`a`**, and an 8-digit number. This number is taken to be the number of seconds the server is running at the start of the corresponding 24-hour period. Information is only written to a file generation by specifying `enable`; output is prevented by specifying `disable`.
`publickey`	`publickey file_name` Specify the public keys file location.
`privatekey`	`privatekey file_name` Specify the private key file location.
`clientlimit`	`clientlimit [ n ]` Sets the `client_limit` variable that limits the number of simultaneous access-controlled clients. The default value is 3.
`clientperiod`	`clientperiod [ 3600 ]`

	Sets the `client_limit_period` variable that specifies the number of seconds after which a client is considered inactive and thus no longer is counted for client limit restriction. The default value is 3600 seconds.
`trustedkey`	`trustedkey [`*`key`*`]` Specifies the encryption key identifiers which are trusted for the purposes of authenticating peers suitable for synchronization. The authentication procedures require that both the local and remote servers share the same key and key identifier for this purpose, although different keys can be used with different servers. The key arguments are 32-bit unsigned integers. **Note:** NTP key 0 is fixed and globally known. If meaningful authentication is to be performed, the 0 key should not be trusted.
`requestkey`	`requestkey [ key]` Specifies the key identifier to use with the NTPDC program, which uses a proprietary protocol specific to this distribution of NTPD. The key argument to this command is a 32-bit unsigned integer. If no `requestkey` command is included in the configuration file, or if the keys do not match, NTPDC requests are ignored.
`controlkey`	`controlkey [ key ]` Specifies the key identifier to use with the NTPQ program, which uses the standard protocol defined in RFC 1305. The key argument to this command is a 32-bit unsigned integer. If no `controlkey` command is included in the configuration file, or if the keys do not match, NTPQ requests are ignored.
`setvar`	`setvar [ value ]`

	This command adds an additional system variable. These variables can be used to distribute additional information such as the access policy. If the variable of the form `name = value` is followed by the `default` keyword, the variable is listed as part of the default system variables (`ntpq rv` command). These additional variables serve informational purposes only. They are not related to the protocol other than that they can be listed. The known protocol variables always override any variables defined using the `setvar` mechanism.
`logfile`	`logfile file_name` Specify the logfile name. The default is `MULTINET:NTPD.LOG` if this option is not specified.
`logconfig`	`logconfig [+|-|=] [{sync|sys|peer|clock}{{,all}{info|statistics|events|status}}]...` Specify logging options.
`enable`	`enable auth|bclient|ntp|kernel|monitor|stats|calibrate` Enable various options. `auth` - Enables the server to synchronize with unconfigured peers only if the peer was correctly authenticated using a trusted key and key identifier. The default for this setting is `disable`. `bclient` - When enabled, this is identical to the `broadcastclient` command. The default for this flag is `disable`. `ntp` - Enables the server to adjust its local clock by means of NTP. If disabled, the local clock free-runs at its intrinsic time and frequency offset. This flag is useful in case the local clock is controlled by some other device or

	protocol and NTP is used only to provide synchronization to other clients. The default for this flag is `enable`. `kernel` - this setting is not used in the MultiNet implementation. `monitor` - Enables the monitoring facility. See the `monlist` command of the NTPDC program for further information. The default for this flag is `enable`. `stats` - Enables the statistics facility. For further information, see *Monitoring Commands*. The default for this flag is `enable`. `calibrate` - this setting is not used in the MultiNet implementation.
`disable`	`disable auth|bclient|ntp|kernel|monitor|stats|calibrate` Disable various options. See the `enable` entry for details.
`slewalways`	`Slewalways` Specify that the clock time is always to be slewed, never stepped. NTPD normally steps the clock when there is a relatively large time error to adjust. The `slewalways` command directs the local NTP server to always slew the clock, regardless of how large the required correction is. This command is useful to avoid an abrupt one hour clock change when daylight savings time (DST) changes occur. For DST changes when `slewalways` is specified, NTPD slews the clock over a period of about 6 hours.
`panic`	`panic max_adjust_time`

	Set the maximum time change that will be allowed. If non-zero, this value should always be at least 4000 seconds to allow for DST time changes even on systems with large time errors. If set to zero, the panic sanity check is disabled and a clock offset of any value will be accepted.
debug	debug [level] Set the debug logging severity level.
set_vms_logicals	set_vms_logicals Causes the NTP server to also adjust the value of the VMS logicals SYS$TIMEZONE_DIFFERENTIAL, SYS$TIMEZONE_DAYLIGHT_SAVING and SYS$TIMEZONE_NAME when it changes the MULTINET_TIMEZONE logical at DST start or end. The following files will also be updated: DTSS$TIMEZONE_DIFFERENTIAL and SYS$TIMEZONE.DAT NTP does NOT set SYS$TIMEZONE_RULE, which generally does not change. The format of SYS$TIMEZONE_RULE is specified in SYS$MANAGER:UTC$TIME_SETUP.COM.
call_dst_proc	call_dst_proc Causes the NTP server to spawn a subprocess to execute the MULTINET:NTPD_DST_PROC.COM procedure, if such a procedure file exists with the proper protections, when changing into or out of DST, or when first starting up. See the *Using the call_dst_procoption* below for more information.
set_clock_daily	set_clock_daily

	When this is included in NTP.CONF, then NTPD will only make one call a day (instead of once an hour) to the routine that sets the TOY clock to ensure that the value is preserved. This reduces the number of entries in the Integrity system event logs. NTPD may set the clock more often than daily, but it will be done only to correct any drift that is detected. In our tests on a RX2600 this happened approximately every 6 hours.

Access Control Commands

NTP implements a general-purpose address- and mask-based restriction list (see the *restrict* config option). The list is sorted by address and by mask, and the list is searched in this order for matches, with the last match found defining the restriction flags associated with the incoming packets. The source address of incoming packets is used for the match, with the 32-bit address combined with the mask associated with the restriction entry and then compared with the entry's address (which was also combined with the mask) to look for a match.

The restriction facility was implemented to conform with the access policies for the original NSFnet backbone time servers. While this facility may be otherwise useful for keeping unwanted or broken remote time servers from affecting your own, it should not be considered an alternative to the standard NTP authentication facility. Source address-based restrictions are easily circumvented by a determined hacker.

Authentication Using a Keys File

The NTP standard specifies an extension which provides cryptographic authentication of received NTP packets. This is implemented in NTPD using the MD5 algorithm to compute a digital signature, or message digest. The specification allows any one of possibly four billion keys, numbered with 32-bit key identifiers, to be used to authenticate an association. The servers involved in an association must agree on the key and key identifier used to authenticate their messages.

Keys and related information are specified in the file TCPWARE:NTP.KEYS, which should be exchanged and stored using secure procedures. There are three classes of keys involved in the current implementation. One class is used for ordinary NTP associations, another for the NTPQ utility program, and the third for the NTPDC utility program.

Key File Format

For MD5, keys are 64 bits (8 bytes), read from the TCPWARE:NTP.KEYS file. While key number 0 is fixed by the NTP standard (as 64 zero bits) and may not be changed, one or more of the keys numbered 1 through 15 may be arbitrarily set in the keys file.

The keys file uses the same comment conventions as the configuration file. Key entries use a fixed format of the form:

```
keyno  type  key
```

- *keyno* is a positive integer.
- *type* is a single character M for the MD5 key format.
- *key* is the key itself.

The key is a one-to-eight-character ASCII string using the MD5 authentication scheme.

> **Note:** Both the keys and the authentication scheme must be identical between a set of peers sharing the same key number.

> **Note:** The keys used by the NTPQ and NTPDC programs are checked against passwords requested by the programs and entered by hand.

Using the call_dst_proc option

When NTPD is started, and whenever the local time zone shifts between daylight savings DST and standard (STD) time, if the local zone rule specifies such behavior, the NTPD server will check the TCPWARE_TIMEZONE logical, and set it if required. The setting will only be between the DST name and the STD name for the zone, so the configuration described above is still necessary, but if your system was down during a DST shift, this can correct the logical name to match the current system clock time and the applicable zone rule when NTPD is started. If your NTP.CONF file specifies the set_vms_logicals option, the SYS$TIMEZONE_DIFFERENTIAL, SYS$TIMEZONE_DAYLIGHT_SAVING and SYS$TIMEZONE_NAME logicals will be updated as well.

Since there are many systems with other time-related logical names, or other items that may need updating or adjusting based on a DST change, the `call_dst_proc` option has been provided. If this option is used in `NTP.CONF`, the `NTPD` server will look for a file called `TCPWARE:NTPD_DST_PROC.COM` any time it checks on the `TCPWARE_TIMEZONE` logical (at startup and at a DST shift). If this file exists, and has the proper protections (no `WORLD` write or execute access, and owned by `SYSTEM` (`[1,4]`)) a sub-process will be spawned to execute it. This procedure can contain any commands needed, but care should be exercised in constructing this file, as it will be executing with the same privileges as the NTPD process. A "placeholder" procedure is included with TCPware contents are all comments and will do nothing as shipped.

The invocation of the `TCPWARE:NTPD_DST_PROC.COM` procedure will be equivalent to this:

```
@TCPWARE:NTPD_DST_PROC.COM p1 p2 p3 p4 p5
```

Where:

- `p1` = Current time zone name - string (e.g. "`EST`" or "`EDT`")
- `p2` = Time zone offset in seconds - integer (e.g. "`-18000`" or "`-14400`")
- `p3` = DST in effect? - boolean ("`Y`", "`N`")
- `p4` = In Twilight Zone? - boolean ("`Y`", "`N`")
- `p5` = Startup or DST change? - string ("`START`" or "`DST`")

`P1`, the Current Time Zone Name, is a string specifying the current name of the local time zone. For North American Eastern Standard Time, this will be "`EST`" in the winter, and "`EDT`" in the summer, when `DST` is active. For time zones that don't do DST, it will always be the zone name.

`P2`, the Time Zone Offset, is a signed integer specifying the offset, in seconds, from UTC for the local zone, at the current time. For North American Eastern Standard Time this is "`-18000`" (-5 hours), for the same zone with DST in effect it is "`-14400`" (-4 hours).

`P3`, the DST flag. This will be "Y" if DST is currently in effect for the zone, and "N" if it isn't, or if the zone doesn't do DST.

`P4`, the Twilight Zone flag. When a zone exits from DST, it sets its time back an hour. This means that for that hour, the time *appears* to be a DST time by the local DST rules, but isn't really, since DST has already ended. That hour is called the "twilight zone" by TCPware NTP. If the current time is in that period, the P4 parameter will be "Y", otherwise it will be "N".

`P5`, the startup/DST flag. This tells the procedure whether it is being called as a part of NTPD's startup processing, or as part of a DST change.

These parameters are provided so that the procedure can take different action under different conditions. They may all be ignored if that is appropriate. The NTPD server doesn't depend on any particular

behavior, so long as the TCPWARE_TIMEZONE logical is left alone and the system clock is not altered. The final completion status of the called procedure will be logged by the NTPD server, along with the PID of the spawned sub-process.

NTP Utilities

There are several utility programs included with NTP. These allow setting the system clock from a time server, querying and controlling NTP servers on the local system or on remote hosts, and tracing the chain of time servers back to the top stratum server being used to set the local time.

NTPDATE

The NTPDATE utility sets the local date and time, by polling the NTP servers given as the server arguments, to determine the correct time. A number of samples are obtained from each of the servers specified and a subset of the NTP clock filter and selection algorithms are applied to select the best of these.

> **Note:** The accuracy and reliability of NTPDATE depends on the number of servers, the number of polls each time it is run, and the interval between runs.

The NTPDATE utility can be run manually as necessary to set the host clock, or it can be run from the system startup command file to set the clock at boot time. This is useful in some cases to set the clock initially before starting the NTP daemon, NTPD. It is also possible to run NTPDATE from a batch job. However, it is important to note that NTPDATE with contrived batch jobs is no substitute for the NTP daemon, which uses sophisticated algorithms to maximize accuracy and reliability while minimizing resource use. Finally, since NTPDATE does not discipline the host clock frequency as does NTPD, the accuracy using NTPDATE is limited.

The NTPDATE utility makes time adjustments in one of two ways. If it determines that the clock is wrong by more than 0.5 second, it simply steps the time by calling the $SETIME system service. If the error is less than 0.5 second, it slews the time by temporarily adjusting system clock variables. The latter technique is less disruptive and more accurate when the error is small, and works quite well when NTPDATE is run by a batch job every hour or two.

The NTPDATE utility declines to set the date if NTPD is running on the same host. When running NTPDATE every hour or two from a batch job, as an alternative to running NTPD, results in precise enough timekeeping to avoid stepping the clock.

Format

```
ntpdate [ -bBdoqsuv ] [ -a key ] [ -e authdelay ] [ -k keyfile ] [ -o
version ] [ -p samples ] [-t timeout ] server [ ... ]
```

Command Line Options

-a *key*

Enables the authentication function and specifies the key identifier to be used for authentication as the argument `key`. The keys and key identifiers must match in both the client and server key files. The default is to disable the authentication function.

-B

Force the time to always be slewed, even if the measured offset is greater than ~128 ms. The default is to step the time if the offset is greater than ~128 ms.

> **Note:** If the offset is large, it can sometimes take several hours to slew the clock to the correct value. During this time, the host should not be used to synchronize clients.

-b

Force the time to be stepped, rather than slewed (default). This option should be used when called from a startup file at boot time.

-d

Enable the debugging mode, in which `ntpdate` will go through all the steps, but not adjust the local clock. Information useful for general debugging will also be printed.

-e *authdelay*

Specify the processing delay to perform an authentication function as the value `authdelay`, in seconds and fraction. This number is usually small enough to be negligible for most purposes, though specifying a value may improve timekeeping on very slow CPU's.

-k *keyfile*

Specifies the path for the authentication key file as the string `keyfile`. The default is `MULTINET:NTP.KEYS`. This file should be in the format described for `NTPD` configuration.

-o *version*

Specifies the NTP version for outgoing packets as the integer version, which can be 1, 2, 3 or 4. The default is 4. This allows NTPDATE to be used with older NTP versions.

-p *samples*
Specifies the number of samples to be acquired from each server as the integer samples, with values from 1 to 8 inclusive. The default is 4.

-q
Query only - don't set the clock.

-s
Enables OPCOM messaging. This is designed primarily for the convenience of batch jobs.

-t *timeout*
Specifies the maximum time waiting for a server response as the value timeout, in seconds and fraction. The value is rounded to a multiple of 0.2 seconds. The default is 1 second, a value suitable for polling across a LAN.

-u
Directs NTPDATE to use an unprivileged port on outgoing packets. This is most useful when behind a firewall that blocks incoming traffic to privileged ports, and you want to synchronize with hosts beyond the firewall.

-v
Be verbose. This option will cause ntpdate's version identification string to be logged.

NTPTRACE

The `NTPTRACE` utility determines where a given NTP server gets its time, and follows the chain of NTP servers back to their master time source. If given no arguments, it starts with `localhost`. Here is an example of the output from `NTPTRACE`:

```
$ ntptrace
localhost: stratum 4, offset 0.0019529, synch distance 0.144135
server2ozo.com: stratum 2, offset 0.0124263, synch distance 0.115784
usndh.edu: stratum 1, offset 0.0019298, synch distance 0.011993, refid
'WWVB'
```

On each line, the fields are (left to right): the host name, host stratum, time offset between that host and the local host (as measured by `NTPTRACE`; this is why it is not always zero for `localhost`), host synchronization distance, and (only for stratum-1 servers) the reference clock ID. All times are given in seconds.

> **Note:** The stratum is the server hop count to the primary source, while the synchronization distance is the estimated error relative to the primary source. The NTP server must be synchronized to a peer.

Format

`ntptrace [-vdn] [-r retries] [-t timeout] [server]`

Command Line Options

`-d`

Turns on some debugging output.

`-n`

Turns off the printing of hostnames; instead, host IP addresses are given. This may be useful if a nameserver is down.

-r *retries*
Sets the number of retransmission attempts for each host. The default is 5.

-t *timeout*
Sets the retransmission timeout (in seconds). The default is 2.

-v
Prints verbose information about the NTP servers.

NTPDC

The NTPDC utility is used to query the NTPD server about its current state and to request changes in that state. The program runs interactively or uses command line arguments. Extensive state and statistics information is available through the NTPDC interface. In addition, nearly all the configuration options that can be specified at startup using NTPD's configuration file may also be specified at run-time using NTPDC.

The NTPDC utility uses NTP mode 7 packets to communicate with the NTP server, and can be used to query any compatible server on the network which permits it.

> **Note:** Since NTP is a UDP protocol, this communication is somewhat unreliable, especially over large distances, in terms of network topology. NTPDC makes no attempt to retransmit requests, and times out requests if the remote host is not heard from within a suitable timeout time.

NTPDC's operation is specific to the NTPD implementation and can be expected to work only with this, and possibly some previous versions, of the daemon. Requests from a remote NTPDC program that affect the state of the local server must be authenticated, which requires both the remote program and local server to share a common key and key identifier.

Command Line Format

`ntpdc [-ilnps] [-c command] [host] [...]`

Command Line Arguments

(If command line arguments are omitted, NTPDC runs in interactive mode.)

-c

The command that follows is interpreted as an interactive format command and is added to the list of commands to be executed on the specified host(s). The command must be in double quotes if it consists of more than one word. Multiple -c options can be given.

-i

Force `ntpdc` to operate in interactive mode. Prompts will be written to the standard output and commands read from the standard input.

`-l`
Obtain a list of peers which are known to the server(s). This switch is equivalent to `-c listpeers`.

`-n`
Displays all host addresses in dotted quad numeric format rather than converting them to canonical hostnames.

`-p`
Print a list of the peers known to the server as well as a summary of their state. This is equivalent to `-c peers`.

`-s`
Print a list of the peers known to the server as well as a summary of their state, but in a slightly different format than the `-p` switch. This is equivalent to `-c dmpeers`.

`host`
Sets the host to which future queries are sent, as either a hostname or a numeric address. If *host* is omitted, the local host is used.

Interactive Commands

Interactive format commands consist of a keyword followed by zero to four arguments. Only enough characters of the full keyword to uniquely identify the command need be typed. The output of a command is normally sent to the standard output, but you can send the output of individual commands to a file by appending a greater than (>) followed by a filename to the command line.

`? [`*`command-keyword`*`]`
`help [`*`command-keyword`*`]`
A question mark (`?`) by itself prints a list of all the known command keywords. A question mark (`?`) followed by a command keyword prints function and usage information.

delay *milliseconds*
Specifies a time interval to be added to timestamps included in requests that require authentication. This is used to enable unreliable server reconfiguration over long delay network paths or between machines whose clocks are unsynchronized.

host [*hostname*]
Sets the host to which future queries are sent. `Hostname` may be either a hostname or a numeric address.

hostnames [yes | no]
If `yes` is specified, host names are printed in information displays. If `no` is specified, numeric addresses are printed instead. The default is `yes`, unless modified using the command line `-n` switch.

keyid [*keyid*]
Allows a key number to be used by `NTPDC` to authenticate configuration requests. This must correspond to a key number the server has been configured to use for this purpose.

quit
Exits `NTPDC`.

passwd
Prompts you to type in a password (which is not echoed) that is used to authenticate configuration requests. The password must correspond to the key configured for use by the NTP server for this purpose if such requests are to be successful.

timeout [*milliseconds*]
Specifies a timeout period for responses to server queries. The default is approximately 8000 milliseconds.

> **Note:** Since `NTPDC` retries each query once after a timeout, the total waiting time for a timeout is twice the timeout value set.

Control Message Commands

Query commands produce NTP mode 7 packets containing requests for information being sent to the server. These are read-only commands in that they make no modification of the server configuration state.

`listpeers`

Obtains and prints a brief list of the peers for which the server is maintaining state. These should include all configured peer associations, as well as those peers whose stratum is such that they are considered by the server to be possible future synchronization candidates.

`peers`

Obtains a list of peers for which the server is maintaining state, along with a summary of that state. Summary information includes the address of the remote peer; local interface address (0.0.0.0 if a local address has yet to be determined); stratum of the remote peer (a stratum of 16 indicates the remote peer is unsynchronized); polling interval (in seconds); reachability register (in octal); and current estimated delay, offset, and dispersion of the peer (all in seconds).

The character in the left margin indicates the mode this peer entry is operating in as per the table:

+	Symmetric active
-	Symmetric passive
=	Remote server is being polled in client mode
^	Server is broadcasting to this address
~	Remote peer is sending broadcasts
*	Peer the server is currently synchronizing to

The contents of the host field may be in one of four forms: a hostname, IP address, reference clock implementation name with its parameter, or `REFCLK` (implementation number, parameter). With `hostnames no`, only IP-addresses are displayed.

dmpeers
A slightly different peer summary list. Identical to the output of the `peers` command, except for the character in the leftmost column. Characters only appear beside peers which were included in the final stage of the clock selection algorithm. Characters indicate server validity according to the following table:

.	peer was cast off in the `falseticker` detection
+	peer made it through `falseticker` detection
*	peer the server is currently synchronizing with

showpeer *peer-address* [...]
Shows a detailed display of the current peer variables for one or more peers. Most of these values are described in the NTP Version 2 specification. Understanding this information will require a detailed understanding of the inner workings of the NTP protocol, which is also available in the RFCs that specify the protocol.

pstats *peer-address* [...]
Shows per-peer statistic counters associated with the specified peer(s).

loopinfo [oneline | multiline]
Prints the values of selected loop filter variables. The loop filter is the part of NTP which deals with adjusting the local system clock.

`loop filter`	is the part of NTP that deals with adjusting the local system clock
`offset`	is the last offset given to the loop filter by the packet processing code
`frequency`	is the frequency error of the local clock in parts per million (ppm)

`time_const`	controls the stiffness of the phase-lock loop and thus the speed at which it can adapt to oscillator drift
`watchdog timer value`	is the number of seconds elapsed since the last sample offset was given to the loop filter

The `oneline` and `multiline` options specify the format in which this information is to be printed, with **`multiline`** as the default.

`sysinfo`
Prints a variety of system state variables, such as the state related to the local server. All except the last four lines are described in the NTP Version 3 specification, RFC 1305.

The system flags can be set and cleared by the **`enable`** and **`disable`** configuration commands, respectively. These are the `auth, bclient, monitor, pll, pps,` and `stats` flags. (See the *NTPD* section for the meaning of these flags.)

The `stability` is the residual frequency error remaining after the system frequency correction is applied, and is intended for maintenance and debugging. In most architectures, this value initially decreases from as high as 500 ppm to a nominal value in the range .01 to 0.1 ppm. If it remains high for some time after starting the server, something might be wrong with the local clock.

The `broadcastdelay` shows the default broadcast delay, as set by the `broadcastdelay` configuration command. The `authdelay` shows the default authentication delay, as set by the `authdelay` configuration command.

`sysstats`
Prints statistics counters maintained in the protocol module.

`memstats`
Prints statistics counters related to memory allocation code.

`iostats`
Prints statistics counters maintained in the input-output module.

timerstats
Prints statistics counters maintained in the timer/event queue support code.

reslist
Obtains and prints the server's restriction list. This list is usually printed in sorted order and may help to understand how the restrictions are applied.

monlist [*version*]
Obtains and prints traffic counts collected and maintained by the monitor facility. You do not normally need to specify the version number.

Runtime Configuration Requests

All requests that cause state changes in the server are authenticated by the server using the `requestkey` in the configuration file (which can be disabled by the server by not configuring a key). The key number and the corresponding key must also be made known to `NTPDC`. This can be done using `NTPDC`'s `keyid` and `passwd` commands, the latter of which prompts at the terminal for a password to use as the encryption key. You are also prompted automatically for both the key number and password the first time a command is given that would result in an authenticated request to the server. Authentication not only provides verification that the requester has permission to make such changes, but also gives an extra degree of protection against transmission errors.

Authenticated requests always include a timestamp in the packet data, which is included in the computation of the authentication code. This timestamp is compared by the server to its receive timestamp. If they differ by more than a small amount, the request is rejected. This is done for two reasons. First, it makes simple replay attacks on the server, by someone who might be able to overhear traffic on your LAN, much more difficult. Secondly, it makes it more difficult to request configuration changes to your server from topologically remote hosts. While the reconfiguration facility works well with a server on the local host, and may work adequately between time synchronized hosts on the same LAN, it works very poorly for more distant hosts. As such, if reasonable passwords are chosen, care is taken in the distribution and protection of keys, and appropriate source address restrictions are applied, the run-time reconfiguration facility should provide an adequate level of security.

The following commands all make authenticated requests.

addpeer *peer-address* [*keyid*] [*version*] [prefer]
Adds a configured peer association at the given address and operates in symmetric active mode.

> **Note:** An existing association with the same peer may be deleted when this command is executed, or may simply be converted to conform to the new configuration, as appropriate. If the optional `keyid` is a non-zero integer, all outgoing packets to the remote server have an authentication field attached, encrypted with this key. If the value is 0 (or not given), no authentication is done. The `version` can be 1, 2, 3 or 4, and defaults to 4. The `prefer` keyword indicates a preferred peer (and thus is used primarily for clock synchronization if possible).

addserver *peer-address* [*keyid*] [*version*] [prefer]
Identical to the `addpeer` command, except that the operating mode is client.

broadcast *peer-address* [*keyid*] [*version*] [prefer]
Identical to the `addpeer` command, except that the operating mode is `broadcast`**.** In this case a valid key identifier and key are required. The `peer-address` parameter can be the broadcast address of the local network, or a multicast group address assigned to NTP. If using a multicast address, a multicast-capable kernel is required.

unconfig *peer-address* [...]
Removes the configured bit from the specified peers. In many cases, this deletes the peer association. When appropriate, however, the association may persist in an unconfigured mode if the remote peer is willing to continue in this fashion.

enable [*flag*] [...]
disable [*flag*] [...]
Operates the same as the `enable` and `disable` configuration file commands of `NTPD`.

restrict address mask flag [flag]
Operates the same as the `restrict` configuration file commands of `NTPD`.

unrestrict address mask flag [flag]
Unrestricts the matching entry from the restrict list.

delrestrict *address mask* [*ntpport*]
Deletes the matching entry from the restrict list.

readkeys
Causes the current set of authentication keys to be purged and a new set to be obtained by rereading the keys file (MULTINET:NTP.KEYS). This allows encryption keys to be changed without restarting the server.

trustedkey *keyid* [...]
untrustedkey *keyid* [...]
Operates the same as the trustedkey and untrustedkey configuration file commands of NTPD.

authinfo
Returns information concerning the authentication module, including known keys and counts of encryptions and decryptions which have been done.

reset
Clears the statistics counters in various modules of the server.

NTPQ

The `NTPQ` utility is used to query NTP servers that implement the recommended NTP mode 6 control message format about current state and to request changes in that state. The program runs interactively or uses command line arguments. Requests to read and write arbitrary variables can be assembled, with output options available. `NTPQ` can also obtain and print a list of peers in a common format by sending multiple queries to the server.

The utility uses NTP mode 6 packets to communicate with the NTP server, and hence can be used to query any compatible server on the network which permits it.

> **Note:** Since NTP is a UDP protocol, this communication is somewhat unreliable, especially over large distances in terms of network topology. `NTPQ` makes one attempt to retransmit requests, and times out requests if the remote host is not heard from within a suitable timeout time.

Command Line Format

`ntpq [ -inp ] [- c` *`command`* `] [` *`host`* `] [ ... ]`

If command line arguments are omitted, NTPQ runs in interactive mode.

`-c`

The `command` that follows is interpreted as an interactive format command and is added to the list of commands to be executed on the specified host(s). The `command` must be in double quotes if it consists of more than one word. Multiple `-c` options may be given.

`-i`

Force `NTPQ` to operate in interactive mode. Prompts will be written to the standard output and commands read from the standard input.

`-n`

Displays all host addresses in dotted quad numeric format rather than converting them to canonical hostnames.

-p
Print a list of the peers known to the server as well as a summary of their state. This is equivalent to the peers interactive command.

host
Sets the host to which future queries are sent, as either a hostname or a numeric address. If `host` is omitted, the local host is used.

Interactive Commands

Interactive format commands consist of a keyword followed by zero to four arguments. Only enough characters of the full keyword to uniquely identify the command need be typed. The output is sent to the standard output.

Internal Commands

Internal commands are executed entirely within the `NTPQ` program itself and do not result in NTP mode 6 requests being sent to a server.

? [*command-keyword*]
help [*command-keyword*]
A question mark (`?`) by itself prints a list of all the known command keywords. A question mark followed by a command keyword prints function and usage information for that command.

addvars *variable_name* [= *value*] [...]
rmvars *variable_name* [...]
clearvars
The data carried by NTP mode 6 messages consists of a list of items of the form `variable_name = value`, where the `= value` is ignored, and can be omitted, in requests to the server to read variables. `NTPQ` maintains an internal list in which data to be included in control messages can be assembled, and sent using the `readlist` and `writelist` commands described below. The `addvars` command allows variables and their optional values to be added to the list. If more than one variable is to be added, the list should be comma-separated and not contain white space. The `rmvars` command can be used to remove individual variables from the list, while the `clearlist` command removes all variables from the list.

authenticate yes | no
Normally NTPQ does not authenticate requests unless they are write requests. The command authenticate yes causes NTPQ to send authentication with all requests it makes. Authenticated requests cause some servers to handle requests slightly differently.

cooked
Causes output from query commands to be "cooked" for user readability. Variables NTPQ recognizes have their values reformatted for readability. Variables that NTPQ determines should have a decodeable value, but do not, are marked with a trailing question mark (?).

debug more | less | off
Turns internal query program debugging on and off.

delay milliseconds
Specify a time interval to be added to timestamps included in requests which require authentication. This is used to enable (unreliable) server reconfiguration over long delay network paths or between machines whose clocks are unsynchronized. The server does not now require timestamps in authenticated requests, so this command may be obsolete.

host [*hostname*]
Sets the host to which future queries are sent. Hostname may be either a hostname or a numeric address.

hostnames [yes | no]
If yes is specified, hostnames are printed in information displays. If no, numeric addresses are printed instead. The default is yes, unless modified using the command line -n switch.

keyid *keyid*
This command allows the specification of a key number to be used to authenticate configuration requests. This must correspond to a key number the server has been configured to use for this purpose.

ntpversion [1 | 2 | 3 | 4]
Sets the NTP version number that NTPQ claims in packets. Default is 4. Mode 6 control messages (and modes) didn't exist in NTP version 1.

quit
Exits `NTPQ`.

passwd
This command prompts you to type in a password (which will not be echoed) which will be used to authenticate configuration requests. The password must correspond to the key configured for use by the NTP server for this purpose if such requests are to be successful.

raw
Causes all output from query commands to be printed as received from the remote server. The only formatting or interpretation done on the data is to transform non-ASCII data into a printable form.

timeout [*milliseconds*]
Specifies a timeout period for responses to server queries. The default is about 5000 milliseconds. Since `NTPQ` retries each query once after a timeout, the total waiting time for a timeout is twice the timeout value set. If the `milliseconds` value is omitted, the current timeout period is displayed.

Control Message Commands

Each peer known to an NTP server has a 16-bit integer association identifier assigned to it. NTP control messages that carry peer variables must identify the peer to which the values correspond by including its association ID. An association ID of 0 is special, and indicates the variables are system variables whose names are drawn from a separate name space.

Control message commands result in one or more NTP mode 6 messages being sent to the server, and cause the data returned to be printed in some format. Most commands currently implemented send a single message and expect a single response. The current exceptions are the `peers` command, which will send a preprogrammed series of messages to obtain the data it needs, and the `mreadlist` and `mreadvar` commands, which will iterate over a range of associations.

associations

Obtains and prints a list of association identifiers and peer status for in-spec peers of the server being queried. The list is printed in columns. The first of these is an index numbering the associations from 1 for internal use, the second is the actual association identifier returned by the server, and the third is the status word for the peer, as described in Appendix B.2.2 of the NTP specification RFC 1305. This is followed by a number of columns containing data decoded from the status word.

clockvar [*assocID*] [*variable_name* [= *value* [...]] [...]
cv [*assocID*] [*variable_name* [= *value* [...]] [...]
Requests that a list of the server's clock variables be sent. Servers which have a radio clock or other external synchronization will respond positively to this. If the association identifier is omitted or zero the request is for the variables of the system clock and will generally get a positive response from all servers with a clock. If the server treats clocks as pseudo-peers, and hence can possibly have more than one clock connected at once, referencing the appropriate peer association ID will show the variables of a particular clock. Omitting the variable list will cause the server to return a default variable display.

lassocations
Obtains and prints a list of association identifiers and peer statuses for all associations for which the server is maintaining state. This command differs from the `associations` command only for servers which retain state for out-of-spec client associations (i.e., fuzzballs). Such associations are normally omitted from the display when the `associations` command is used, but are included in the output of `lassociations`.

lpassociations
Print data for all associations, including out-of-spec client associations, from the internally cached list of associations. This command differs from `passociations` only when dealing with fuzzballs.

lpeers
Like the `peers` command, except a summary of all associations for which the server is maintaining state is printed. This can produce a much longer list of peers from fuzzball servers.

mreadlist *assocID assocID*
mrl *assocID assocID*
Like the `readlist` command, except the query is done for each of a range of (non-zero) association IDs. This range is determined from the association list cached by the most recent `associations` command.

mreadvar *assocID assocID* [*variable_name* [= *value* [...]
mrv *assocID assocID* [*variable_name* [= *value* [...]
Like the `readvar` command, except the query is done for each of a range of (non-zero) association IDs. This range is determined from the association list cached by the most recent `associations` command.

opeers
An old form of the `peers` command with the reference ID replaced by the local interface address.

passociations
Displays association data concerning in-spec peers from the internally cached list of associations. This command performs identically to the `associations` except that it displays the internally stored data rather than making a new query.

peers
Obtains a list of in-spec peers of the server, along with a summary of each peer's state. Summary information includes the address of the remote peer; the reference ID (0.0.0.0 if the ref ID is unknown); stratum of the remote peer; type of the peer (local, unicast, multicast, or broadcast), when the last packet was received; polling interval (in seconds); reachability register (in octal); and the current estimated delay, offset and dispersion of the peer (all in milliseconds). The character in the left margin indicates the fate of this peer in the clock selection process:

character	rv cmd	Description
<space>	`reject`	The peer is discarded as unreachable, synchronized to this server (synch loop) or outrageous synchronization distance.
`x`	`falsetick`	The peer is discarded by the intersection algorithm as a falseticker.
`.`	`excess`	The peer is discarded as not among the first ten peers sorted by synchronization distance and so is probably a poor candidate.
`-`	`outlyer`	Discarded by the clustering algorithm

+	candidat	The peer is a survivor and a candidate for the combining algorithm.
#	selected	The peer is a survivor, but not among the first six peers sorted by synchronization distance. If the assocation is ephemeral, it may be demobilized to conserve resources.
*	peer	The peer has been declared the system peer and lends its variables to the system variables.
o	peer	The peer has been declared the system peer and lends its variables to thesystem variables. However, a PPS signal is in use

> **Note:** Since the `peers` command depends on the ability to parse the values in the responses it gets, it may fail to work with servers that poorly control the data formats.

The contents of the host field may be in one of three forms. It may be a hostname, an IP address, or a reference clock implementation name with its parameter. With `hostnames no`, only IP addresses are displayed.

`pstatus` *`assocID`*
Sends a read status request to the server for the given association. (See the `associations` command for `assocIDs`). The names and values of the peer variables returned are printed. The status word from the header is displayed preceding the variables, both in hexidecimal and in English.

`readlist [` *`assocID`* `]`
`rl [` *`assocID`* `]`
Requests that the values of the variables in the internal variable list be returned by the server. If the association ID is omitted or is 0 the variables are assumed to be system variables. Otherwise they are treated as peer variables. If the internal variable list is empty a request is sent without data, which should induce the remote server to return a default display.

readvar [*assocID* [*variable-name* [= *value*] [...]]]
rv [*assocID* [*variable-name* [=*value*] [...]]]
Requests that the values of the specified variables be returned by the server, by sending a read variables request. If you omit the association ID or give it as zero, the variables are system variables; otherwise they are peer variables and the values returned are those of the corresponding peer. (See the `associations` command for `assocIDs`). Omitting the variable list sends a request with no data, which should induce the server to return a default display. If more than one variable is requested, separate the variable list with commas and do not include spaces.

writevar assocID variable_name [= value [...]
Like the `readvar` request, except the specified variables are written instead of read.

writelist [*assocID*]
Like the `readlist` request, except the internal list variables are written instead of read.

NTP Management

Implementing NTP

To implement NTP:

1. Determine the hosts (peers and servers) with which the local system should negotiate for synchronization.
2. Configure the NTP configuration file by adding these peers and servers to it.
3. Start the NTPD daemon running.
4. Use one of the query programs to verify proper operation.

Modifying the NTP Configuration File

To configure only NTP, you can enter **`@TCPWARE:CNFNET NTP`**. Then add entries to the `TCPWARE:NTP.CONF` peer configuration file. The commands you add to the file have the syntax and meaning described in the previous section on the `NTP.CONF` file.

Monitoring

NTP includes a comprehensive monitoring facility suitable for continuous, long term recording of server and client timekeeping performance. (See the `statistics` configuration option for a listing of each type of statistic currently supported.)

Troubleshooting Tips

Here are some troubleshooting tips:

- Make sure the entries in the NTP configuration file `TCPWARE:NTP.CONF` are correct. At the minimum, there must be a server or peer declaration for a machine that is reachable, and if authentication is enabled, set it up to properly authenticate NTP packets. The machine serving time must be connected either to lower stratum machines or to some reference time source.
- Make sure that the logical `TCPWARE_TIMEZONE` is properly defined to reflect the time zone (and daylight savings). If the logical is undefined or incorrect, NTP is likely to abort. `TCPWARE_TIMEZONE` should be set by configuring TCPware with the configuration procedure `TCPWARE:CNFNET.COM`.
- If using the `slewalways` command, make sure the system time is within 4000 seconds (or whatever panic is set to) of the correct time before starting NTPD. If the local system time is off

by more than this amount from server time, NTPD logs a message and stops running. Also, if the local clock is not within a minute or two of correct time when starting NTPD with `slewalways` set, it may take some time for NTPD to synchronize the clock. Ideally, set the clock with `NTPDATE` or `SET TIME` before starting NTPD.

- Make sure that TIMED and DTSS services are not running on the system. These services are used to synchronize time and interfere with NTP unless NTP was configured in special cases to work with them. (See the `master-clock` command.)
- The following messages are generated by the NTP server. They go to both OPCOM and the `TCPWARE:NTPSERVER.LOG` file. This log file is the best source of information for troubleshooting in that it contains a record of these messages as well as additional informational messages. Messages appear in the log file without the bracketed prefix. There are four types of messages generated:
    - Configuration messages
    - Peer contact messages
    - Synchronization messages
    - Unexpected error condition messages

Access error messages help by entering:

```
$ HELP TCPWARE MESSAGES
```

Troubleshooting Using NTPQ

The `NTPQ` utility has a few commands that are helpful in identifying problems. The `peers` command is one of the simplest and is a quick way to check the offset (time difference) between the local host and peer machines.

The `readvar` command is useful for more in depth information. Without arguments, it displays information about the local host. When `readvar` is followed by an `assocID`, it displays information about the peer corresponding to the `assocID` (use `associations` to display the `assocIDs` for all peers). Of interest is the record of time offsets and round-trip delays for packets (the `filtoffset` and `filtdelay` fields). This provides a record of the last eight time updates obtained from a peer.

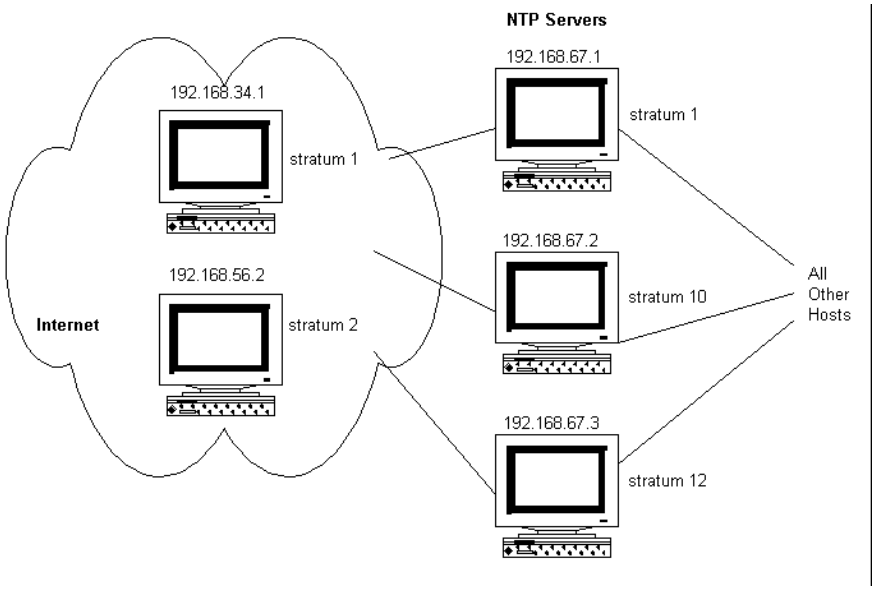
The command `readvar assocID` flash displays a useful variable, `flash`, which can be of particular interest for troubleshooting. The bits in the `flash` variable, if set, have the following meaning in relation to a peer:

```
0x01    /* duplicate packet received */
0x02    /* bogus packet received */
0x04    /* protocol unsynchronized */
0x08    /* peer delay/dispersion bounds check */
0x10    /* peer authentication failed */
0x20    /* peer clock unsynchronized */
```

```
0x40    /* peer stratum out of bounds */
0x80    /* root delay/dispersion bounds check */
```

Configuration Example

The diagram below shows a highly redundant and robust configuration with multiple levels of backups. On the Internet close to your network, you have host 192.168.34.1 running at stratum 1, and 192.168.34.2 at stratum 2. In-house, you have host 192.168.67.1 synchronized with a radio clock and configured as a stratum 1 master clock.



As backup servers, you have two hosts, 192.168.67.2 and 192.168.67.3, in the climate-controlled room, one configured at stratum 10 and the other at 12. All other workstations on the floor point to these three servers as their synchronization source. When everything is running, every local host is synchronized to 192.168.67.1, since it is closer than Internet host 192.168.34.1. All the machines (peers) run at stratum 2.

If internal host 192.168.67.1 goes down and the Internet connection is still up, either Internet host 192.168.34.1 or 192.168.34.2 is selected depending on its availability, and the backup servers, 192.168.67.2 and 192.168.67.3, run at stratum 2 or 3, depending on which Internet host was selected. The peers synchronize off 192.168.67.2 or 192.168.67.3 at stratum 3 or 4, again depending on which Internet host was selected.

With 192.168.67.1 still unavailable and the Internet connection lost or all the Internet servers unavailable, 192.168.67.2 runs at stratum 10, since it was configured that way as a local clock. It then

becomes the lowest stratum number in the network and all other hosts (including 192.168.67.3) are synchronized to it at stratum 11.

If 192.168.67.2 goes down, 192.168.67.3 runs at stratum 12 and all other hosts synchronize at stratum 13. It is important to set the stratum of 192.168.67.3 to 12. If set to 11, it might have a problem synchronizing to 192.168.67.2, since it may try to synchronize off it but finds it has the same stratum value. 192.168.67.3 would rather synchronize to 192.168.67.2 than to itself.

The below example shows the configuration file entries for each of the three local servers (the other local hosts would all be configured as peers). You do not need to explicitly identify the peer strata, and the order of items is irrelevant.

```
; NTP Configuration on 192.168.67.1
master-clock 1

; NTP Configuration on 192.168.67.2
local-master 10
server 192.168.67.1
server 192.168.34.1
server 192.168.34.2
peer 192.168.67.3


; NTP Configuration on 192.168.67.3
local-master 12
server 192.168.67.1
server 192.168.34.1
server 192.168.34.2
peer 192.168.67.2


; NTP Configuration for Computer Room Host 192.168.67.x
server 192.168.67.1
server 192.168.67.2
server 192.168.67.3
peer 192.168.67.y
peer 192.168.67.z
.
.
.
```

11. Managing FTP

Introduction

This chapter describes management of the TCPware FTP client and server. Topics include:

- Client considerations
- Server security considerations
- Including messages on client login
- ANONYMOUS support
- Server logicals
- Server FTP protocol implementation

Client Considerations

Client considerations include creating a startup command file and determining the status on an exit condition.

Startup Command File

You can create a system-wide startup file that is executed each time the local client starts an FTP session. To create a startup command file, you need to:

1. Create a file containing the FTP commands you want performed at the beginning of each FTP session. For example:

   ```
   $ CREATE SYS$COMMON:[SYSMGR]FTP_STARTUP.COM
   OPEN IRIS SMITH "Sandy"
   SHOW STATUS
   Return
   $ SET PROTECTION=WORLD:RE SYS$COMMON:[SYSMGR]FTP_STARTUP.COM
   ```

   The `SET PROTECTION` command ensures that the client user can read and execute the `FTP_STARTUP.COM` file.

2. Define the `FTP_STARTUP` logical to point to the `FTP_STARTUP.COM` file. For example:

   ```
   $ DEFINE/SYSTEM/EXEC FTP_STARTUP SYS$MANAGER:FTP_STARTUP.COM
   ```

Client users can override this startup file by creating their own. Including the command DEFINE/PROCESS FTP_STARTUP in a user's LOGIN.COM file overrides any DEFINE/SYSTEM/EXEC command in the SYS$MANAGER:SYSTARTUP_V5.COM file.

3. The FTP configuration consists of these questions:

```
Configuring FTP-OpenVMS:
Do you want to enable the FTP server? Y
Do you want to enable FTP accounting? Y
Name of host that will run accounting collection program:
bigboote.example.com
Port number that accounting collection program listens on: 2222
```

   The last two questions are optional depending on the value of the second question.

See the *User's Guide*, Chapter 3, *FTP: Transferring Files*, *Startup Command File* about setting up a client-specific FTP_STARTUP.COM file.

SET DEBUG /CLASS=REPLIES (VERBOSE mode) is enabled by default in TCPware's FTP client so that you do not need to explicitly add the command to the file. However, any VERBOSE command may toggle it to OFF.

See the *User's Guide*, Chapter 3, *FTP: Transferring Files*, about setting VERBOSE mode. See your OpenVMS documentation about the SET PROTECTION command.

Status on Exiting FTP Status

To exit FTP, use the EXIT command or type **Ctrl+Z**.

FTP exits with the last error status, if any. DCL command procedures can use the $STATUS and $SEVERITY symbols to test for success or failure of the FTP commands issued. A success status indicates that all commands succeeded. A warning, error, or severe status indicates that one or more commands failed to execute.

When possible, the status code is a System Service (defined in $SSDEF), RMS (defined in $RMSDEF), or shared (defined in $SHRDEF) status. In some cases, status codes are TCPWARE_ private codes with a facility number of 1577.

Server Security

The FTP server provides security through access restrictions on which IP addresses can connect, login procedures, the use of log files, and the automatic termination of idle control connections.

Incoming Access Restrictions

Access control lists for FTP can be defined in `TCPWARE:FTP.CONF`. The below table lists the commands that can be in this file.

Command	Description
`ALLOW` *ip-address* [*mask*]	Connections that match the specified IP address and optional mask are accepted. The default is to allow connections from all IP addresses.
`DENY` *ip-address* [*mask*]	Connections that match the specified IP address and optional mask are rejected. The default is to deny connections from no IP addresses.
`MAX_SERVERS`	The maximum number of servers. This command overrides the existing logical if it is used.
`REJECT_BY_DEFAULT`	Connections that do not match an IP address mentioned in an `ALLOW` statement are rejected.
`REJECT_MESSAGE`	The message to be displayed before rejecting the connection. The default is "`You are not allowed to access this server.`"
`REQUIRE_TLS YES`	Specifies that user authentications other than anonymous and users that have no password must use TLS authentication.
`RFC4217_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 and higher you can use `@SSL$COM:SSL$CERT_TOOL`.
`RFC4217_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.
STATISTICS ON	Enable SNMP usage statistics in processSoftware.2.21.1 to .17. This requires that the SNMP Agent be running on the system and that Agent X support is enabled.

Lines in the file that start with ! (exclamation point) or # (sharp) are considered comments.

Tips on Using the FTP Configuration File Commands

- Keep the file short, as each line in the file causes additional startup time for the server.
- If the REJECT_BY_DEFAULT keyword is used in FTP.CONF, comment out any DENY lines that are redundant.
- If using the *mask* option, comment out any duplicate ALLOW or DENY lines. For example, if you have the line "ALLOW 26.26.26.100 255.255.255.128", then you do not need the line "ALLOW 26.26.26.*x*" (where *x* is 0 to 127), as the mask already covers that host.
- Only printable characters are allowed in the REJECT_MESSAGE message, and the message size must be less than 255 bytes.

Following is an example of an FTP.CONF file:

```
REJECT_BY_DEFAULT
    ! but allow
    allow 127.0.0.1
    allow 123.123.95.1 255.255.255.128
     allow 123.123.140.101 255.255.255.192 # blabla
     allow 123.123.142.44
     allow 123.123.142.42  # Test1
     allow 123.123.142.50
     allow 123.123.142.51
     allow 123.123.142.52
     !still deny
     deny 123.123.95.2
REJECT_MESSAGE "Access denied. Ask the network administrator for
permission."
```

Configuring the FTP server for TLS

Follow these steps to configure the TCPware FTP server to allow TLS authentication:

1. Generate or obtain certificate and key files. On OpenVMS V8.3+ SSL$COM:SSL$CERT_TOOL can be used to do this.

2. Place the certificate and key file where you want them and verify that the protection is set such that world has no access.
3. Edit `TCPWARE:FTP.CONF` and add the `RFC4217_CERTIFICATE` and `RFC4217_KEY` commands to specify the location of the certificate and key files.
4. Optionally add `REQUIRE_TLS` command to `TCPWARE:FTP.CONF`
5. Restart the FTP server with `@TCPWARE:RESTART FTP`

Login Procedures

The FTP server uses the same login procedures as DECnet network connections and does not support OpenVMS accounts with two passwords.

Using FTP Log Files

The TCPware FTP server keeps a log of all FTP transactions that occur after login between the client and the server in an `FTPSERVER_DTP.LOG` file in the user's login directory on the server system. A log file is created for each FTP client session. A new log is written when a new session starts, and you can specify a number of log files to retain. The default is to retain only the latest. Users can change this value by defining it in their `LOGIN.COM` file, or it can be defined on a system-wide basis if this is desired for all users.

> **Note:** If the TCPware FTP server process does not start or mysteriously disappears, examine the beginning of `FTPSERVER_DTP.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 `FTPSERVER_DTP.LOG` file.

`TCPWARE:FTPSERVER_DTP.COM` runs when an FTP user logs in. It does the following things:

1. Purges the old versions of `SYS$LOGIN:FTPSERVER_DTP.LOG`, keeping `TCPWARE_FTP_SERVER_LOG_LIMIT` versions of the file.
2. If the logical `TCPWARE_FTP_SET_DEFAULT_TO_ROOT` is defined, it sets the default to `TCPWARE_FTP_`*`username`*`_ROOT` (or `TCPWARE_FTP_ROOT`). If `TCPWARE_FTP_SET_DEFAULT_TO_ROOT` is not defined, the user is not changed. This is traditional TCPware behavior.
3. Defines logicals about the remote connection:

- defines TCPWARE_FTP_GET_REMOTE_INFO, then runs TCPWARE:FTP_SERVER (it exits immediately afterwards)
- TCPWARE_FTP_ADDRESS = *remote host IP address*
- TCPWARE_FTP_HOSTNAME = *remote host name*
- TCPWARE_FTP_LOCAL_ADDRESS = *local host IP address*
- TCPWARE_FTP_ANONYMOUS_PASSWORD (if logging in as ANONYMOUS, this is the string the anonymous user supplied for the password).
- Deassigns TCPWARE_FTP_GET_REMOTE_INFO.

> **Note:** The values of these logicals can be used to control later aspects of the FTP session by adding DCL commands after this point.

4. Sets the default values for the following logicals if they are not defined:
   TCPWARE_FTP_SEMANTICS_VARIABLE_IGNORE_CC TRUE
   TCPWARE_FTP_UNIX_STYLE_CASE_INSENSITIVE TRUE
   TCPWARE_FTP_DISALLOW_UNIX_STYLE TRUE
5. Starts the FTP service program TCPWARE:FTP_SERVER.

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

```
------------------------------------------------------
FTP Login request received at Mon Jun 14 19:05:10 2020
from remote IP address 127.0.0.1
------------------------------------------------------
>>> 230 User HOLMES logged into U1:[HOLMES] at Mon 07-Jun-2020 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-2020 19:05:23.08
```

By setting the logical name TCPWARE_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.

> **Note:** If you are not going to add the TCPWARE_FTP_SERVER_LOG_LIMIT logical to your login.com file but you want to make this a system-wide logical, use the following syntax:
>
> ```
> define/system/exec tcpware_ftp_server_log_limit 42.
> ```

Directory size restrictions limit the number of potential files that can 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 TCPWARE_FTP_SERVER_LOG_LIMIT 42
```

When TLS authentication is used the FTP server will create an additional log file in TCPWARE_SPECIFIC:[TCPWARE.FTPS]FTPS.LOG. These files contain status information from the process that controls the encrypted command channel.

Directory Access Restrictions

The FTP server lets you define three logicals for access restrictions to specific directory trees. These include the TCPWARE_FTP_ROOT logical for system-wide access restrictions, the TCPWARE_FTP_ANONYMOUS_ROOT logical for ANONYMOUS user access restrictions, and the TCPWARE_FTP_*username*_ROOT logical for specific username access restrictions. See *Server Logicals* and *ANONYMOUS Support*.

Log File

The FTP server creates a log file in your default directory each time a client user successfully logs in. This FTPSERVER_DTP.LOG file contains information about files transferred during the FTP session. If client users have problems logging in and are sure they specified a proper username and password,

you can check the SYLOGIN and user account login command procedures for commands that could have caused the login to fail.

Examining the FTPSERVER_DTP.LOG file might help isolate the problem. You may need to execute some operations only if the process mode is interactive. (Use the F$MODE() lexical function to determine the mode and then skip around the offending commands if not an interactive login.) The FTP server runs in network mode.

If you suspect break-in attempts, you can also define the TCPWARE_FTP_LOGFILE system logical to specify the name of a log file.

Idle Control Connection Timeout

If the control connection (other than during a data transfer) is idle for more than 10 minutes, the FTP server aborts the connection, unless you change the idle timeout value using the TCPWARE_FTP_IDLE_TIMEOUT logical.

SNMP Network Service Monitoring

Partial support for RFC 2789 (Network Services Monitoring MIB) has been added to FTP. This feature is enabled with the STATISTICS ON command in the TCPWARE:FTP.CONF file. Information is maintained only while the service is active. The following items from the Network Services Monitoring MIB (RFC 2789) are available in the enterprises.105.2.21 MIB:

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 is the banner that gets 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
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. 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 7 for more information on SNMP and Agent X. This information can be displayed with the NETCU SHOW SNMP MIB_VARIABLE command.

Session Accounting

TCPware 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 from @TCPWARE:CNFNET ACCOUNTING and reads TCPWARE:ACCOUNTING.CONF for additional

configuration information. The format of the accounting records is described in TCPWARE_ROOT:[TCPWARE.EXAMPLES]ACCOUNTING.H

A sample program using this is in TCPWARE_ROOT:[TCPWARE.EXAMPLES]ACC_DUMP.C

You must configure FTP and session accounting to activate the accounting function.

To configure FTP, do the following:

```
$ @TCPWARE:CNFNET FTP
The following information displays on your screen:
TCPware(R) for OpenVMS Version 6.1 Network Configuration procedure for:
        TCP/IP Services:
                FTP-OpenVMS
                NFS-OpenVMS Client
                NFS-OpenVMS Server
                SMTP-OpenVMS
                TELNET-OpenVMS
                Kerberos Services
                SSH-OpenVMS Server

This procedure helps you define the parameters needed to get
TCPware(R) for OpenVMS running on this system.

This procedure creates the configuration data file,
TCPWARE_SPECIFIC:[TCPWARE]TCPWARE_CONFIGURE.COM, to reflect your
system's configuration.

Type <return> to continue...

Configuring FTP-OpenVMS:
Do you want to enable the FTP server [YES]: Return
Do you want to enable FTP accounting [NO]: YES
Name of host that will run accounting collection program
[construction.example.com]: RETURN
Port number that accounting collection program listens on []: 2222

Do you want to restart FTP-OpenVMS [NO]: YES

Shutting down FTP-OpenVMS ...
Starting FTP-OpenVMS ...
%RUN-S-PROC_ID, identification of created process is 00000195
$
```

Configuring Session Accounting

To configure Session Accounting, follow these steps:

1. Edit the ACCOUNTING configuration file.

2. Invoke CNFNET to enable and start ACCOUNTING:

```
$ @TCPWARE:CNFNET ACCOUNTING
```

Configuration File

The Accounting configuration file is TCPWARE: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.

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

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 TCPWARE: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 TCPWARE: device is assumed if a device is not specified as part of the file specification.

Enabling the Session Accounting Facility

```
$ @TCPWARE:CNFNET ACCOUNTING
```

Configuring the Accounting listener

TCPware accounting consists of two components: The accounting record logger, which this procedure configures and controls, and the services that can use the accounting process.

This procedure controls the startup of the accounting record logger. The details such as the name of the accounting file, the port that the accounting record logger listens on, and the list of IP addresses that can use the accounting logger are controlled by `TCPWARE:ACCOUNTING.CONF`:

```
Do you want to activate the Accounting listener on this host [NO]: Y
```

Displaying the Contents of the Logging File

To view accounting information, do the following:

```
$ TCPWARE ACCOUNTING/INPUT=accounting_data_file [/output=output_file] -
_$[/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 (e.g. 1999, 2021).
    - *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 the same as for *start_date*. 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.

Accounting file record format

The accounting file is written using OpenVMS RMS records. The format of these records is defined in `TCPWARE_ROOT:[TCPWARE.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];   /* VMS time that session started */
    int  end_time[2];     /* VMS 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
```

Special Messages

You can include special informational text messages in a specified file in directories so that the message appears when an FTP client user logs in or changes to that directory. The TCPWARE_FTP_MESSAGE_FILE logical determines the filename to check in each directory. This feature is particularly helpful for ANONYMOUS FTP client users to get informational messages when changing directories (see the next section for a description of ANONYMOUS support).

For example, the FTP_CONTROL.COM file that you activate on startup includes the following line commented-out:

```
$! DEFINE/SYSTEM/NOLOG/EXEC TCPWARE_FTP_MESSAGE_FILE ".MESSAGE"
```

You can activate this logical with the default .MESSAGE file definition, or change it to WELCOME.TXT, for example. What the file should contain depends on the context. If the file is in a login directory, it should contain a general message about the system, such as Welcome to OpenVMS AXP v8.2 (IRIS). If the file is in another directory to which the user can move, it should have specifics about the directory, such as Welcome to the SUPPORT directory. It contains TECHNICAL SUPPORT information.

The FTP client user must set VERBOSE mode to be able to see the messages. VERBOSE (REPLIES) mode is set by default in TCPware's FTP client.

The FTP command line client interprets the exclamation point (!) as the start of a comment. To send an exclamation point to the server it must be enclosed in quotes ("). The quote character (") must be doubled up in the string if it is to be sent to the server. For example:

```
FTP>"! send this string to the server."
```

sends the whole line, including the exclamation point and the period.

The FTP command line client prompts with the name of the opened node when the TCPWARE_FTP_PROMPT_NODENAME logical is defined.

ANONYMOUS Support

The FTP server provides special support for ANONYMOUS accounts.

To set up an ANONYMOUS account, issue commands using the OpenVMS AUTHORIZE utility:

```
$ AUTHORIZE
UAF>ADD ANONYMOUS/PASSWORD=GUEST/UIC=[uic] -
_UAF>[/other qualifiers] /NOPWDEXP/NOPWDLIFE
UAF>MODIFY ANONYMOUS/NOLOCAL/NOBATCH/NOREMOTE/NODIALUP
UAF>MODIFY ANONYMOUS/PRIV=NONETMBX/DEFPRIV=NONETMBX
```

The /NOPWDEXP and /PWDLIFE=NONE qualifiers ensure that the password remains active indefinitely. The /NOLOCAL, /NOBATCH, /NOREMOTE, and /NODIALUP qualifiers prevent access to the account from those sources. Removing the NETMBX privilege prevents DECnet access.

See your OpenVMS documentation for details on the AUTHORIZE utility.

The FTP server can recognize other names as anonymous names as well if the logical TCPWARE_ANONYMOUS_USERNAMES is defined. If the name "ANONYMOUS" is still desired, then it should be included in the definition:

```
$ DEFINE TCPWARE_ANONYMOUS_USERNAMES "ANONYMOUS,GUEST"
```

FTP users have automatic read access through ANONYMOUS accounts. You can also assign write, rename, or delete access by defining the TCPWARE_FTP_ANONYMOUS_RIGHTS logical.

Be aware of the following:

- You must create the ANONYMOUS account to use the password GUEST. If the account has any other password, users cannot log in. If the account has another password, use the MODIFY ANONYMOUS /PASSWORD=GUEST/NOPWDEXP command in the AUTHORIZE utility to change it.
- It is recommended that the ANONYMOUS account use the rooted logical (such as ANONYMOUS_ROOT) to point to the top level directory and set the default to ANONYMOUS_ROOT:[000000]. In this way, when a client changes the directory to a slash (/) (or uses a slash in front of a directory specification, as is the case with some web browsers), SYS$DISK:[000000] maps to this root directory. (Note that this mapping is independent of the access restrictions applied by the TCPWARE_FTP_*_ROOT logicals.)
- If you define the TCPWARE_FTP_ANONYMOUS_ROOT logical (or the TCPWARE_FTP_ROOT logical on a system-wide level), the system restricts ANONYMOUS user access to files in the root directory and subdirectories only. When the ANONYMOUS user logs in, the root directory becomes the default.

- In response to a username, the system sends a reply message of `331 Send ident as Password` instead of the usual `331 Password required`.
- If you define the `TCPWARE_FTP_ANONYMOUS_230_REPLY` logical, the system uses it to generate the reply message when the ANONYMOUS user logs in. If you do not define this logical, the system uses the default 230 reply instead.
- Using a hyphen (−) as the first character of the password causes the system to turn off the message generated by the logicals. The system sends the default 230 reply instead.
- If you define the `TCPWARE_FTP_LOGFILE` logical, the server writes a record to the log file:

```
date/time ANONYMOUS FTP login successful (password) from ia, port
```

Note: You must restart FTP after setting this logical by issuing this command:

```
$ @tcpware:restart ftp
```

Each command is logged with the following format:

```
date/time ANONYMOUS FTP user (password) at ia, job 120 reply to user
command job 120 PWD
```

The response to each command is logged with the following format:

```
date/time ANONYMOUS FTP user (password) from (internet address), job
(pid), (reply string)
```

or

```
date/time user name from (internet address), job (pid), (reply string)
```

- `SITE SPAWN` is not a valid command with ANONYMOUS FTP.

Server Logicals

The FTP server supports the following special system-definable logicals.

> **Note:** With the root logicals (TCPWARE_FTP_ROOT, TCPWARE_FTP_ANONYMOUS_ROOT, and TCPWARE_FTP_*username*_ROOT), any logical you refer to in the equivalence name (such as a disk name) must also be an executive mode, system table logical. With all of these logicals, if the user account cannot access the directory, FTP operations will fail with the error %RMS-E-PRV.

TCPWARE_FTP_220_REPLY

The TCPWARE_FTP_220_REPLY logical defines a message displayed when a user connects to the server and can log in. This message replaces the default message.

You can define lines of the message text, one comma-separated equivalence string for each line. You can also specify a file that contains the message text by defining an equivalence string starting with the at-sign (@) and followed by the complete file specification. For example, you can define the welcome text equivalence string as follows:

```
$ DEFINE/SYSTEM/EXEC TCPWARE_FTP_220_REPLY -
_$ "**AUTHORIZED USE ONLY **",-
_$ "bart.example.com (192.168.34.56)", -
_$ "TCPware FTP (c) Process Software"
```

Alternately, you can include the last three equivalence strings in an FTP_WELCOME.TXT file and define the logical as follows:

```
$ DEFINE/SYSTEM/EXEC TCPWARE_FTP_220_REPLY -
_$ "@SYS$MANAGER:FTP_WELCOME.TXT"
```

In either case, when a user connects to a host, the message appears as follows:

```
220-** AUTHORIZED USE ONLY **
220-bart.example.com (192.168.34.56)
220 TCPware FTPD (c) Process Software
_Username []:
```

TCPWARE_FTP_221_REPLY

The TCPWARE_FTP_221_REPLY logical defines a message to appear when a user ends the FTP session. If you do not define this logical, TCPware uses the default message instead. As with TCPWARE_FTP_220_REPLY, you can define a text string or file. For example:

```
$ DEFINE/SYSTEM/EXEC TCPWARE_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
```

TCPWARE_FTP_230_REPLY

The TCPWARE_FTP_230_REPLY logical defines a message to appear when a user successfully logs in. If you do not define this logical, TCPware uses the default message instead. As with TCPWARE_FTP_221_REPLY, you can define a text string or file. For example:

```
$ DEFINE/SYSTEM/EXEC TCPWARE_FTP_230_REPLY -
_$ "Login successful"
```

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

```
230 Login successful
```

TCPWARE_FTP_421_REPLY

The TCPWARE_FTP_421_REPLY logical defines a message sent when a user connects 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. Be sure to deassign the logical after this period to allow FTP access again. As with TCPWARE_FTP_230_REPLY, you can define a text string or file. For example:

```
$ DEFINE/SYSTEM/EXEC TCPWARE_FTP_421_REPLY -
_$ "System maintenance in progress until 17:30"
```

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

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

> **Note:** The TCPWARE_FTP_421_REPLY logical has precedence over the TCPWARE_FTP_220_REPLY logical.

TCPWARE_FTP_ACCESS

If the SYSTEM logical TCPWARE_FTP_ACCESS or TCPWARE_FTP_*username*_ACCESS (to specify a particular username) is defined to any combination of NOLIST, NOWRITE, NOREAD, NOSPAWN, or NODELETE, then the FTP server will not allow the specified actions.

TCPWARE_FTP_ALL_VERSIONS

The logical name TCPWARE_FTP_ALL_VERSIONS requests the NLST and LIST commands to display all versions of the specified files. If TCPWARE_FTP_ALL_VERSIONS is defined as TRUE, the logical name TCPWARE_FTP_STRIP_VERSION has no effect. If this logical is defined as FALSE, NO, or 0 (zero), only the latest version of files in VMS-style directories displays. The default is to display all versions in VMS-style directories.

> **Note:** TCPWARE_FTP_ALL_VERSIONS is ignored if the FTP server is in UNIX emulation mode.

TCPWARE_FTP_ALLOWCAPTIVE

By default, the FTP server does not allow file transfers for CAPTIVE accounts. However, by defining the TCPWARE_FTP_ALLOWCAPTIVE logical, you can allow CAPTIVE accounts to use all FTP commands except SITE SPAWN. Define the logical as follows:

```
$ DEFINE/SYSTEM/EXEC TCPWARE_FTP_ALLOWCAPTIVE " "
```

You must also modify the CAPTIVE account procedure to allow the FTP server to start the data transfer process. The procedure can check if the logical TT is equal to TCPWARE:FTPSERVER_DTP.COM and exit out of the login procedure, as follows:

```
$! Check if this is the TCPware FTP data transfer process:
$ IF F$LOGICAL("TT") .EQS. "TCPWARE:FTPSERVER_DTP.COM" THEN EXIT
$! Refuse other network connections (such as DECnet):
$ IF F$MODE() .EQS. "NETWORK" THEN LOGOUT
$! (or allow by using "...THEN EXIT" above)
$! Remainder of CAPTIVE procedure follows:
$....
```

TCPWARE_FTP_ANONYMOUS_230_REPLY

The TCPWARE_FTP_ANONYMOUS_230_REPLY logical defines a message to appear when an ANONYMOUS user successfully logs in. If you do not define this logical, TCPware uses the default message instead. As with TCPWARE_FTP_230_REPLY, you can define a text string or file. For example:

```
$ DEFINE/SYSTEM/EXEC TCPWARE_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
```

TCPWARE_FTP_ANONYMOUS_RIGHTS

The TCPWARE_FTP_ANONYMOUS_RIGHTS logical lets you define write, rename, and delete access rights for the ANONYMOUS FTP user in addition to read access. For example:

```
$ DEFINE/SYS/NOLOG/EXEC TCPWARE_FTP_ANONYMOUS_RIGHTS -
_$ "WRITE,RENAME,DELETE"
```

WRITE	Lets you PUT, COPY, SEND, APPEND, and MPUT files into the ANONYMOUS FTP area. It also allows execution of the CREATE/DIRECTORY command.
RENAME	Lets you rename files in the ANONYMOUS FTP area.
DELETE	Lets you delete files and directories from the ANONYMOUS FTP area.

The definition of these rights does not override the actual file protections. If a directory does not allow write access, users cannot write to the directory even though the TCPWARE_FTP_ANONYMOUS_RIGHTS logical grants them write access. Likewise, if a file does not allow delete access, users cannot delete it even if the TCPWARE_FTP_ANONYMOUS_RIGHTS logical grants them delete access.

TCPWARE_FTP_ANONYMOUS_ROOT

The TCPWARE_FTP_ANONYMOUS_ROOT (system level, executive mode) logical defines access restrictions for users logged in as ANONYMOUS. For example, you can set access restrictions for users

logged in as ANONYMOUS to allow access to just the ANONYMOUS$USER directory and its subdirectories, as follows:

```
$ DEFINE/SYSTEM/EXEC TCPWARE_FTP_ANONYMOUS_ROOT ANONYMOUS$USER:
```

If you do not set this logical, the FTP server defaults to the setting in the TCPWARE_FTP_ROOT logical (described above) if it exists.

TCPWARE_FTP_CONNECT_BANNER

This logical limits the information given out on connection or when using the STAT command

```
$ DEFINE/SYSTEM/EXEC TCPWARE_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.

TCPWARE_FTP_DISALLOW_UNIX_STYLE

This logical controls whether UNIX-style filename parsing is done. The default value for TCPWARE_FTP_DISALLOW_UNIX_STYLE is TRUE, 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 (TCPware is one of them) you will have to enclose the directory specification in quotes (" ") when it contains a slash (/) 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 TCPWARE_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 TCPWARE_FTP_DISALLOW_UNIX_STYLE FALSE
```

and

```
$ DEFINE TCPWARE_FTP_UNIX_STYLE_BY_DEFAULT ANYTHING
```

TCPWARE_FTP_DISALLOW_WILDCARD_DELETES

If this logical is defined (any value) then the FTP server will not allow wildcard deletes to be done.

TCPWARE_FTP_DONT_REPORT_FILESIZE

If this logical is defined, the reporting of the estimate of the number of bytes to be transferred in the 150 response line is suppressed. Some FTP clients expect this number to be exact. The FTP server is unable to determine an exact count without processing the entire file, so an estimate of the number of bytes used to store the file is returned. The inaccuracy comes from the differences in the way OpenVMS records and line breaks are handled.

```
$ DEFINE/SYSTEM/EXEC TCPWARE_FTP_DONT_REPORT_FILESIZE TRUE
```

TCPWARE_FTP_EXTENSION_QUANTITY

Defines the default allocation /extension quantity of blocks for new files and appends. See FAB$W_DEQ in the *OpenVMS Record Management Services Manual* for an explanation of the effect of this.

```
$ DEFINE/SYSTEM/NOLOG/EXEC TCPWARE_FTP_EXTENSION_QUANTITY n
```

TCPWARE_FTP_GETHOST_MAX_TIME

When a new connection arrives at the FTP server it attempts to resolve the name of the host that originated the connection. If this process takes a long time, it can stall all other connections, both active and new. To adjust how long the FTP server is allowed to take to look up the host name, set the logical TCPWARE_FTP_GETHOST_MAX_TIME to the VMS delta time that can elapse before it gives up. The default value is 10 seconds (0 0:0:10).

TCPWARE_FTP_IDLE_TIMEOUT

If you want to change the timeout for FTP connection attempts to something other than the default of 10 minutes, use the TCPWARE_FTP_IDLE_TIMEOUT system logical. The FTP server checks the timeout when you enter and complete a command. Therefore, you can set this logical at any time, and it effectively changes the idle timeout for open, non-idling connections as well as for any future ones. Make sure to use delta time for the time syntax. For example:

```
$ DEFINE/SYSTEM/EXEC TCPWARE_FTP_IDLE_TIMEOUT "0 00:20:00"
```

This example changes the idle timeout to 20 minutes. If omitted, the default is 10 minutes. If you set the value to 0, idle timeout is disabled.

Disabling idle timeout is not recommended as it creates a potential denial of service security problem.

TCPWARE_FTP_IGNORE_UNIX_DASH_OPTIONS

By default, the FTP server ignores Unix-style dash options on LIST and NLST when in Unix mode (for example, the "-l" in "ls -l"). Define this to be FALSE to tell the FTP server to pay attention to Unix-style dash options.

```
$ DEFINE /SYSTEM /EXEC TCPWARE_FTP_IGNORE_UNIX_DASH_OPTIONS FALSE
```

TCPWARE_FTP_LOGFILE

The TCPWARE_FTP_LOGFILE (system level, executive mode) logical can be defined to specify the name of a log file. This is good if you suspect break-ins to the FTP server. For example:

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

Note: You must restart FTP after setting this logical by issuing this command:

```
$ @tcpware:restart ftp
```

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 logical specifies the name of the file to which ALL commands and responses to ANONYMOUS FTP services are logged. If `TCPWARE_FTP_LOG_ALL_USERS` is also defined, then commands and responses for all users are logged.

TCPWARE_FTP_LOG_ALL_USERS

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

```
$ DEFINE/SYSTEM/EXEC TCPWARE_FTP_LOG_ALL_USERS TRUE
```

TCPWARE_FTP_LOWERCASE_NLST

When doing an `NLST` function (which is used as part of `MGET`) the FTP server now returns the filenames without making them lower case to preserve the case on ODS-5 disks. To restore the old behavior of making the filenames all lowercase define this logical to `TRUE`.

TCPWARE_FTP_MAX_PRE_ALLOCATION

The logical `TCPWARE_FTP_MAX_PRE_ALLOCATION` may be defined to limit the size that a file will be pre-allocated to when the size information is available at transfer time. This can be important when transferring very large files, as it can take a long time to pre-allocate the file at the start of the transfer and timeout routines in FTP and/or firewalls may cause connections to be dropped. This logical does not have any effect for `STRU` OVMS transfers of indexed, Contiguous, on Contiguous, Best Try files; these files need to have accurate allocation size information at the start of the transfer.

TCPWARE_FTP_MAXREC

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 TCPWARE_FTP_MAXREC logical to override the default of 8192. The definition of the TCPWARE_FTP_MAXREC logical is commented out but defined in the FTP_CONTROL.COM file as follows:

```
$ DEFINE/SYSTEM/NOLOG/EXEC TCPWARE_FTP_MAXREC 8192
```

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

TCPWARE_FTP_MAX_SERVERS

The logical name TCPWARE_FTP_MAX_SERVERS allows the maximum number of servers to be set. The default is 10000.

```
$ DEFINE/SYSTEM/EXEC TCPWARE_FTP_MAX_SERVERS "1500"
```

TCPWARE_FTP_MESSAGE_FILE

The TCPWARE_FTP_MESSAGE_FILE logical defines the message file the FTP user sees when connecting to the server or moving between directories. The definition of the TCPWARE_FTP_MESSAGE_FILE logical is commented out but defined in the FTP_CONTROL.COM file as follows:

```
$ !DEFINE/SYSTEM/NOLOG/EXEC TCPWARE_FTP_MESSAGE_FILE ".MESSAGE"
```

See *Special Messages*.

TCPWARE_FTP_NEW_LOGFILE

If the logical TCPWARE_FTP_NEW_LOGFILE is defined, then FTP_LISTENER opens a new logfile to record anonymous actions rather than appending to the existing one.

TCPWARE_FTP_NOKEEPALIVES

If the TCPWARE_FTP_NOKEEPALIVES logical is defined then the FTP server will not send keepalives on the control channel. The KEEPALIVE command allows the FTP client program to toggle whether it desires keepalives to be sent on the control channel. The SET [NO]KEEPALIVE command allows the FTP client to explicitly set whether it desires keepalives on the control channel.

TCPWARE_FTP_NO_PASV_SECURITY

If the TCPWARE_FTP_NO_PASV_SECURITY logical is defined then the FTP server does not perform the security check of comparing the IP address of the accepted port with the command port while operating in passive mode.

TCPWARE_FTP_ONLY_BREAK_ON_CRLF

If the TCPWARE_FTP_ONLY_BREAK_ON_CRLF logical is set and an ASCII file is transferred, a new line is created in the file upon receipt of a carriage return/line feed sequence.

If this logical is not set and an ASCII file is transferred, a new line is created upon receipt of either a carriage return/line feed sequence or a line feed.

```
$ DEFINE TCPWARE_FTP_ONLY_BREAK_ON_CRLF anything
```

TCPWARE_FTP_PASSWORD_WARNING_MESSAGE

The logical TCPWARE_FTP_PASSWORD_WARNING_MESSAGE defines the message that the FTP server displays when the user's password is going to expire within the warning time. If the amount of time before the password expires is to be displayed, use a %s in the logical.

```
$ DEFINE/SYSTEM/EXEC TCPWARE_FTP_PASSWORD_WARNING_MESSAGE "%s"
$ DEFINE/SYSTEM/EXEC TCPWARE_FTP_PASSWORD_WARNING_MESSAGE "message"
```

TCPWARE_FTP_PASSWORD_WARNING_TIME

The logical TCPWARE_FTP_PASSWORD_WARNING_TIME uses the VMS delta time to specify the minimum remaining lifetime for the user's password. If the remaining lifetime is greater than the VMS delta time, then no message is displayed. It is necessary to define this value to enable checking for the remaining lifetime of a password.

```
$ DEFINE/SYSTEM/EXEC @TCPWARE_FTP_PASSWORD_WARNING_TIME -
```

```
_$ "dddd hh:mm:ss.hh"
```

TCPWARE_FTP_RECEIVE_THRESHOLD

The TCPWARE_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.

```
$ DEFINE/SYSTEM/EXECUTIVE TCPWARE_FTP_RECEIVE_THRESHOLD 6144
```

TCPWARE_FTP_RECODE_NONVMS_FILE_NAMES

If this logical is defined, and the FTP server is not operating in UNIX mode, it recodes filenames that are not legal OpenVMS file names in the same manner that it would normally recode filenames when operating in UNIX mode. This is useful for handling filenames with multiple dots (.), spaces, and other characters that VMS does not allow in filenames while retaining the OpenVMS directory syntax.

```
$ DEFINE TCPWARE_FTP_RECODE_NONVMS_FILE_NAMES anything
```

TCPWARE_FTP_ROOT

The TCPWARE_FTP_ROOT (system level, executive mode) logical defines the system-wide default directory access restrictions for client users. The logical may be defined as a single directory or a search list of directories.

For example, you can restrict all users logged in via FTP to the COMMON$USER directory and its subdirectories, as follows:

```
$ DEFINE/SYSTEM/EXEC TCPWARE_FTP_ROOT COMMON$USER:
```

The FTP server defaults to this logical if the TCPWARE_FTP_ANONYMOUS_ROOT or TCPWARE_FTP_*username*_ROOT logicals (described in the next section) are not set.

> **Note:** The user is not placed automatically in this directory upon successful login unless the logical TCPWARE_FTP_SET_DEFAULT_TO_ROOT is defined to True.

TCPWARE_FTP_*username*_ROOT

The TCPWARE_FTP_*username*_ROOT (system level, executive mode) logical defines access restrictions for an FTP client logging in as *username*. The logical may be defined as a single directory or a search list of directories.

For example, you can restrict user CLARK to the COMMON$USER:[CLARK] directory and its subdirectories, as follows:

```
$ DEFINE/SYSTEM/EXEC TCPWARE_FTP_CLARK_ROOT COMMON$USER:[CLARK]
```

Because the FTP server restricts access by default to the directory setting in the TCPWARE_FTP_ROOT logical (described earlier), if it exists, you may want to use the special wildcard (*) setting with the TCPWARE_FTP_*username*_ROOT logical to bypass the default for *username*. For example, to restrict the bulk of users to DISK$SYS_LOGIN, restrict users KATE and PAUL to ENG$DISK, but allow SYSTEM full access to locations covered by its account, define the following logicals:

```
$ DEFINE/SYSTEM/EXEC TCPWARE_FTP_ROOT DISK$SYS_LOGIN ! default
$ DEFINE/SYSTEM/EXEC TCPWARE_FTP_KATE_ROOT ENG$DISK  ! limits KATE
$ DEFINE/SYSTEM/EXEC TCPWARE_FTP_PAUL_ROOT ENG$DISK  ! limits PAUL
$ DEFINE/SYSTEM/EXEC TCPWARE_FTP_SYSTEM_ROOT *       ! full SYSTEM
```

ANONYMOUS user access restrictions are described under *TCPWARE_FTP_ANONYMOUS_ROOT*.

> **Note:** The user is not placed automatically in this directory upon successful login unless the logical TCPWARE_FTP_SET_DEFAULT_TO_ROOT is defined to TRUE.

TCPWARE_FTP_SEMANTICS_FIXED_IGNORE_CC

If the TCPWARE_FTP_SEMANTICS_FIXED_IGNORE_CC logical is defined to TRUE, then GET operations of fixed lengths record files will not have a carriage return-line feed (CRLF) added to the end of each record.

```
$ DEFINE TCPWARE_FTP_SEMANTICS_FIXED_IGNORE_CC TRUE
```

TCPWARE_FTP_SEMANTICS_VARIABLE_IGNORE_CC

When this logical is defined to TRUE, files with variable length records and carriage return carriage control will NOT have a new line character inserted after each line when the file is transferred in image (binary) mode. The default is TRUE and is defined in FTPSERVER_DTP.COM.

```
$ DEFINE TCPWARE_FTP_SEMANTICS_VARIABLE_IGNORE_CC FALSE
```

Users can change this value by defining it in their LOGIN.COM file, or it can be defined on a system-wide basis if this is desired for all users.

TCPWARE_FTP_SERVER_DATA_PORT_RANGE

This specifies the upper and lower port boundaries that are to be used in passive data connections. The string should contain two numbers separated by a space.

```
$ DEFINE TCPWARE_FTP_SERVER_DATA_PORT_RANGE 5000 7000
```

TCPWARE_FTP_SERVER_LOG_LIMIT

By setting the logical name TCPWARE_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 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 TCPWARE_FTP_SERVER_LOG_LIMIT 42
```

> **Note:** If the TCPWARE_FTP_SERVER_LOG_LIMIT logical is not defined in a user's login.com, the system manager can make this a system-wide logical with the syntax:
>
> ```
> $ DEFINE/SYSTEM/EXEC TCPWARE_FTP_SERVER_LOG_LIMIT 42
> ```

TCPWARE_FTP_SERVER_RELAXED_PORT_COMMAND

The server normally compares the IP network address value specified in the PORT command with the IP network address of the IP address that it is receiving commands from. If these are not in agreement, the PORT command is not accepted. Some multi-homed clients, and clients that can do third party transfers send values that do not match. Defining this logical allows the PORT command to be accepted for these clients by disabling this check. The ? in the logical represents where defined values go. Defined value can be either alpha or numeric.

```
$ DEFINE TCPWARE_FTP_SERVER_RELAXED_PORT_COMMAND ?
```

TCPWARE_FTP_SET_DEFAULT_TO_ROOT

If this logical is defined to True, and TCPWARE_FTP_*username*_ROOT or TCPWARE_FTP_ROOT are defined, then the default directory is set to the specified root directory before the FTP server is started.

TCPWARE_FTP_STRIP_VERSION

The logical name TCPWARE_FTP_STRIP_VERSION causes VMS mode output to have no version numbers if TCPWARE_FTP_ALL_VERSIONS has been defined as FALSE.

```
$ DEFINE/SYSTEM/NOLOG/EXEC TCPWARE_FTP_STRIP_VERSION FALSE
```

TCPWARE_FTP_SYST_BANNER

If the FTP server is in UNIX mode, the SYST command displays the banner "UNIX TCPware Unix Emulation". If the FTP server is in VMS mode, the SYST command displays the equivalence string associated with the TCPWARE_FTP_SYST_BANNER logical name (if defined). Otherwise, the SYST command displays "VMS TCPware V*x.y*(*rev*)" where:

- V*x.y* is the TCPware version number.
- (*rev*) is the revision number of the FTP server.

```
$ DEFINE/SYSTEM/NOLOG/EXEC TCPWARE_FTP_SYST_BANNER "TCPware FTP"
```

Note: The logical name TCPWARE_FTP_SYST_BANNER is ignored if the FTP server is already in UNIX mode.

TCPWARE_FTP_STOU_OLDNAM

If the logical TCPWARE_FTP_STOU_OLDNAM is defined then the FTP server will attempt to preserve the existing file name when doing a store unique operation.

TCPWARE_FTP_TLS_ALLOW_CCC

If this logical is defined /system to False then users that come in via TLS are not allowed to change to a clear text command channel. A clear text command channel can be useful so that firewalls can open the necessary ports when the data channel is opened with the PORT, EPRT, PASV, EPSV commands and replies.

TCPWARE_FTP_TLS_ALLOW_CDC

If this logical is defined /system to False then users that come in via TLS must set the protection level to private before doing any data transfers. This includes directory commands, append, put and get.

TCPWARE_FTP_UNIX_STYLE_BY_DEFAULT

If you define the logical name TCPWARE_FTP_UNIX_STYLE_BY_DEFAULT, the FTP server starts in UNIX emulation mode.

Note: This logical should be used in conjunction with TCPWARE_FTP_DISALLOW_UNIX_STYLE to obtain the desired effect.

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

```
$ DEFINE/SYSTEM/NOLOG/EXEC TCPWARE_FTP_UNIX_STYLE_BY_DEFAULT TRUE
```

TCPWARE_FTP_UNIX_STYLE_CASE_INSENSITIVE

The logical name `TCPWARE_FTP_UNIX_STYLE_CASE_INSENSITIVE` allows UNIX style filename handling to be case insensitive.

```
$ DEFINE/SYSTEM/NOLOG/EXEC TCPWARE_FTP_UNIX_STYLE_CASE_INSENSITIVE
TRUE
```

TCPWARE_FTP_UNIX_YEAR_OLD_FILES

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 `TCPWARE_FTP_UNIX_YEAR_OLD_FILES` is defined `False`, the old behavior is restored, displaying all files with Month, Day, and Time.

TCPWARE_FTP_USE_SRI_ENCODING_ON_ODS5

The logical `TCPWARE_FTP_USE_SRI_ENCODING_ON_ODS5` can be defined to `TRUE` to cause the file name encoding used for UNIX-style file names on ODS-2 disks to be used on ODS-5 disks. This also sets the default case of letters in filenames to lowercase and ignores the stored case.

TCPWARE_FTP_WINDOW

The FTP client and the FTP server set the TCP window size of the data connection to either:

- The value of the `TCPWARE_FTP_WINDOW` logical if you define it (the minimum value is 512 bytes, the maximum value is 1,048,576 bytes).
- The larger of 32,768 bytes and the default TCP window size.

```
$ DEFINE/SYSTEM/NOLOG/EXEC TCPWARE_FTP_WINDOW 8192
```

Server Quota Control Logicals

The following logicals can be defined before starting FTP (or TCPware) to control the quotas that are used by the FTP listener and server.

- TCPWARE_FTP_SERVER_WSLIMIT
- TCPWARE_FTP_SERVER_WSQUOTA
- TCPWARE_FTP_SERVER_WSEXTENT
- TCPWARE_FTP_SERVER_PAGEFILE
- TCPWARE_FTP_SERVER_TQELM
- TCPWARE_FTP_SERVER_ENQLM
- TCPWARE_FTP_SERVER_ASTLM
- TCPWARE_FTP_SERVER_PRCLM
- TCPWARE_FTP_SERVER_BYTLM
- TCPWARE_FTP_SERVER_FILLM
- TCPWARE_FTP_SERVER_DIOLM
- TCPWARE_FTP_SERVER_BIOLM

Implementation

This section describes the FTP server implementation of the File Transfer Protocol (FTP) as defined in the RFC 959, RFC 2228, RFC 2428 and RFC 4217. The material in this section requires a thorough understanding of the protocols used.

The FTP server is “UNIX friendly” and accommodates pathname specifications in some Web browsers; the forward slash (/) at the beginning of directory structures is recognized.

The FTP server implements the following FTP service commands defined in the FTP protocol:

ACCT *arguments*	TCPware ignores this command but acknowledges it with successful completion.
ALLO *arguments*	If specified before a STOR operation, rounds up the size (in bytes) specified with ALLO to the number of blocks and uses it as the initial allocation. If the size specified with ALLO is negative, the allocation is contiguous.
APPE *filespec*	Appends the data received from the requesting host to the specified file (if the file does not exist, TCPware creates it).

AUTH TLS	Requests TLS authentication be used for user authentication as specified in RFC 4217.
CCC	Requests that the command channel return to clear text as specified in RFC 2228 and RFC 4217.
CDUP	Sets the default working directory to the parent directory for the current directory. XCUP is a synonym.
CWD *directory*	Specifies the new default working directory. XCWD is a synonym.
DELE *filespec*	Deletes the file (or files) specified.
EPRT *argument*	Specifies the data port number used for data transfers as per RFC 2428. Only IPv4 addresses are allowed. The FTP server reports a 501 Bad parameter value error if a port less than 1024 is specified. If you want to use a privileged port for the destination of data transfer, define the following logical to disable this feature: $ **DEFINE/SYSTEM TCPWARE_FTPD_ALLOW_PRIV_PORT "TRUE"** The code also checks that the IP address is the same as the client FTP address. When TLS is not in use this can be overridden by defining the logical TCPWARE_FTP_SERVER_RELAXED_PORT_COMMAND.
EPSV *argument*	Requests the server data transfer process to be passive as per RFC 2428. This means to "listen" on a non-default data port and wait for a connection instead of initiating one upon receiving a transfer command. The server responds with the host name and port number on which it is listening.
FEAT	Lists the available features (optional commands) that the FTP server includes support for.
HELP [*topic*]	Requests help information.

LIST *filespec*	Returns a directory listing.
MDTM *filespec*	Returns the modification time of the file in UTC as *YYYYMMDDHHMMSS*.
MKD *filespec*	Creates the specified directory. XMKD is a synonym.
MODE *arguments*	Specifies the transfer mode. The valid arguments are S (STREAM), C (COMPRESSED) and Z (DEFLATE).
NLST *filespec*	Returns a list of file names without a .DIR extension if the client is not an OpenVMS machine. Retrieving a directory file (*.DIR) if the client is not an OpenVMS machine results in an error message.
NOOP	TCPware ignores this command but acknowledges it with successful completion.
OPTS *parameters*	Set optional parameters for features that support them. The only feature that currently supports options is MODE Z which can have the LEVEL parameter for the ZLIB engine set to -1 to 9. -1 is the default which gives a balance of compression and CPU time usage, 0 (zero) is no compression, and 9 is maximum compression. Other options (ENGINE, METHOD, EXTRA and BLOCKSIZE) documented in draft-preson-ftpext-deflate-04 are not implemented as they do not apply to the ZLIB deflate engine.
PASS *password*	Logs the user into the host. If the first character of the password is a hyphen (−), the default successful login (230) message appears. The optional messages defined by the TCPWARE_FTP_230_REPLY or TCPWARE_FTP_ANONYMOUS_230_REPLY logicals do not appear. This supports clients that cannot receive the multi-line replies these logicals can generate.
PASV	Requests the server data transfer process to be passive. This means to "listen" on a non-default data port and wait for a connection instead of initiating one upon receiving a transfer command. The server responds with the host name and port number on which it is listening.

PBSZ *size*	Specifies the protection buffer size as described in RFC 2228 and RFC 4217. The only size allowed by RFC 4217 is 0 (zero).
PORT *arguments*	Specifies the data port number used for data transfers. The FTP server reports a 501 Bad parameter value error if a port less than 1024 is specified. If you want to use a privileged port for the destination of data transfer, define the following logical to disable this feature: $ **DEFINE/SYSTEM TCPWARE_FTPD_ALLOW_PRIV_PORT "TRUE"** The code also checks that the IP address is the same as the client FTP address. When TLS is not in use this can be overridden by defining the logical TCPWARE_FTP_SERVER_RELAXED_PORT_COMMAND.
PROT *level*	Specifies the protection level as described in RFC 2228 and RFC 4217. The only protection levels allowed by RFC 4217 are C and P. The server does not allow the protection level to be changed after the command channel has been set to clear text mode.
PWD	Returns the current working directory. XPWD is a synonym.
QUIT	Closes the connection.
REIN	Logs out the user and resets the file transfer parameters to the initial values.
RETR *filespec*	Reads the file and transmits it to the requesting host.
RMD *filespec*	Deletes the specified directory if the directory is empty. XRMD is a synonym. An error reply is sent if the directory is not empty.
RNFR *filespec*	Specifies the file to be renamed.
RNTO *filespec*	Specifies the new name of the file designated in the RNFR command.

SITE *arguments*	Used for site-specific requirements or capabilities. The following SITE commands are supported: SITE HELP - Returns a list of supported SITE commands. SITE PRIV [*privileges*] - Turns process privileges on or off. The arguments are ALL, NONE, or a privilege name. With no arguments, SITE PRIV displays the current process privileges. SITE RMS RECSIZE [*value*] - Controls the record size used when writing binary files; any valid RMS record size value is permitted. With no arguments, displays the current value. Applies only when STREAM is OFF. The default is 512. SITE SHOW TIME - Returns the current date and time-of-day for the OpenVMS system in the reply message. SITE SPAWN *command-line* - A subprocess executes the specified command line. Use this command for submitting batch jobs and printing files. The status returned for the SITE command depends on the status returned by the utility or command executed (see the VMS documentation regarding the DCL $STATUS symbol). SITE +VMS+ - Receiving this command from an HP TCP/IP Services for OpenVMS (UCX) client sets the file transfer mode to VMS_PLUS. WINDOW_SIZE - Displays or sets the TCP window size.
SIZE *filespec*	Returns the size in bytes of the file specified.
STOR *filespec*	Writes the file from data received from the requesting host. The STORE command supports the /ASCII, /BINARY, /BLOCK, /CONTIGUOUS=*blocks*, /FORTRAN, /IMAGE[=*n*], /RECORD, /VARIABLE, and /VMS qualifiers.
STOU *filespec*	Writes the data received from the requesting host to a unique filename. If you specify a *filespec*, TCPware uses it as the seed for the unique filename; otherwise, the server creates a unique filename. The STOU command uses a data connection.

Note: The STOU *filespec* pathname can contain the /ASCII, /BINARY, /BLOCK, /FORTRAN, or /IMAGE qualifier to specify the transfer mode. A qualifier can cause unpredictable results depending on the current TYPE and STRU settings. The pathname can also contain /CONTIGUOUS=*blocks*, in which case TCPware creates the file with an initial contiguous allocation of the specified number of blocks.

STRU *arguments*	Specifies the file structure. The valid arguments are F (for file, or no record structure), R (for record structure), and O VMS (or VMS, for VMS file structure). For VMS file structure, the data sent over the connection consists of a small header containing RMS file information, followed by raw data from the file, block by block.
SYST	Returns the name of the operating system running on the server.
TYPE *arguments*	Specifies the file type. The valid arguments are A (ASCII), I (image), L 8 (image), A N (ASCII non-print), A C (FORTRAN carriage control), and A T (Telnet format effectors). Use I (image) for both formatted binary and image format transfers. Formatted binary data includes the necessary record headers and checksums.
USER *name*	Logs the user into the host.

All other commands result in error 500 or 502 (command not implemented). This implementation accepts and may issue all response codes.

If you want the device name, the file name, and the directory name included in the results of all NLST commands, define the logical TCPWARE_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 TCPWARE_FTP_OBSERVE_VMS_PROTECTION to true.

RETRIEVE, STORE, and APPEND Command Qualifiers

The FTP server accepts the following qualifiers with client commands that send RETR, STOR, and APPE commands:

> **Note:** When using the /ASCII, /BINARY, /BLOCK, /FORTRAN, or /IMAGE qualifier with commands that send RETR or STOR commands, make sure to separate the file specification and the qualifier with a space character. Otherwise, the qualifier can be considered part of a UNIX file specification. For example, use the following on the client:
>
> ```
> ftp>put sample.txt "sample.txt /block"
> ```

/ASCII	TCPware reads or writes the file as an ASCII file.
/BINARY	TCPware reads or writes the file as a formatted binary file. Use this qualifier when transferring variable length binary files that do not have a file extension of .OBJ, .STB, .BIN, or .LDA.
/BLOCK	TCPware reads or writes the file using block-I/O mode. Use this qualifier when transferring STREAM_LF, STREAM_CR, STREAM or UNDEFINED files.
/CONTIGUOUS	(Applies to STOR only) the local output file should have an initial contiguous allocation of the specified number of blocks. If the output file is smaller, the FTP

	server truncates it. If the output file is larger, the additional allocations are noncontiguous.
/FORTRAN	TCPware reads or writes the file as a FORTRAN carriage control file.
/IMAGE*[=n]*	TCPware reads or writes the file as an image file. If you specify a record length, it only applies to output files.
/VARIABLE	TCPware writes an image format file as a variable length record format file. Ignored for all other transfer formats.

The FTP server also supports the STRU O VMS (or STRU VMS) format that allows OpenVMS systems to exchange any RMS file, including RMS indexed files.

> **Note:** Some combinations of these qualifiers and the TYPE and STRU commands may produce unpredictable results. Use these qualifiers carefully.

Troubleshooting

Q: How can I apply Access Control Lists (ACLs) to my FTP executables so that only I have access?

A: Assume you want to set up your username as FTP_USER and give yourself (and no one else) read and execute privileges to the FTP executables:

```
$ SET DEFAULT$SYSTEM
$ MCR AUTHORIZE
UAF>ADD/ID FTP_USER
UAF>GRANT/ID FTP_USER yourname
```

Then, for the FTP client:

```
$ SET DEFAULT TCPWARE
$ EDIT/ACL FTP.EXE
(IDENTIFIER=FTP_USER,ACCESS=READ+EXECUTE)
(IDENTIFIER=*,ACCESS=NONE)
Ctrl+Z
```

For the FTP server:

```
$ SET DEFAULT TCPWARE
$ EDIT/ACL FTP_DTP.EXE
(IDENTIFIER=FTP_USER,ACCESS=READ+EXECUTE)
(IDENTIFIER=*,ACCESS=NONE)
Ctrl+Z
```

12. Managing NFS Client

Introduction

This chapter describes how to manage the NFS client. Topics include client management concepts and how to mount remote file systems. The information applies to both NFSv2 and NFSv3 clients unless otherwise specified.

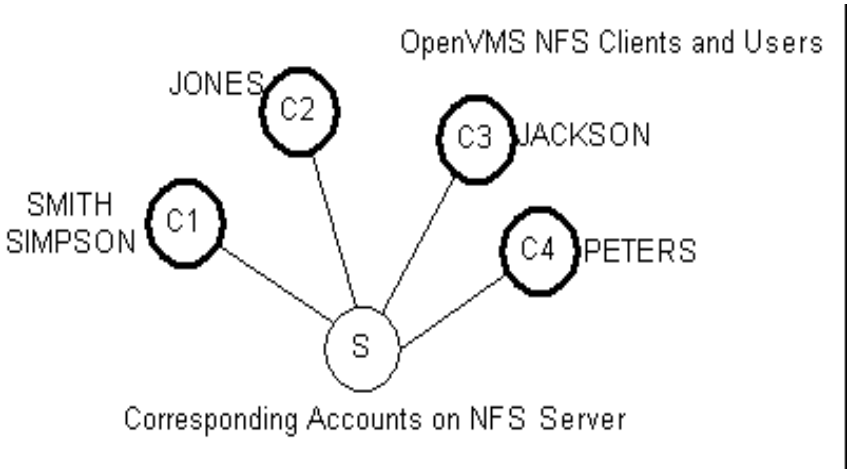
Client Concepts

The management concepts discussed in this chapter include:

- The NFS client-server concept
- User and file protection
- Filename and file version mapping

Client-Server

The client provides access to NFS-served file systems while resolving differences in user space and file access between the two systems. Consider the scenario below where separate clients use a single NFS server.



The indicated users for OpenVMS clients C1, C2, C3 and C4 need access to corresponding accounts on server S. If client users want group privileges to server files, the client system manager must create a group identity for the client users that maps to a group identity on S.

For example, SMITH and SIMPSON on C1 must have access to the SMITH and SIMPSON accounts on S. If they also want group access to files on S, the client system manager must give them group identity (say, ACCOUNTING) on C1, even though they may not be part of the same user group on C1.

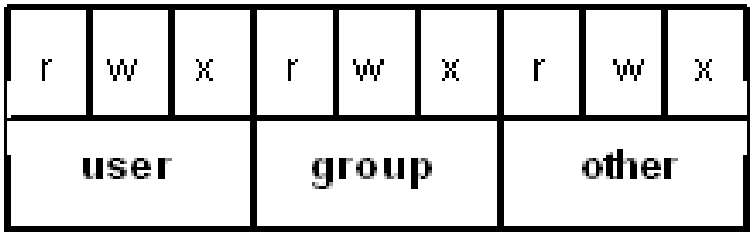
User and File Protection

User and file protection are different in NFS and OpenVMS. Because of this, TCPware must map user and file protection between the systems.

For the client to perform a server operation, two things apply:

1. The server must authorize the operation based on what its account can do following NFS/UNIX rules.
2. The client does its own user and file protection checking following OpenVMS rules.

The server has ultimate authority as to whether it should let the client perform the operation and may deny access based on NFS rules. The below figure shows this.



So that your client users can have access to server files:

1. Make sure the server system manager has the name of your client host in its export database. On many UNIX systems, this database is in the `/etc/exports` file; on hosts running TCPware, it is the `EXPORT` database.
2. Register each of your local users as having individual access to the appropriate server accounts. Do this by maintaining the `PROXY` database.
3. Register each of your local users as having the same group access to files as user groups on the server. Do this by maintaining the `GROUP` database.

The client protects files and checks file access on the server using the following criteria:

- User and group identification of whoever requests access to a file.

- Owner of the file
- Type of file access the user can have.
- Special user privileges

The following sections cover these criteria more fully.

User and Group Identification

One way to protect a file is to check the identity of the user requesting access. If the server identifies that the user has access to the file, the server grants access to it.

NFS User Identification

NFS uses UNIX semantics. These consist of a User ID (UID), Group ID (GID), and GID list. A user has a unique UID, belongs to a primary group, and can be a member of a limited number of other groups.

All NFS hosts must share the same user space so that a user has the same identity on all systems. Because an account with a single UID on the server can belong to many groups (can have multiple GIDs), you must associate a list of groups with that account.

Most UNIX servers have `/etc/passwd` and `/etc/group` files that maintain UID, GID, and group list information. The `/etc/passwd` file includes the account's login name, password, UID, and GID. The `/etc/group` file includes group names and their associated GID numbers and list of users. Each user can have a group list of up to 16 GIDs.

Parts of sample `/etc/passwd` and `/etc/group` files appear in the below examples.

```
nobody:Nologin:-2:-2:anonymous NFS user:/:/bin/date
ris:Nologin:11:11:Remote Installation Services
Account:/usr/adm/ris:/bin/sh
daemon:*:1:1:Mr Background:/:
sys:PASSWORD HERE:2:3:Mr Kernel:/usr/sys:
bin:PASSWORD HERE:3:4:Mr Binary:/bin:
root:PASSWORD HERE:0:1:supervisor:/
edwards:PASSWORD HERE:100:/usr/users/edwards
```

```
login:*:15:joe2
other:*:20:
accounting:*:10:edwards,root
testing:*:11:edwards,root
```

OpenVMS User Identification

In OpenVMS, a user has a unique user ID code (UIC) in the format `[group,member]`, where *group* and *member* are alphanumeric, or in the format *USERNAME*, which is the *member* part of the UIC. For example, a UIC can be `[306,210]`, `[GROUP1,JONES]`, or just `JONES`.

You can also identify groups of OpenVMS users through general or system-defined rights identifiers. An example is the `ACCOUNTING` identifier that gives all users in the accounting department the same access rights to files. The OpenVMS system manager defines the general identifiers in the system rights database using the `AUTHORIZE` utility.

The below table reviews the differences between NFS and OpenVMS system user identification.

User Identification Mapping
User identification mapping between client and server is straightforward. Because an NFS account has the same UID across multiple groups, the client maps UIDs directly to OpenVMS UICs. If the client finds an appropriate mapping entry in the `PROXY` database, the local user has access to the server account.

NFS user identification...	**Compared to OpenVMS user identification...**
User ID (UID), Group ID (GID): identified as: *uid* *gid* as in: 100 15	User Identification Code (UIC): GROUP number MEMBER number identified as: [*group*,*member*] as in: [306,210]
GID List: as in: 16,17,18	Rights Identifier: as in: ACCOUNTING

Group Identification Mapping
Group identification mapping occurs through a special GROUP database because of the difference between the NFS and OpenVMS group concept. This database ensures that the group privileges in OpenVMS more accurately reflect the file group privileges on the server side.

Although OpenVMS users may be in the same OpenVMS group, they must take into account that their corresponding NFS server accounts may be in different groups. NFS accounts in the same group should allow group access to their corresponding users in OpenVMS, even though the latter may not belong to the same UIC-based group.

You must populate the GROUP database, as well as the rights identifiers list in OpenVMS. Entries in the GROUP database map NFS group numbers to assigned OpenVMS groups. The mappings are either to wildcarded OpenVMS group entries, such as `[1000,*]` (which means "group 1000, any member"), or to rights identifiers, such as `ACCOUNTING`.

The below table reviews how the client handles user identification mapping.

OpenVMS user identification...	**Maps using...**	**For NFS authorization...**
UIC	PROXY database	UID, GID
UIC, Rights Database	GROUP database	GID List

File Ownership and Protection

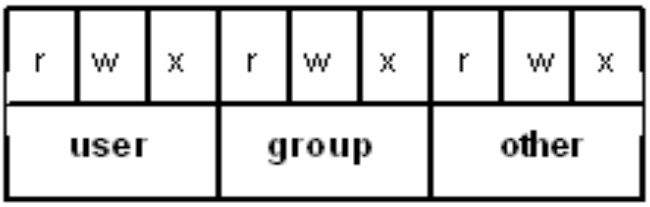
NFS File Ownership and Protection

Each NFS file has an owner and access restrictions (file protection) for various classes of users. File ownership and protection are file attributes.

Each NFS file has a UID and GID. When you create a new file, the NFS system:

- Sets the file's owner UID to the effective UID of the creating process.
- Bases the file's owner GID on the `set-gid` bit in the file's parent directory:
    - If on, the owner GID becomes that of the parent directory.
    - If off, the owner GID becomes the effective GID of the creating process.

NFS systems use a protection mask scheme for file protection. The NFS file protection categories are part of the file's mode attribute and are `user`, `group`, and `other`, each having read (`r`), write (`w`) or execute (`x`) access. NFS systems arrange the protection masks as shown below:



You can see the protection mask when you issue an `ls -l` command on the UNIX system server, as in the following example:

```
>ls -l
total 13949
-rwxr-x--x 1 smith 13330Jan 15 17:31 book
-rwxr-x--- 1 smith   44 Jan 15 17:31 games.com
drwxr-x--- 2 smith  512 Jan 15 17:38 Work
drwxr-x--- 1 smith   63 Jan 15 17:31 MARKET.rpts
```

In the example, the `book` file grants read (`r`), write (`w`), and execute (`x`) access to the file's owner; `r` and `x` access to the group; and `x` access to all other users.

The lines beginning with `d` indicate directories. None of the files other than `book` provide access for all users, nor do any of the directories.

OpenVMS File Ownership and Protection

You own a file in OpenVMS if your UIC and the file's owner UIC are the same. When you create a new file, its owner is:

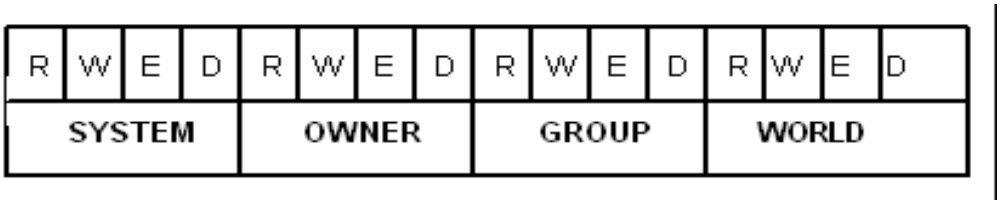
- The owner of the previous version of the file if you have rights to that owner
- The parent directory's owner if you have rights to that owner
- Your UIC

Each OpenVMS file has a protection mask that defines:

- The categories assigned to the file
- The types of access granted to each category

The four categories of OpenVMS file access are SYSTEM, OWNER, GROUP, and WORLD. Each category can have up to four types of access: read (R), write (W), execute (E), and delete (D).

OpenVMS arranges the protection masks as shown below:



In the following example, the SYSTEM and OWNER categories both have read (R), write (W), execute (E), and delete (D) access to the file. However, the GROUP category only has R and E access, and the WORLD category has no access at all:

```
SYSTEM=RWED, OWNER=RWED, GROUP=RE, WORLD=<NO ACCESS>
```

File Ownership Mapping

The below table shows how the client maps file ownership between server and client.

NFS file attribute...	Maps using...	For OpenVMS file attribute...
UID, GID	PROXY database	Owner UIC
GID List	GROUP database	(special group handling)

File Protection Mapping

File protection mapping from server to client is slightly different than mapping from client back to server. Both map the access privileges for non-group file access categories to the corresponding privileges on the other system. However, you must establish group access through the GROUP database. The client handles file protection mapping from server to client as shown in the below table.

> **Note:** The client honors the file protection scheme in the special, invisible ACL it creates for the file, and not in any other regular ACL.

NFS category...	In OpenVMS is...	With NFS type...	In OpenVMS is...
user	OWNER/SYSTEM	r	R
		w	W
		x	E
			D (unless ADF denies) [1]
group	GROUP	r	R (if GROUP database allows) [2]
		w	W (if GROUP database allows) [2]
		x	E (if GROUP database allows) [2]
			D (unless ADF denies) [1]
other	WORLD	r	R
		w	W
		x	E
			D (unless ADF denies) [1]

[1]The Client allows delete (D) access only if a special attributes data file (ADF) the client may create (and associates with the file) does not explicitly deny file deletion.

[2] If a GROUP entry that maps to a rights identifier (such as ACCOUNTING) exists, the client ignores the group protection mapping as given and uses the protection scheme in the special Access Control List (ACL) it creates instead. If a GROUP entry that maps to other than a rights identifier (such as a wildcarded group reference like `[1000,*]`) exists, the client honors the group protection mapping as given.

The client also handles file protection mapping from client back to server (such as when you create a file or change its attributes in OpenVMS), as in the table below:

OpenVMS category...	**In NFS is...**	**With OpenVMS type...**	**In NFS is...**
SYSTEM	(not mapped)		
OWNER	user	R	r
		W	w
		E	x
		D	(not mapped)
GROUP	group	R	r
		W	w
		E	x
		D	(not mapped)
WORLD	other	R	r
		W	w
		E	x
		D	(not mapped)

Special Users and Privileges

Systems have users (or privileges given to users) that OpenVMS treats specially when checking access.

OpenVMS provides SYSPRV privilege (which gives access to the SYSTEM category), BYPASS privilege (which bypasses all checking, giving all access), and READALL privilege (which provides a user at least READ and CONTROL access).

So that the NFS server can grant these privileges, the user must have superuser access on the server. The superuser usually acquires UID=0.

The client places undefined users by default in the `nobody` category, which provides a set of minimum access privileges. UID=-2 becomes user nobody and GID=-2 becomes group nobody.

Filename Mapping

For specific rules for mapping filenames between client and server, see Appendix A, *NFS-to-OpenVMS Filename Mapping*.

File Version Mapping

File version mapping can get rather complex due to the difference in behavior between OpenVMS and NFS systems. The general rule is that OpenVMS supports file versions; many NFS systems do not, and simply overwrite files on upgrading them. However, the TCPware client does preserve file versions on the server (unless you use the `/NOVERSION` qualifier the `NFSMOUNT` command to limit file versions to one).

The client still preserves an unversioned file on the server, which it hard-links to the highest (not necessarily most recent) version of the file every time it is upgraded.

In OpenVMS you could find the following `DIR` output:

```
Directory NFS4:[000000]
FILE-A.TXT;2 FILE-A.TXT;1 FILE-B.TXT;3 FILE-C.TXT;1
Total of 4 files.
```

The corresponding `ls` output on a UNIX NFS server would have the same files as follows:

```
total 6
-rwxr-x---2 root 5 Jun2  11:36   file-a.txt
-rwxr-x---1 root 2 Jun2  11:35   file-a.txt;1
-rwxr-x---2 root 5 Jun2  11:36   file-a.txt;2
-rwxr-x---2 root 2 Jun2  11:36   file-b.txt
-rwxr-x---2 root 2 Jun2  11:36   file-b.txt;3
-rwxr-x---1 root 2 Jun2  11:36   file-c.txt
```

The below table shows the file version rules when translating files from OpenVMS to NFS. The following table shows the file version rules when translating files from NFS to OpenVMS.

Rule	What Happens to Filenames from OpenVMS to NFS...
1	An initial version of a file gets no version number: `FOOBAR.TXT;1` becomes `foobar.txt` EXCEPTION: A file explicitly created as version 1 when a higher version already exists, which creates an explicit `foobar.txt;1`.
2	An upgraded file is linked with the unversioned file, and if the previous version was unversioned, it gets a version number: `FOOBAR.TXT;2` becomes `foobar.txt` (with a hard link to `foobar.txt;2`) `FOOBAR.TXT;1` becomes `foobar.txt;1` This rule also applies if using `NFSMOUNT /NOVERSION` and upgrading a file that already has a version in NFS, or creating one with an explicit version.
3	If using `NFSMOUNT /NOVERSION` and upgrading a file that shows only as unversioned in NFS, the file is overwritten and remains unversioned: `FOOBAR.TXT;1` becomes `foobar.txt` (with `foobar.txt;1` purged) EXCEPTION: An attributes data file (ADF) specifies a version limit other than one, or an explicit version upgrade is specified.

Rule	What Happens to Filenames from OpenVMS to NFS...
1	An unversioned file gets a version number preceded by a semicolon:

	`foobar.txt` becomes `FOOBAR.TXT;1`
2	If a filename does not include a file extension dot (`.`), it acquires one before the version number semicolon: `foobar` becomes `FOOBAR.;1`
3	After being translated, the file will not show up in the OpenVMS listing if its version number is greater than 32767.

Filesystem Mounting

The client links authorized (exportable) remote NFS filesystems to your OpenVMS system by mounting them (making them available) on a file structure you specify.

OpenVMS arranges file storage the user can access in directory trees. OpenVMS roots each tree (at the top) at an NFS device (such as `NFS1:`). The format of an NFS device is `NFS`*`n`*`:` where *`n`* is a number from 1 to 9999.

If you specify `NFS0:`, the client uses the template device and increments *`n`* by one for each new mount on the client host. For example, if someone mounts a filesystem on your host's `NFS5:` device and you specify a mount on `NFS0:`, the next mount is on `NFS1:` (or the next available device). The client uses the template device only when you specify `NFS0:` or omit the mount point specification entirely.

The mount point is both the point on the remote directory tree to be mounted and the point on the local directory tree where the client "attaches" the remote filesystem. A mount point can be at any level. OpenVMS's Record Management Services (RMS) lets you specify eight directory levels in addition to the master file directory (MFD or `[000000]`).

The most common client mount point is the MFD of the NFS device (`NFS`*`n`*`:[000000]`). Lower-level mounts create virtual directories. For example, if you specify a mount point `NFS1:[A.B]`, the Client creates virtual directories `A` and `B` on `NFS1`. These directories disappear once you dismount the filesystem.

NFS mounts are node-specific and not cluster-wide. Other nodes in the cluster cannot access these files. This has implications for printing files and running batch jobs and programs in a cluster. (See the next section.)

When a mount is initiated, it is first tried as NFS v3 and if that fails as NFS v2. This can be overridden with the `/NFS={2|3}` qualifier which will cause it to only try the specified NFS version. With NFSv3 the device will be mounted as an OpenVMS ODS-5 device on systems that support it if the NFSv3 server recognizes differences in case in file names. When the device is mounted as an ODS-5 device the filename mapping detailed in Appendix A is NOT used.

Cluster Environments

NFS is not VMS clustering. VMSclusters use RMS file locking that is more tightly coupled than the NFS advisory file level locking mechanism. In NFS, cluster-wide programs that store or exchange file IDs are unlikely to function properly. The NFS device is not available cluster-wide and the same filesystem mounted on different nodes has different file IDs.

The best NFS strategy is to allow only one NFS client system to write to the server files. If you need multiple clients to write to the same file, use the Network Lock Manager (NLM) by specifying `NFSMOUNT /LOCK`. Also ensure that all systems (client and server) use the NLM to coordinate file access of the participating systems. NLM provides advisory locking, not mandatory locking as with VMSclusters.

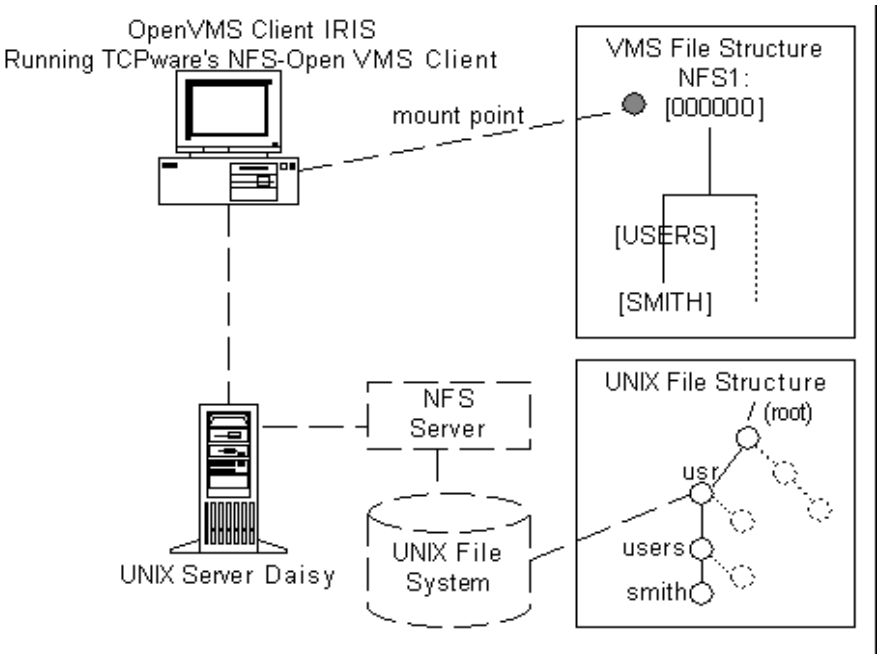
The fact that the client mounts filesystems only on the local OpenVMS cluster node has implications for printing files and running batch jobs and programs in a cluster environment. In printing a file, make sure that you set up the printer on the node on which you mount the NFS filesystem. Also make sure that no one remounted the NFS filesystem during the print operation. Otherwise, the print operation fails because the client changes the file ID with each mount (the printing system keeps track of jobs using file IDs). The same applies to batch jobs involving NFS files.

To print files in or submit batch jobs for mounted NFS filesystems across the cluster, first copy the files to a "standard" non-NFS disk which you can access cluster-wide.

For details on network file locking and its implications for both client and server users, see Chapter 14, *NFS Server Management*, *Network File Locking*. Also see *Network File Locking* in this chapter.

Mount Example

The below diagram shows an example of an exported UNIX filesystem mounted in OpenVMS.



In the figure, an OpenVMS user on host IRIS needs access to the `/usr/users/smith` subdirectory on UNIX system server Daisy. Other IRIS users may need access to subdirectories below `/usr`.

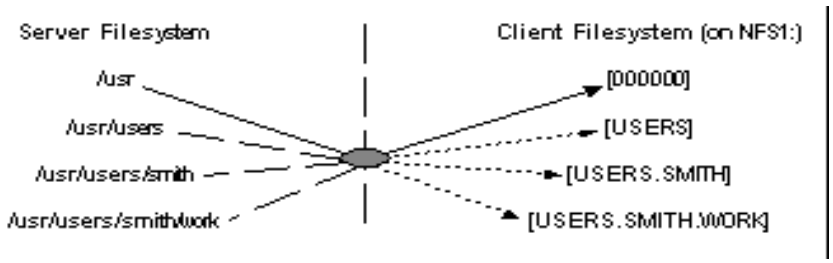
Using the `NFSMOUNT` command at the DCL prompt, IRIS's system manager mounts `/usr` on IRIS's `NFS1:` device, where the files are now "attached" to `NFS1:[000000]`. The client creates two virtual directory levels (`[USERS.SMITH]`) below `NFS1:[000000]`. If the user wants access to files in `/usr/users/smith`, the way to access them is through `NFS1:[USERS.SMITH]`.

Had the system manager mounted `/usr/users/smith` instead of just `/usr` on the same mount point, `/usr/users/smith` would be equivalent to `NFS1:[000000]`. However, the user would then be excluded from any directory level above `/smith` (`/usr` and `/users`).

Mount Flexibility

The flexibility of the client's mount capabilities appears in the below diagram. The NFS filesystem is on the left and the corresponding OpenVMS filesystem is on the right. Each row is a directory level. The solid line pointing from `/usr` to `[000000]` indicates the mount scenario indicated in *Mounting a UNIX Filesystem on the OpenVMS Client***Error! Reference source not found.**, where the user mounted the `/usr` filesystem on `NFS1:[000000]`.

The dotted lines indicate that you can essentially bend the arrow anywhere around the central pivot point so that you can mount any server filesystem level on any client filesystem level. You can mount a low server directory on a high client mount point, a high directory on a low mount point, a low directory on a low mount point, and so on.



You can even mount a file on a file, such as `/usr/users/smith/junk.txt` on `NFS1:[USERS.SMITH]JUNK.TXT`. However, mounting a file limits the user to accessing that file only. This makes new file creation impossible since the client cannot "go up" the directory tree to get the necessary attributes to create a new file.

You can only access server resources from the mount point on down. For example, if you use the following `NFSMOUNT` command, you can access all available resources in Daisy's `/usr/user/smith` directory mounted locally as `NFS1:[USERS.SMITH]`, but you cannot back up to `NFS1:[USERS]` and expect to access resources in Daisy's `/usr` directory that are not in the `/users` subdirectory:

```
$ NFSMOUNT DAISY "/usr/users/smith" NFS1:[USERS.SMITH]
```

You can use `NFSMOUNT` command options for certain mount strategies for your specific implementation, such as automounting and background mounting.

For details, see *Mount Strategies.*

Mount Commands

The client commands related to file mounting appear in the below table.

Command	Description
`NFSMOUNT`	Mounts a remote NFS directory on a local mount point; similar to UNIX systems' `/etc/mount` file
`NFSDISMOUNT`	Dismounts a remote NFS directory from a local mount point; similar to UNIX systems' `/etc/unmount` file
`NFSMOUNT /CONFIG`	Mounts a remote NFS directory on a local mount point based on information in a configuration file

`NFSMOUNT /SHOW`	Shows the mount information for the local mount point

Perform mount commands at the DCL level. The NFS mount command format is:

```
NFSMOUNT server "pathname"[mountpoint [logical]]
```

server	name of the remote NFS server
"pathname"	server's exported pathname (enclosed in quotation marks)
mountpoint	optional NFS device (and directory path, if necessary) where the mount occurs on the OpenVMS host. If you do not specify the mount point, the default is `NFS0:[000000]`, the MFD mount on the template device, as discussed earlier.
logical	is the optional logical name associated with the *mountpoint*.

The following command mounts Daisy's `/usr/users` filesystem on the `NFS1:[000000]` mount point:

```
$ NFSMOUNT DAISY "/USR/USERS" NFS0: DAISY$USR_USERS
```

The client immediately creates the `NFS1:` mount point based on the `NFS0:` template specification. The mount point also gets the (optional) logical name `DAISY$USR_USERS`.

The NFS dismount command format is:

```
$ NFSDISMOUNT mountpoint
```

mountpoint is the mount point or logical name, as defined above.

Mount Strategies discusses the various mount and dismount options. For details on the mount and dismount commands, see *Client Commands*.

Symbolic Links

The client supports symbolically linked (known as "soft-linked") files on UNIX servers. This support preserves existing symbolic links when you back up your client filesystems and restore them on these servers.

The client does not "chase" symbolic links. If you open a soft-linked file in OpenVMS, it displays the pathname of the linked file, unlike UNIX systems that display the actual content of the linked file.

Client Auditing

The client supports OpenVMS security auditing that keeps track of security events users can cause and issues security alarms for these events.

See HPE's *Guide to VMS System Security* for details.

Mount Strategies

The client offers several ways to mount a filesystem:

- Regular mount
- Shared mount
- Automount
- Background mount
- Overmount
- Occluded mount
- Mount using network file locking

Regular

The following shows a sample confirmation message given when mounting SIGMA's `/usr` directory on an `NFS0:` template device:

```
$ NFSMOUNT SIGMA "/usr" NFS0:
%NFSMOUNT-S-MOUNTED, /usr mounted on NFS101:[000000]
```

If you use the same command again, the client creates a new mount point (such as `_NFS102:[000000]`).

There are other options you can set using command qualifiers during a regular mount, such as setting SYSGEN parameters and cache timeout.

These options appear in *Other Mount Options*. For the mount qualifiers, see the NFSMOUNT command in *Client Commands*.

Shared

The client supports shared mounting through the use of the /SHARE qualifier to the NFSMOUNT command. The format of a shared mount request is as follows:

```
$ NFSMOUNT BART "/ENG" NFS1: BARTMOUNT
$ NFSMOUNT/SHARE BART "/ENG" NFS1:
```

The first mount request creates the NFS1: device, and puts the BARTMOUNT logical in the system logical name table. The mount count is set to one. The second (shared) mount request, which includes the same mount information, increments the mount count by one. Unless you specify the /SHARE qualifier and the same hostname, path, and device/mount point for the second mount request as for the first, the second mount request is considered a new mount request and not a shared mount request.

Logical names go into the SYSTEM logical name table by default. A /SHARE mount, however, puts the logical name in the JOB logical name table. In this way the logical name is only available to processes of the job mounting the device.

The mount count increments and decrements under the following circumstances, instead of being automatically set to zero:

- With an initial SYSTEM or GROUP mount
- With an NFSMOUNT /SHARE command that completes without an error (the exception being an overmount, where the previous mount point is dismounted, in which case it may even be decremented if the previous mount point was shared)

In this way, if the main process of the job logs out, the job mount is deallocated, and the volume mount count decrements by one (and if zero, dismounts the device).

The NFSDISMOUNT command dismounts one or more (in the case when using the /ALL qualifier) mount points. If the mount point being dismounted is the only or last one for the device, the device is dismounted for all users who mounted it and the mount count is decremented to zero. If more than one mount point exists, the mount point is dismounted along with any specifically shared mounts.

Automounting

Use the /AUTOMOUNT qualifier to automount a filesystem, which automatically and transparently mounts a remote server path as soon as the client obtains the pathname. You can specify an inactivity period (the default is five minutes, seconds being rounded to the nearest minute), after which the client automatically dismounts the path.

In this example, the client mounts SIGMA's /usr filesystem when it references the pathname, keeping the path mounted until an inactive period of 10 minutes, after which time it unmounts the filesystem (subsequent references causing the client to remount it):

```
$ NFSMOUNT SIGMA "/usr" NFS0: /AUTOMOUNT=(INACTIVITY=00:10:00)
```

Background

Use the /BACKGROUND qualifier to mount a filesystem in background mode, where the client tries to mount a filesystem at least once. If the first try fails, the client informs you of the failure and tries again every 30 seconds (by default). Unless you set it otherwise, the client stops retrying after 10 attempts.

In this example, the client tries to mount the /usr filesystem; if it cannot, it waits one minute and retries the connection up to 20 times:

```
$ NFSMOUNT SIGMA "/usr" NFS0: /BACKGROUND=(DELAY=00:01:00,RETRY=20)
```

Overmounting

Use the /FORCE qualifier to overmount a filesystem, where you mount another path on an already existing mount point. The client dismounts the original filesystem and replaces it with the new one. (If you omit /FORCE, the message overmounting requires the use of /FORCE appears.) Mounting a higher or lower directory level in a previously used path also causes an overmount, as in the following example:

```
$ 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
```

Occluded

Use the /FORCE qualifier for an occluded mount, where you mount a filesystem onto a client mount point that is higher or lower in the directory structure than an existing, active mount. (If you omit /FORCE, the message occlusion requires the use of /FORCE appears.) No dismounting occurs and both mounts are visible. However, the client occludes (hides from view) the subdirectory (or

subdirectories) added to or dropped from the original mount specification when you perform a directory listing.

In the following example, the mount point specification moves up one subdirectory. If you enter the `NFSMOUNT /SHOW` command, both mounts are visible. 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]
-TCPWARE-I-OCCLUDED, previous contents of _NFS2:[USERS.SMITH] occluded
```

Network File Locking

Use the `NFSMOUNT /LOCK` command to enable network file locking during an NFS mount. The NLM applies the lock to any file you create or to which you request exclusive access in the specified filesystem. The locks are on entire files only and not on file regions or byte ranges. Here is a typical example:

```
$ NFSMOUNT SIGMA "/usr" NFS0: /LOCK
```

Other Mount Options

This section identifies other mount options you can set.

Auto-converting Text Files

By default, the client automatically converts newly created text files of variable-length, implied carriage return control (VAR-CR) format to STREAM-LF format. This is appropriate for UNIX system servers, where the files show up as STREAM-LF. For a PC system server, however, use the `NFSMOUNT /CONVERT=STREAM_CRLF` option to do a carriage-return-line-feed conversion for the mount point. Converted files will show up on the server as stream files, as do files that do not have attributes data files (ADFs) associated with them (see the next section).

Some OpenVMS applications require that certain files remain in VAR-CR format on the client (such as with TCPware's NFS Server). You can retain the VAR-CR format by specifying the `/NOCONVERT` qualifier during a mount. For example:

```
$ NFSMOUNT SIGMA "/usr" NFS0: /NOCONVERT
```

Attributes Data Files

Attributes Data Files (ADFs) are special companion files the client maintains on a non-VMS server to preserve attribute information the server would not normally recognize.

The client maintains regular and default ADFs for files such that:

1. If a regular ADF exists, the client uses the attributes from that file.
2. If a default ADF exists, the client uses the attributes from that file.
3. If no ADF exists, the file must be STREAM-LF.

The client may create a regular ADF for a file in response to a write attributes operation that sets an OpenVMS attribute that NFS cannot normally maintain. For example, a `SET FILE /NOBACKUP` command would cause the client to create an ADF, since NFS has no concept of this OpenVMS attribute.

Default ADFs minimize the number of regular ADFs, since one default ADF can serve all files of a particular type. The client provides default ADFs for files with .EXE, .HLB, .MLB, .OBJ, .OLB, .STB, and .TLB extensions. The client does not provide ADFs for files with the .TXT and .C extensions, since most of these are STREAM-LF.

For example, `TCPWARE:EXE.ADF` is the default ADF for all .EXE type files. When you create .EXE files (or if they exist on the server), the record attributes from the single default ADF are probably enough to define each file. Each file does not need its own regular ADF.

> **Note:** The client uses only certain record attributes and file characteristics in default ADFs. It uses the 32-byte `ATR$C_RECATTR` attributes other than the `FAT$L_HIBLK, FAT$L_EFBLK`, `FAT$W_FFBYTE`, and `FAT$W_VERSION` fields, and uses four-byte `ATR$C_UCHAR` attributes other than the `FCH$M_DIRECTORY` and `FCH$M_CONFIG` bits. All other information stored in an ADF is ignored for default ADFs. For additional details on these file attributes, see HPE's *OpenVMS I/O User's Reference Manual*, the *ACP-QIO Interface* chapter.

When a user creates a file on the client, the client only creates a "regular" ADF for it if the default ADF attributes or default attributes do not match.

You can create customized ADFs for special applications. To do so:

1. On the client, create a special application file that creates an ADF on the server.

2. Suppose that application file is TEST.GAF. On the server, check the listing for the data file, along with its ADF (`.$ADF$test.gaf;1`):

```
>ls -a
.
..
.$ADF$test.gaf;1
test.gaf
```

3. On the server, copy the ADF file to a newly created default ADF file on the client:

```
>cp .\$ADF\$test.gaf\;1 gaf.adf
```

   Note the backslashes (`\`) required to recognize the UNIX system special `$` symbol and the `;` symbol required to specify a version number.

4. On the client, copy the new default ADF file to the `TCPWARE_COMMON:[TCPWARE]` directory:

```
$ COPY GAF.ADF TCPWARE_COMMON:[TCPWARE]
```

5. Dismount all the NFS volumes and remount them again. This starts another NFS ancillary control process (ACP) so that the newly copied default ADF file can take effect.

You can also specify how you want ADFs used. The client offers three options with the `/ADF` qualifier of the `NFSMOUNT` command:

`CREATE`	If ADFs exist on the server, the client uses them, updates them, and creates them for new files. This setting is the default and recommended setting.
`UPDATE`	If ADFs exist on the server, the client uses them and updates them, but does not create them for new files.
`USE`	If ADFs exist on the server, the client uses them, but does not update them, nor does it create them for new files.

> **Note:** Avoid using `UPDATE` and `USE`. The client may still create ADFs in certain cases, such as when renaming files. Also, changing OpenVMS attributes for a hard-linked file can result in inconsistent OpenVMS attributes between the linked files.

You can also specify /NOADF. In this case, the client treats all files as STREAM-LF unless a default ADF matches, and it can use it. Note that this mode of operation is only appropriate for read-only filesystems since the client cannot adequately handle application-created files when /NOADF is in effect.

Cache Timeout

Use the /CACHE_TIMEOUT qualifier to set the caching timeout period for the mount point. For example:

```
$ NFSMOUNT /CACHE_TIMEOUT=(DIRECTORY=::15,ATTRIBUTE=::10)
```

The DIRECTORY timer specifies the time (in delta time) the client waits between rereading a directory's status or contents. The default is ::30 (30 seconds). The ATTRIBUTE timer specifies the time the client waits between rereading a file's attributes from the server. The default is ::15 (15 seconds).

Read/Write Transfer Size

Use the /DATA qualifier to specify the largest amount of NFS data you want to read (receive) or write (transmit) in a single network operation. For example:

```
$ NFSMOUNT /DATA=(1024,1024)
```

The first value is the read value and the second is the write value. Most servers let you read or write 8192 bytes (the maximum and default setting). However, some may require less. The minimum you can specify is 512 bytes.

If you eliminate the parentheses and specify only one value, this serves for both the read and write value. However, if the NFS server requests a smaller transfer size than the one you set, the server's requested value will override yours.

Default User

Use the /USER qualifier to specify the default user to which you want to map unknown UIDs. For example:

```
$ NFSMOUNT /USER=SMITH
```

The client tries to use the user account. If not found, the DECNET account becomes the default. If the DECNET account is not found, the [200,200] account becomes the default.

Default UIDs and GIDs

Use the /UID and /GID qualifiers to specify the default UID and GID. The client uses the default UID and GID if there is no mapping for the requesting user in the PROXY database. Usually the default UID is -2 and default GID is -2. For example:

```
$ NFSMOUNT /UID=100/GID=15
```

Limiting File Versions

Use the /NOVERSION qualifier to enforce a limit of one version on a non-TCPware server file. This is a way of imposing an NFS file versioning scheme on OpenVMS files. (/VERSION, allowing multiple versions, is the default).

With /NOVERSION, unversioned files stay unversioned, and new files are unversioned along with any subsequent upgrades (which is consistent with most NFS servers). When higher versions already exist, the number of versions cannot grow beyond the current number, so that the lowest version is purged on each upgrade.

For example, with /NOVERSION in effect, if you start with FILE.TXT;1 (which shows up as file.txt on the server) and you edit FILE.TXT, you have an overwritten FILE.TXT;1 on the client, and file.txt on the server. If you already have FILE.TXT;1 and FILE.TXT;2 and you edit FILE.TXT, you end up with FILE.TXT;3 and FILE.TXT;2 on the client, with version 1 purged. The server shows file.txt, file.txt;2, and file.txt;3 (hard-linked to file.txt).

To prepare a directory for use with /NOVERSION, it may be best to purge and rename its files, as follows, being aware that purged files are lost forever and to back up your files whenever possible:

```
$ PURGE *.*                ; deletes old versions
$ RENAME *.*;* *.*;1       ; forces server to rename files to unversioned
$ SET FILE /VERSION=1 *.*; overrides existing ADFs
```

Superusers

Use the /SUPERUSER qualifier if you want to allow access to the superuser account UID on the server. For example:

```
$ NFSMOUNT /SUPERUSER=200
```

To enable superuser privilege, the server must allow access to the superuser and the OpenVMS user must have SYSPRV, BYPASS, or READALL privileges. Normally, the superuser UID is 0. The default is /NOSUPERUSER.

Mount Type

If you specify a logical name for the mount point, the client creates a system logical name by default. This is equivalent to using the `/SYSTEM` qualifier of the `NFSMOUNT` command. If you specify the `/GROUP` qualifier, the client considers the mount a group mount and places the logical name (if specified) in the group table. Both mounts are subject to a privilege check.

Server Type

Use the `/SERVER_TYPE` qualifier to set the server type to `UNIX`, `TCPWARE`, or `IBM_VM`. By default, the server type is `UNIX` or `TCPWARE`, depending on the server. For example:

```
$ NFSMOUNT /SERVER_TYPE=IBM_VM
```

The server types displayed in the below table are available with the client.

Option	Description
`IBM_VM`	IBM Virtual Machine (VM) machines
`TCPWARE`	OpenVMS systems running the TCPware NFS server
`UNIX`	All UNIX/Linux system machines

TCPWARE Server Type

When mounting a filesystem served by TCPware's NFS server, either omit `/SERVER_TYPE` or specify `/SERVER_TYPE=TCPWARE`. If omitted, the client determines the `TCPWARE` server type automatically. Note the following:

- The client and server map UICs to UIDs and GIDs. As long as system managers on each system maintain the PROXY databases properly, this saves having to maintain the same set of UICs on the client and server systems.
- The client and server use ACLs as is. This means that identifiers on the client and server systems must be the same to produce the desired results.

IBM_VM Server Type

IBM's VM NFS server partially supports requests to change file size. This means that:

- OpenVMS Files-11 ODS-2 is a block-oriented filesystem. Applications (and RMS) can only read and write 512-byte blocks. The client uses ODS-2 file attributes to maintain information about the true last data byte of a file.

- To accommodate the IBM VM NFS server's inability to truncate a file to its real size (the client normally truncates the file based on the end-of-file information), the client stores the size information in the ADF for the file.
- With any access to the file from a non-TCPware NFS client or an application on the server, you may see garbage beyond the true end of the data. (This garbage data exists because of the block nature of ODS-2 and the server's inability to truncate the file to its real size.)
- With a file stored on the IBM VM NFS server by a non-TCPware client or an application on the server itself, the ADF does not reflect any changes to the file. This can cause problems if a client later opens the file, expecting changes.

When mounting a filesystem on an IBM VM minidisk, you must specify certain IBM VM NFS server specific parameters in the pathname parameter of the mount command. For example, a mount to an IBM VM minidisk might be:

```
$ NFSMOUNT IBMVM "test2.191,rw,user=simpson,pass=bart,record=nl"
```

You may need to specify one or both of the following parameters:

```
record={binary | nl | text}
```

`binary` (default)	The IBM VM NFS server does not convert data to EBCDIC. This mode is most useful when storing data to which you do not have access from applications on the IBM system, or when transferring binary data.
`nl`	The IBM VM NFS server translates EBCDIC to ASCII (and vice versa). This mode is most useful when storing text files to which you have access from applications on the IBM system. Do not use it when you have access to or store binary data files.
`text`	The file conforms to the IBM VM CMS structure. Use of this parameter value is not generally recommended.

```
name={trans | fold | mixed}
```

`trans` (default)	Supports the widest range of filenames. The IBM VM NFS provides transparent mapping for filenames that contain invalid characters or are longer than CMS allows. However, the client does not use this mapping if the filename (ignoring case) is valid on the CMS filesystem. Therefore, for short filenames, the mapping may not be transparent.
`fold`	Only supports filenames valid to the CMS filesystem and ignores case.

`mixed`	Like `name=fold` except that it preserves case.

For complete details on these server types, see the IBM *TCP/IP for VM: User's Guide*.

Retry Times

Use the `/RETRIES` qualifier to specify the maximum number of times the client retransmits a Remote Procedure Call (RPC) request. For example:

```
$ NFSMOUNT /RETRIES=10
```

There is no maximum value you can specify. The default is zero (0) retries, where the client retries the requests indefinitely.

Timeout Times

Use the `/TIMEOUT` qualifier to set the minimum timeout period (specified in OpenVMS delta time) for initial RPC request retransmissions.

The timeout period value should reflect the estimated typical round-trip time for RPC requests. For slower speed links (such as over SLIP or WAN lines), use a larger value than the default of one second. For example, for a maximum read/write size of 8192 bytes (see the `/DATA` qualifier) over a 19,200-baud SLIP line, the absolute minimum timeout value is:

```
10240 bytes * 8 bits per byte
------------------------------------ = 4.27 seconds
19200 bits per second
```

The 10240 bytes are 8192 plus RPC overhead. Since 4.27 seconds is the absolute minimum, a more realistic value for this link is in the range of 15 to 30 seconds to allow for other traffic.

Volume Labels

Use the `/LABEL` qualifier to specify the volume label to use for the remote pathname during a mount. If you omit `/LABEL`, the client uses a default label consisting of the first 12 characters of the combined *`server:mountpoint`* parameter. The client applies the `/LABEL` qualifier on the first mount of an NFS device only and ignores it with subsequent mounts on that device. If you perform a `SHOW DEVICE NFS`*`n`*`:` DCL command, you see only the first 12 characters of the volume label specified.

Cache Space

One of the options during a mount is the `/PROCESSOR=UNIQUE` qualifier setting. As a general rule, the larger the remote filesystem, the more likely you are to use this option.

With `/PROCESS=UNIQUE`, a new NFSACP process is created for each mount. This creates multiple address space, in which case the collective ACPs can accommodate much more cached information. The size of the cached information depends mostly on the number of NFS files the client recognizes by obtaining a file handle and creating a mapping to a file ID. This happens with any file or directory access.

Each NFSACP process can support up to 250 mounted filesystems. If one process is handling all mounts, there is only one address space to cache the information. The size of this address space depends on a number of system parameters such as VIRTUALPAGECNT, and process parameters such as the working set limits and paging file limits.

Disk Quotas

You can display quota information for the current user's mount by using the `NFSMOUNT /SHOW` command with the `/QUOTA` qualifier. The output displays block usage, soft limit (quota), hard limit, and grace period. Using the additional `/FULL` qualifier displays four additional values that are relevant to UNIX servers: file usage, quota, limit, and grace period.

You can use the additional `/USER` qualifier to request quotas for other than the current user. However, the `NFSMOUNT` required the `/SUPERVISOR` qualifier and SYSTEM, BYPASS, and READALL privileges. (The DCL command `SHOW QUOTA` also works in this case.)

The following shows sample output:

```
$ 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             TCPware
  Directory time    0 00:00:30.00       Advisory Locking       Disabled
  Cache Validation    MODIFY TIME       Default user             [USER]
  Superuser                   No       Default UID,GID          100,15
```

Implementation

There are only minor differences between the way the client and Files-11 ODS-2 handle files. For example, the client:

- Does not determine the physical placement of files on the disk.
- Does not support the `INDEXF.SYS` file, which means that you cannot perform operations such as `ANALYZE/VERIFY` and `BACKUP/IMAGE` in OpenVMS.

> **Note:** The NFS client is not supported in the POSIX environment.

Client Commands

The below table shows the mount and dismount commands available at the DCL level in OpenVMS.

DCL command	Description
`NFSMOUNT`	Mounts a remote NFS directory on a local mount point; similar to UNIX systems' `/etc/mount` file
`NFSDISMOUNT`	Dismounts a remote NFS directory from a local mount point; similar to UNIX systems' `/etc/unmount` file
`NFSMOUNT /CONFIG`	Mounts a remote NFS directory on a local mount point based on information in a configuration file
`NFSMOUNT /SHOW`	Shows the mount information for the local mount point

The mount and dismount commands use OpenVMS delta time for all time-related values.

The delta time syntax is:

```
dddd hh:mm:ss.cc
```

Troubleshooting

The NFS client can produce messages for the `NFSMOUNT` and `NFSDISMOUNT` commands, and in OPCOM. Access error messages help by entering:

```
$ HELP TCPWARE MESSAGES [identifier]
```

NFSMOUNT

Mounts a remote NFS directory to a local mount point. The command is similar to the UNIX system `/etc/mount` command.

DCL 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 TCPware Network Control Utility (NETCU) 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:

```
NFSn:[[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:

```
NFSn:[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=TCPWARE)	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 VMS attributes for a hard-linked file may result in inconsistent VMS 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 *Delta Time Examples*). 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 times 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 *Delta Time Examples*). 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 | STREAM_CRLF}
/NOCONVERT
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=TCPWARE, in which case TCPware 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.

/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
Controls whether 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
Specifies if the client should use advisory network file locking by way of the Network Lock Manager (NLM) to coordinate access to server files.

/NOLOOKUPS
/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.

/NFS={2|3}
Specifies that only a particular version of NFS be used when attempting to mount the unit. If this qualifier is not specified, then NFSv3 is attempted first and then NFSv2 if that fails. The NFSv3 ACP can only be used to service NFSv3 mount points and the NFSv2 ACP can only be used to service NFSv2 mount points, so caution is advised when using the /PROCESSOR qualifier. NFSv3 mount points will be presented as an ODS-5 disk for OpenVMS systems that recognize ODS-5 when the server maintains the case of filenames and maintains the number of hard links. When the device is presented as an ODS-5 device there is no mapping of filenames; case sensitivity and parsing rules are controlled by the VMS process parameters.

/NOREADDIRPLUS
For NFSv3 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. (See *Cache Space*.)

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 NFSv2 and NFSv3 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, TCPWARE, or IBM_VM.

The default is either UNIX or TCPWARE (if the server runs TCPware's NFS server).

With /SERVER_TYPE=TCPWARE, TCPware 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. (See *Shared*.)

/SUPERUSER=*uid*

Controls if 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:

```
10240 bytes * 8 bits per byte
------------------------------------ = 4.27 seconds
19200 bits per second
```

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` or `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.

/NOVERSION
/VERSION
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 TCPware NFS server. (See *Limiting File Versions*.)

/WRITE
/NOWRITE
Allows that you mount the filesystem either with write access (/WRITE – the default) 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]
-TCPWARE-I-OCCLUDED, previous contents of _NFS2:[USERS.SMITH] occluded
```

NFSMOUNT /CONFIG

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.

Format

`NFSMOUNT /CONFIG=`*`filespec`*

Parameter

`filespec`

OpenVMS file containing the configuration information. The contents of the file should include line entries in the format prescribed by the `NFSMOUNT` command:

```
server "nfs-path" mountpoint [logical] [qualifiers]
```

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.

Qualifiers

See the `NFSMOUNT` command for details on the supported qualifiers.

> **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.

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
```

NFSMOUNT /SHOW

Displays the mounted directories at all mount points or at a particular mount point.

Format

`NFSMOUNT /SHOW [`*`mountpoint`* `|` *`device`*`:]`

Parameters

mountpoint

Full NFS device name and directory tree for which to show mount information. For example:

```
NFS1:[USER.NOTES]
```

Alternately, you can use a logical name for the mount point.

```
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, NFSv2
  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    TCPware, NFSv2
  Directory time   0 00:00:30.00   Advisory Locking     Disabled
  Cache Validation   MODIFY TIME   Default user           [USER]
  Superuser                  No   Default UID,GID        100,15
```

NFSDISMOUNT

Dismounts an NFS mount point from the local device and directory structure.

Format

`NFSDISMOUNT [mountpoint | device:]`

Parameters

mountpoint

Full NFS device name and directory tree to dismount. For example:

```
NFS1:[USER.NOTES]
```

You can also use a logical name for the mount point. At the end of the `NFSDISMOUNT` operation, the client deletes the logical name from the job logical name table.

device:

NFS device name part of the *mountpoint* parameter (such as `NFS1:`). You can use the *device*: alone only with the `/ALL` qualifier.

Alternately, you can use a logical name for the device specification. TCPware considers only the NFS device part of the logical name.

Qualifiers

/ALL

Dismounts all filesystems from all servers, or a specified server or NFS device. The following options are available:

`NFSDISMOUNT /ALL`	Dismounts all filesystems from all servers
`NFSDISMOUNT /ALL /HOST=`*server*	Dismounts all filesystems on the specified server. (See the `/HOST` qualifier below.)

NFSDISMOUNT *device*: /ALL	Dismounts all filesystems on the specified device (such as NFS1:).

> **Note:** Dismounting a /SYSTEM mount requires SYSNAM privilege. Dismounting a /GROUP mount requires GRPNAM privilege.

/HOST=*server*
When used with the /ALL qualifier, dismounts all filesystems from the specified server. The /HOST qualifier is otherwise meaningless.

/WAIT
/NOWAIT
Specifies whether or not to dismount the mounted filesystem if there are still outstanding activities.

With /WAIT, the command waits until the client completes the dismount. If you try to open any files on the mount point, the dismount fails.

With /NOWAIT (the default), the client completes the command immediately. However, the dismount does not actually occur until all file activity has completed.

Examples

1. This example dismounts the specified mount point only. The client dismounts only [USR.MNT] and no other mount in the directory structure of the NFS3: device.

```
$ NFSDISMOUNT NFS3:[USR.MNT]
```

2. This example dismounts the NFS1:[000000] mount point and waits for it to occur.

```
$ NFSDISMOUNT NFS1: /WAIT
```

3. This example dismounts all mount points on all devices.

```
$ NFSDISMOUNT /ALL
```

4. This example dismounts all mount points served by host SIGMA.

```
$ NFSDISMOUNT /ALL /HOST=SIGMA
```

13. Managing NFS Server

Introduction

This chapter describes how to manage the NFS server. It includes the following topics:

- Server security
- Mounting directories on a client
- Network file locking
- Managing Server parameters
- Maintaining databases
- PCNFSD services and remote printing
- Filename mapping
- Server implementation of NFS protocols

Server Security

The 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.

You can take the following additional system security measures:

- Assign an NFS rights identifier to further restrict file access (see the `NFS_ACCESS_IDENTIFIER` logical under *Server Parameters*).
- Require all Remote Procedure Call (RPC) requests to originate from privileged ports.
- Restrict all remote mounts to the NFS superuser only.
- Restrict mounts only to explicit directories and not their subdirectories.

- Require the PROXY database to define the mount requester's identification (see the next section).

PROXY Database

The PROXY database maps OpenVMS user identification to NFS user identification. NFS user identification is different from that of OpenVMS in that it follows the UNIX model.

OpenVMS identifies users by a username and user identification code (UIC). The UIC consists of a group and a member number. An OpenVMS user can belong to only one group, which can have many members.

NFS follows the UNIX model in identifying users by user ID (UID) and group ID (GID) numbers. An NFS user can belong to many groups, and thus have several GIDs. Each NFS request includes the NFS user's effective UID, GID, or list of GIDs. You can find users' UIDs and GIDs in the `/etc/passwd` file on the UNIX client.

The server uses the PROXY database:

When an NFS user requests access to the OpenVMS filesystem	TCPware maps an NFS user's UID and GID to an OpenVMS username and UIC. The server uses the UIC to check file access permission.
When the NFS client requests file attributes from the server	The server maps the file owner's UIC to a UID/GID pair.
When a PC requests authentication using PCNFSD	The server uses the username and password to validate the user, and the UIC to find and return a UID/GID pair.

Maintaining PROXY

The server creates an empty PROXY database during installation. You maintain the PROXY database with the `ADD PROXY`, `CREATE PROXY`, `REMOVE PROXY`, and `SHOW PROXY` commands in TCPware's Network Control Utility (NETCU).

A PROXY database entry specifies an OpenVMS username and a corresponding NFS user's UID and GID. The `/HOST` qualifier with the `ADD PROXY` command also lets you specify the name of the hosts or hosts where the user account is valid.

The following example shows how to use the ADD PROXY command to assign the SMITH OpenVMS account to an NFS user with a UID=210 and GID=15 on host tulip:

```
$ NETCU
NETCU>ADD PROXY SMITH /UID=210 /GID=15 /HOST=TULIP
```

The PROXY database must contain an entry for each NFS user, including the superuser (see the next subsection).

When you add entries to the PROXY database:

- The OpenVMS username determines file access rights, not the NFS user's UID and GID. The NFS user account has the same access rights as are assigned the OpenVMS account.
- Assign each NFS user the same UID/GID on each NFS client. (See your NFS client documentation for details on global user ID space.)
- Avoid using wildcard UIDs or GIDs. A one-to-one mapping between OpenVMS users and NFS users is easier to maintain.
- Use the /HOST qualifier to allow access only to users from a particular host.
- For PCNFSD users, assign an arbitrary UID and GID for each PC user. Choose a unique UID for each user. Give the same GID to users that need to have group access to each other's files.

Adding Superusers

The superuser (or root) is a UNIX system user with UID=0 who can perform any operation on a file or process on the client system. However, the superuser cannot automatically access the OpenVMS filesystem on the server. The PROXY database must register a superuser.

The NFS convention is to replace the superuser's UID/GID pair (UID=0, and any GID) with the default values of UID=-2 and GID=-2. By UNIX conventions, this translates to user nobody, which gives the superuser limited access rights. To register a superuser, you must use UID=0 and GID=1, as follows:

```
NETCU>ADD PROXY DECNET /UID=0 /GID=1
```

The OpenVMS account to which you assign the superuser access rights determines what rights a superuser has on the OpenVMS system. Superusers require enough access rights so that they can mount directories. (In fact, some server configurations restrict mounting to the superuser). Also, when a user runs a setuid() to root program, the UID/GID in any resulting NFS request has the root UID and, therefore, requires superuser access.

You can create a PROXY entry for a superuser that provides limited access to an OpenVMS filesystem but still allows a superuser to mount exported directories. One example is the DECNET account. Alternately, you can use the OpenVMS AUTHORIZE command to add an account for the superuser on the OpenVMS host.

If you have trusted superusers at particular hosts and wish to give them full privileges on the OpenVMS system, add a separate superuser entry. Assign the superuser to a privileged account (such as SYSTEM) and use the /HOST qualifier to restrict access to a specified host. In the following example, only the superuser on lilac has SYSTEM account privileges:

```
NETCU>ADD PROXY SYSTEM /UID=0 /GID=1 /HOST=LILAC
```

Reloading PROXY

The PROXY database is normally static. This means that you have to reload the database every time you use ADD PROXY or REMOVE PROXY to change it. However, you can opt to update the PROXY database dynamically (make it dynamic). You can do so in two ways:

1. Define the TCPWARE_NFS_DYNAMIC_PROXY logical to enable dynamic PROXY database reloading, as follows:

   ```
   $ DEFINE/SYSTEM/EXEC TCPWARE_NFS_DYNAMIC_PROXY keyword[,keyword]
   ```

   The *keywords* are CLIENT, SERVER, NOCLIENT, and NOSERVER, used in any reasonable combination.

   Use CLIENT to enable client reloading and SERVER to enable server reloading. However, the /NOCLIENT and /NOSERVER qualifiers used with the ADD PROXY or REMOVE PROXY commands override the logical setting.

2. Use the /CLIENT or /SERVER qualifiers with the ADD PROXY or REMOVE PROXY commands. You can also mix and match by using /CLIENT with /NOSERVER, /NOCLIENT with /SERVER, and so on. Here is an example of its use:

   ```
   $ NETCU ADD PROXY SMITH /UID=210 /GID=5 /NOCLIENT /NOSERVER
   ```

If you disable PROXY database reloading on either the client or server, both methods require the RELOAD PROXY command. RELOAD PROXY is best used if you also specify a username parameter, so that you can reload for a specific username only. Otherwise, it reloads the entire database into memory each time. Therefore, it is best to use RELOAD PROXY at the initial configuration, and only sparingly thereafter.

EXPORT Database

The EXPORT database contains entries that specify an OpenVMS directory and the host or group of hosts allowed to mount that directory. More than one host can access a directory. The EXPORT

database differs from the PROXY database in that the server grants access to a host rather than to a user. If an OpenVMS directory is not in the EXPORT database, an NFS client cannot mount that directory.

An EXPORT database entry specifies a pathname for the OpenVMS directory. Because the OpenVMS device and directory specifications differ from those NFS clients use, the server lets you reference the OpenVMS directory by a UNIX-style pathname. You can assign any pathname to the OpenVMS directory.

> **CAUTION!** An authorized user at a remote host can access all subdirectories and files below the export point you specify. Unless you work in a trusted environment, do not export a top level directory, even though it may seem easier to do so. Export only the level of directories that the remote users need, and none higher.

Maintaining EXPORT

The server creates an empty EXPORT database during installation. You maintain the EXPORT database using the `ADD EXPORT`, `CREATE EXPORT`, `REMOVE EXPORT`, `RELOAD EXPORT`, and `SHOW EXPORT` commands in NETCU. For example, the following command places an entry in the EXPORT database:

```
NETCU>ADD EXPORT "/work/notes" $DISK2:[WORK.NOTES] -
_NETCU>/HOST=(ORCHID, ROSE)
```

This command exports the OpenVMS directory `$DISK2:[WORK.NOTES]` as path `/work/notes` to hosts ORCHID and ROSE. The pathname is an arbitrary one selected to reference the OpenVMS directory. The `ADD EXPORT` command requires that you enclose the pathname in quotes.

When a client mounts a subdirectory of an exported directory, each element in the path beyond the exported path must match the corresponding OpenVMS subdirectory name. Separate each element with a slash (`/`). For example, suppose the NFS client mounts:

```
$ DISK2:[WORK.NOTES.LETTERS.STUFF]
```

To match `/work/notes`, the NFS client uses this path:

```
/work/notes/letters/stuff
```

The NFS filename mapping rules apply to the path elements below the export point.

Reloading EXPORT

Updating the EXPORT database (using `ADD EXPORT` or `REMOVE EXPORT`) usually updates only the server on the host executing the command. You must use either the `RELOAD EXPORT` command, or restart all the other servers on the cluster to implement changes to the EXPORT database on them.

However, you can automatically reload updates to the shared database on the cluster by setting the `TCPWARE_NFS_DYNAMIC_EXPORT` logical to `CLUSTER`, as follows:

```
$ DEFINE/SYSTEM/EXEC TCPWARE_NFS_DYNAMIC_EXPORT CLUSTER
```

This causes the server to use locks to communicate changes to all the servers on the cluster. The default for `TCPWARE_NFS_DYNAMIC_EXPORT` is `LOCAL` (not to use locks).

EXPORT Options

The options you can specify while adding entries to the EXPORT database are as follows, using the indicated `ADD EXPORT` command qualifiers:

- If you want only specified host or hosts to access the exported OpenVMS directory, use the `/HOST=[`*`host`*`[,`*`host`*`,...]` qualifier.
- Whether or not to enable on-the-fly file conversion: `/[NO]CONVERT`
- Whether or not clients can mount subdirectories of a mount point: `/[NO]EXPLICIT_MOUNT`
- What kind of filename mapping you want to use: `/FILENAME=`*`option`*

  The NFS server includes the `UPPERCASE` keyword for this qualifier (**this does not apply to ODS-5 exports**). `UPPERCASE` changes the default case for exported filenames from lowercase to uppercase, for SRI filename mappings only. The full syntax of the command is

  ```
  $ NETCU ADD EXPORT /FILENAME=(SRI, UPPERCASE)
  ```

  Examples of filename conversions are as follows:

VMS Name	Lowercase	Uppercase
foobar.txt	foobar.txt	FOOBAR.TXT
$foobar.txt	FOOBAR.TXT	foobar.txt

foo$bar.txt	fooBAR.TXT	FOObar.txt

- Whether or not you want only the highest version files to appear in a directory request: /[NO]HIGHEST_VERSION
- Whether or not you want incoming requests to originate from a privileged port: /[NO]PRIVILEGED_PORT
- Whether or not you want mount requests to originate from a user mapped in the PROXY database: /[NO]PROXY_CHECK
- What kind of record format you want to use for newly created files: /RFM=*options*
- Whether or not you want the server (and not just the client) to perform file access checking: /[NO]SERVER_ACCESS
- Whether or not you want only the superuser to mount filesystems: /[NO]SUPERUSER_MOUNT
- Whether or not the filesystem should be read-only: /[NO]WRITE

Mounting Client Directories

NFS clients access OpenVMS files on the NFS server by mounting directories on the client. The MOUNT protocol services the mount request.

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 an example mount command provided by the TCPware NFS client:

```
$ NFSMOUNT IRIS "/WORK/RECORDS" NFS0:[USERS.MNT]
```

In the example, IRIS is the name of the OpenVMS server host. /WORK/RECORDS is the pathname of the exported directory. NFS0:[USERS.MNT] is the mount point on the OpenVMS client host.

Check your NFS client documentation before mounting directories. Mount commands and procedures vary by operating system. Chapter 12, *NFS Client Management* describes the client mount commands.

Network File Locking

The server supports file locking through its implementation of the Network Lock Manager (NLM) and Network Status Monitor (NSM) protocols. Many NFS client systems support file locking, even on the record and byte level, as long as the byte ranges do not overlap. File locking on the server is multi-threaded, where the server can satisfy more than one lock request at a time.

NFS file locking is only advisory. When a client requests a lock on a server file, the goal is for one of its processes to gain exclusive access to this file (or part of the file) and force other processes to wait until the original process releases the lock again. However, the only way NFS denies a client user access to a locked file is if the user also requests a lock on it.

There are two views on network file locking, one from the NFS client's viewpoint and one from the OpenVMS resident user's viewpoint. (See the following sections.)

NFS Client Users' View

When an NFS client user requests an advisory lock on a server filesystem, this sends a `lockd` request to the NLM of the server also running NFSD. This server checks its lock database to see if it can grant the lock. The server cannot grant the lock if:

- Another client has the same file (or region or byte range of the file) already locked.
- An OpenVMS user has the same file open for exclusive access.
- The server waits to reclaim locks during the grace period described below.

The server also includes a Network Status Monitor (NSM). The NSM cooperates with other status monitors on the network to notify the NLM of any changes in system status (such as when a crash occurs).

For example, if the server crashes and comes back up, the server NSM notifies the client NSM that it should resend requests for locks in place before the crash, within a certain grace period (usually 45 seconds). You can request new locks only after this grace period. However, if a client with mounted server files crashes, nobody knows to resend lock requests until the client comes back up again.

OpenVMS Users' View

To prevent OpenVMS users from accessing files that NFS clients have locked, the server's NLM requests NFSD to open these files for exclusive access. This essentially prevents all access to these files by OpenVMS users. When the client releases the lock by closing the file, the NLM requests NFSD to close the file, at which point OpenVMS users again access it.

If network file locking is to occur in a VMScluster environment, we advise exporting a filesystem from a single node in the cluster only. This way, only a single OpenVMS exclusive lock need occur. Client

users can then apply locks on files (or parts of files if enabled) without conflicting with exclusive locks applied from other nodes.

Mapping Filenames

Once you mount a filesystem, the server tries to make the client files recognizable in OpenVMS. Often the filename syntax for NFS files is very different from that of OpenVMS files. For example, NFS filenames do not include file version numbers.

The server translates (maps) filenames from the client so that your OpenVMS host can recognize and use them. Three types of mapping schemes are available:

- Stanford Research Institute (SRI) International mapping, the default scheme between NFS and OpenVMS systems
- PATHWORKS case-insensitive mapping (`PATHWORKS`)
- PATHWORKS case-sensitive mapping (`PATHWORKS_CASE`)

Set up the appropriate filename mapping scheme using the `/FILENAME` qualifier of the `ADD EXPORT` command in NETCU. If you do not specify the scheme using this qualifier, the server uses the SRI scheme by default.

The below table shows examples of how the server maps NFS directory names and filenames using the SRI mapping scheme. All the client files in the table are NFS files.

For the filename mapping rules, see Appendix A, *NFS-to-OpenVMS Filename Mapping*.

The filename mapping schemes for the server and the NFS client are identical and totally compatible.

Filename on server...	**Is mapped to filename on client...**
`SERVERFILE.;1`	`serverfile`
`$C$ASE$S$HIFTED$F$ILE.;1`	`CaseShiftedFile`
`DOT.FILE$5NTEXT;1`	`dot.file.text`
`DOT$5NDIRECTORY$5NLIST.DIR;1`	`dot.directory.list` (identified as a directory in the UNIX listing)
`SPECIAL$5CCHAR$5FFILE.;1`	`special#char&file`

DOLLAR$$$$IGN$$5CFILE.;1	dollar$Sign$5cfile

Protecting Files

The server protects an OpenVMS file by comparing its protection information with the user's identification and access rights. It then grants or denies access based on the results of these comparisons.

When an NFS user requests access to an OpenVMS file, the server uses the PROXY database to map the user's user and group identification (UID/GID) on the remote host to a username and UIC on the OpenVMS host. In most cases, this allows the NFS user to have the same access to files as the proxy OpenVMS user.

The NFS client can also do local access checking based on its user and file information and access checking rules before sending the request to the server host. In some cases, this results in the NFS user not having the same access to files as the proxy OpenVMS user.

The following sections explain how the server resolves differences between the two filesystems to provide the best possible mapping between client and server.

UIC Protection

The type of access an OpenVMS user has to a file depends how the file and user UICs are related.

OpenVMS has four file ownership categories: SYSTEM, OWNER, GROUP, and WORLD. Each category can have up to four access types: read (R), write (W), execute (E), and delete (D). Each file has a protection mask that defines:

- The categories assigned to the file
- The types of access granted to each category

Here is an example of an OpenVMS protection mask:

```
SYSTEM=RWED, OWNER=RWED, GROUP=RE, WORLD=<NO ACCESS>
```

UID/GID Protection

NFS uses a similar protection scheme as OpenVMS.

Each NFS user has a UID and GID. The file protection categories are: OWNER, GROUP, and OTHER, with the file access types of read (r), write (w) and execute (x). The NFS user's access to a file depends on how the file and owner UIDs/GIDs are related.

The server maps OpenVMS and NFS system protection masks and user identifications so that the relationship between a user and a file remains consistent. For example, if an OpenVMS user owns a particular file and an NFS user is mapped to the account through the PROXY database, the NFS client also considers the local user to be the owner of the file.

> **Note:** In OpenVMS, the owner of the file has absolute control over it. This also applies to files remote users create in the mounted filesystem.

OpenVMS-to-NFS File Attribute Mapping

When the NFS client requests the attributes of a file, the server maps:

- The protection mask to an NFS protection mask
- The owner UIC to a UID/GID

The below table shows how the server maps the protection mask from OpenVMS to NFS.

The server does not map the OpenVMS SYSTEM category and delete (D) access type because they do not exist in the NFS system environment.

The server maps OpenVMS execute (E) to NFS execute (x). However, the OpenVMS system uses the E access type more often than does NFS. Thus, some files might appear to be executable to an NFS host when they are not.

OpenVMS category...	In NFS is...	With OpenVMS type...	In NFS is...
SYSTEM	(not mapped)		
OWNER	user	R	r
		W	w
		E	x
		D	(not mapped)

GROUP	group	R	r
		W	w
		E	x
		D	(not mapped)
WORLD	other	R	r
		W	w
		E	x
		D	(not mapped)

The below table shows the rules the server follows to ensure that it correctly maps the UIC to the UID/GID.

If the server cannot find the UIC in the PROXY database, or the UID or GID are wildcards, the server returns the default UID or GID.

If the file's OWNER UIC...	**Then the Server Returns the...**
Matches the requesting NFS user's UIC	UID/GID of the requester
Group matches the requesting NFS user's UIC group	GID of the requester and returns the UID from the PROXY database
Does not match the requesting NFS user's UIC	UID/GID from the PROXY database

NFS-to-OpenVMS File Attribute Mapping

When the NFS client sets or changes the attributes of a file, the server maps the NFS file protection mask to an OpenVMS file protection mask.

The below table shows how the server maps the protection mask from NFS to OpenVMS.

NFS category...	In OpenVMS is...	With NFS type...	In OpenVMS is...
user	OWNER/SYSTEM	r	R
		w	W
		x	E
			D (unless ADF denies) [1]
group	GROUP	r	R
		w	W
		x	E
			D (unless ADF denies) [1]
other	WORLD	r	R
		w	W
		x	E
			D (unless ADF denies) [1]

[1]The server allows delete (D) access only if a special attributes data file (ADF) the server may create (and associates with the file) does not explicitly deny file deletion.

Access Control Lists

Access Control List (ACL) file protection is an OpenVMS feature that grants or denies access to a file based on a rights identifier.

If a file has an ACL, the OpenVMS system first uses the ACL for protection checking. If the ACL grants or denies access, OpenVMS goes no further. If the ACL does not grant or deny access, OpenVMS checks the protection mask.

NFS clients using the OpenVMS filesystem may encounter files or directories protected by ACLs. But since the ACLs are unique to the OpenVMS system, the NFS client only checks the protection mask. If the protection mask denies access, the NFS client does not attempt access, even if the file's ACL overrides the protection.

Because the NFS client uses only the protection mask, it is recommended that OpenVMS files protected by ACLs have:

- The ACL set to deny access
- The protection mask set to allow file access

This allows the NFS client to attempt access on the basis of the protection mask, and lets the OpenVMS system control whether access is granted or denied.

When an NFS user creates a file on the OpenVMS host and the directory has an ACL that specifies `+DEFAULT`, the new file gets the ACL of the directory.

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.

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 described in *Converting Files Manually*.

See the `NFS_DIRREAD_LIMIT` parameter in *Advanced Parameters*.

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.

Converting Files Manually

You can improve server performance by manually converting files using the OpenVMS `CONVERT` utilities described in this section.

Variable to STREAM_LF

Use this conversion procedure to make a variable-length file available to a UNIX system client without using the server's automatic conversion feature. To convert a variable-length record file to STREAM_LF, the command format is:

```
$ CONVERT/FDL=TCPWARE:STREAMLF source-file destination-file
```

The *`source-file`* specification is the variable-length record file. The *`destination-file`* specification is the name of the new file to contain the STREAM_LF records.

STREAM_LF to Variable

Use this conversion procedure to make a file created by a UNIX system client available to an OpenVMS application that does not understand the STREAM_LF record format. To convert a STREAM_LF file to variable-length, the command format is:

```
$ CONVERT/FDL=TCPWARE:VMSTEXT source-file destination-file
```

The *source-file* specification is the STREAM_LF file. The *destination-file* specification is the name of the new file to contain the variable-length records.

Variable to STREAM

Use this conversion procedure to make an OpenVMS variable-length file available to a PC client. Keep in mind that the server's automatic conversion procedure uses LF characters, not CRLF character sequences, for record terminators.

To convert a variable-length record file to STREAM format (with CRLF line terminators), the command format is:

```
$ CONVERT/FDL=TCPWARE:STREAMCRLF source-file destination-file
```

The *source-file* specification is the variable-length record file. The *destination-file* specification is the name of the new file to contain the STREAM records.

> **Note:** The variable-to-stream conversion does not add a Ctrl+Z to the end of the file. If a PC application requires the Ctrl+Z, use the conversion program the NFS client software provides.

Server Parameters

TCPware provides several basic parameters you can adjust to better suit your needs. To change the value of any of these parameters, invoke the network configuration command procedure (CNFNET) by entering the following command:

```
$ @TCPWARE:CNFNET NFS
```

The server also provides advanced parameters that you rarely need to change but appear here for reference purposes only.

The default parameter values appear in parentheses following the parameter name. All parameters are logicals and are static. When you make a change to a parameter, you must stop and restart the server for the change to take effect. TCPware uses logical names (the parameter names prefixed by `TCPWARE_`) to communicate the parameters to the NFS server. The `STARTNET` procedure defines these logicals.

Basic Parameters

The basic parameters described here are in the same order in which the server prompts you to provide values for them during the NFS configuration procedure. The default setting for each parameter appears in parentheses.

`NFS_ACCESS_IDENTIFIER` (`null`)	Specifies the name of a rights identifier you want assigned to all NFS users. You can then modify the access control lists (ACLs) of files to grant or deny access to holders of the rights identifier. The default is null (no rights identifier). OpenVMS files protected by ACLs should have the UIC-based protection mask set to allow file access and the ACL set to deny access. This lets the NFS client access on the basis of the protection mask, and lets the OpenVMS system control whether to grant or deny access.
`NFS_SECURITY` (`0`)	Enables various security features. This parameter is a bit mask value (in decimal) as defined in the *NFS_SECURITY Bit Mask Values* table below. The following global parameters supersede the values set using the corresponding qualifiers of the `ADD EXPORT` command, if applicable, as indicated in the *NFS_SECURITY Bit Mask Values* table below.
`NFS_LOG_CLASS` (`-1`)	Enables the type of information written to the log file `TCPWARE:NFSSERVER.LOG` This parameter is a bit mask value (in decimal), as defined in the *NFS_LOG_CLASS Bit Mask Values* table below.

NFS_SECURITY bit mask values:

Bit No.	Mask	Meaning When Set	Supersedes Qualifier for ADD EXPORT
0	1	Superuser mount enabled. Restricts remote mounts of the OpenVMS filesystem to the superuser UID (UID=0)	/[NO]SUPERUSER
1	2	Explicit mount enabled. You can only mount directories (and not their subdirectories) in the EXPORT database.	/[NO]EXPLICIT_MOUNT
2	4	Mount PROXY check. The UID and GID specified in all mount requests must exist in the PROXY database.	/[NO]PROXY_CHECK
3	8	Privileged port check. Requires that all incoming NFS requests originate from privileged ports on the client. Privileged ports are port numbers less than 1024.	/[NO]PRIVILEGED_PORT
4	16	Access checks for all files to the client performed by server only. The server reports mode 777 (octal). This allows full use of OpenVMS ACLs used to grant or deny file access. One implication is that the client reports an access mode of rwxrwxrwx for all files.	/[NO]SERVER_ACCESS
(remaining)		Reserved for future use.	

You cannot disable fatal errors and the server writes them to OPCOM. The default (-1) is all classes of information enabled.

NFS_LOG_CLASS bit mask values:

Bit...	Means when set...	Which are...
1	Warnings	Error recovery messages
2	MOUNT requests	MOUNT call messages
4	General	General operation messages
8	Security	Security violation messages
16	NFS errors	NFSERR_IO messages
(remaining)		Reserved for future use

Parameter	Description
NFS_DFLT_UID (-2), NFS_DFLT_GID (-2)	Specifies the default UID and GID. The server uses these defaults in the following cases: • The server receives a request from a user without a PROXY mapping and who is also the superuser (UID=0, and any GID). The server replaces the superuser UID and GID with the default UID and GID. • The server processes a get attributes request and cannot find a file's owner UIC in the PROXY database. The server uses the default UID and GID instead.
NFS_DIRLIFE_TIMER (:3)	Sets when to delete internal directory cache data structures. The server periodically scans these data structures and deletes them if a directory's cache has existed for longer than the NFS_DIRLIFE_TIMER value. This preserves memory. Specify the interval as OpenVMS delta time. The default is 3 minutes. If you are unfamiliar with delta time, see Chapter 13, *NFS Client Management, Client Commands*.
NFS_DIRREAD_LIMIT (-1)	Sets the maximum size in bytes for each file read while processing a get attributes request. If the estimated file size exceeds this value, TCPware does not read the file to determine its exact size and returns an estimated size instead. The estimated file size is always larger than the exact size. The -1 default effectively turns off file size estimation. This parameter applies only to filesystems exported with the /CONVERT option (the default). A value of 0 disables TCPware from determining exact file sizes on requests. This parameter may provide the NFS client with inexact file sizes. This is generally not a problem, but may affect some applications.
NFS_DIRTIME_TIMER (::30)	Sets a time interval that determines when the server updates the directory access time between NFS operations. Specify the interval as an OpenVMS delta time. The default is 30 seconds.

NFS_FILE_CACHE_SIZE (1024)	Determines the maximum number of files allowed to have attributes in cache at any one time. The number must be larger than the SYSGEN parameter CHANNELCNT. The value must also be larger than the number of combined TCP and UDP threads (see the NFS_TCP_THREADS and NFS_UDP_THREADS parameters).
NFS_NOCHECKSUM (0)	Enables or disables checksum generation for UDP datagrams. This parameter is a Boolean value. When the value is 0 (false), the server generates checksums for outgoing datagrams. When the value is 1 (true), the server does not generate checksums. Enabling checksums maintains data integrity, and is the default. **Note:** Disabling checksums may increase system performance but could have an adverse effect on certain NFS clients.
NFS_OPENFILE_TIMER (::6)	Sets a time interval (in delta time) a file remains open after you last accessed it. This can speed up request processing since a file can remain open for successive read or write requests. You do not need to open and close it for each request. The default is six seconds. You should not leave a file open for extended time, nor leave it open for too short an interval, which can decrease performance.

Advanced Parameters

You should not normally change the parameters described in this section. If you need to change a value for an advanced parameter, edit the TCPWARE_SPECIFIC:[TCPWARE]TCPWARE_CONFIGURE.COM file.

The advanced parameters that follow appear in alphabetical order. The default setting for each parameter is in parentheses.

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.

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 `TCPWARE:NFSSERVER.LOG` (as long as you enable security logging through the `NFS_LOG_CLASS` parameter).
- 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).

NFS Protocol Procedures

The server implements the following NFSv3 protocol procedures (while continuing to support NFSv2):

Procedures	**Description**
ACCESS `(access)` NFSv3 only	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)` NFSv3 only	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. • **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 server-initiated 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 TCPware 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

NFSv3 only	
GET FILESYSTEM STATISTICS (`statfs`) NFSv2 only	Returns filesystem statistics. The server handles certain file attributes in ways that are compatible with the OpenVMS system. These attributes are: **Block size**—The block size is 1024. **Total number of blocks**—The total number of blocks is the SYS$GETDVI MAXBLOCK parameter divided by 2. **Blocks free**—The number of blocks free is the SYS$GETDVI FREEBLOCK parameter divided by 2. **Blocks available**—The number of blocks available to unprivileged users is the same as the number of blocks free.
GET STATIC FILESYSTEM INFO (`fsinfo`) NFSv3 only	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 settaddr 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 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 modify 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 VMS 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.

NFSv3 only	
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	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.

(`symlink`)	
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. 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*).

Troubleshooting

If you are experiencing network communication-related problems on the NFS server, please check the following items:

1. Make sure TCPware is running on the OpenVMS system.
2. Make sure the server is running. If not, start it by entering the following command at the DCL prompt:

   ```
   $ @TCPWARE:STARTNET NFS
   ```

   If the server is not running but was started, examine the `TCPWARE:NFSSERVER.LOG` file. This file contains information to help you isolate problems with the server. After correcting any problems that were reported in the log file, restart the server.

3. 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 FTP or TELNET are not available on your system, try using the TCPware PING utility.
4. 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.

If you are experiencing problems performing NFS operations from a NFS client, check the server's `TCPWARE:NFSSERVER.LOG` file. It may contain messages that can help isolate the problem. A new `NFSSERVER.LOG` file may be created by typing `NETCU SET LOG/NEW/NFS`.

Certain messages can also come up with the `NETCU SHOW EXPORT`, `SHOW MOUNT`, and `UNMOUNT` commands.

Access error messages help by entering `HELP TCPWARE MESSAGES`.

14. Managing Print Services

Introduction

This chapter describes how to manage the TCPware print services, which include an Internet Printing Protocol (IPP) client, the Line Printer Services (LPS) client and server, and the Terminal Server Print Services (TSSYM).

Line Printer Services Client

You can configure an OpenVMS host with both an LPS client and a server. The LPS client lets users send print jobs to printers attached to remote hosts. It supports the UNIX-like LPR commands and the OpenVMS `PRINT` command. You can configure the LPS client to use:

UNIX-style LPR commands (`lpr`, `lpq`, and `lprm`)	During configuration, enter information about the default remote host and printer when you use an LPR command.
`PRINT`	Command used with one or more OpenVMS print queues on the client. TCPware creates and starts these queues during `STARTNET`. These queues can use: • OpenVMS `/FORM` features on the local or remote print queues. • The `/PARAMETERS` qualifier to achieve minimal formatting on remote print queues.

You can set up the print queues during TCPware configuration, or base the settings on entries in a local PRINTCAP (printer capability) database. The `PRINT` command supports two different print symbionts:

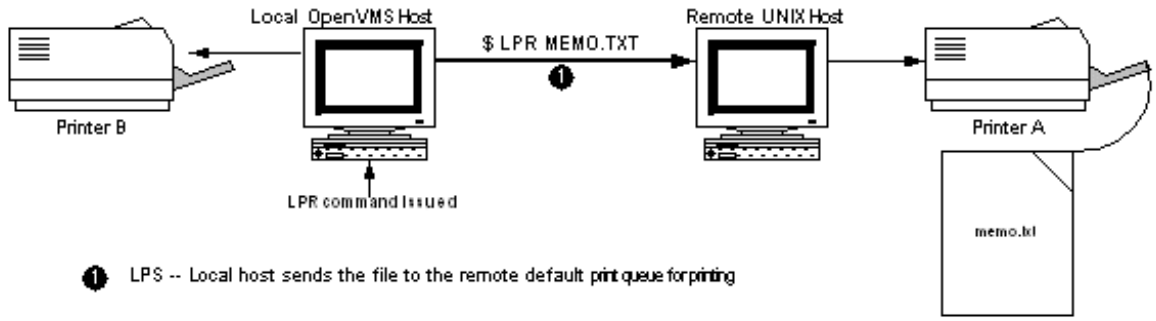
`TCPWARE_VMSLPRSMB`	Provides local print queue formatting.
`TCPWARE_LPRSMB`	Provides remote print queue formatting.

LPS Client Commands

The LPS client supports the following commands:

Command	Description	Command	Description
LPQ	Displays the remote print job status	LPRM	Removes a job from a remote print queue.
LPR	Sends a job to the default remote printer designated during configuration	PRINT	Places a job in the designated print queue; then sends the job to the printer associated with that queue.

The below figure shows how to use an LPR command with an LPS configuration.



OpenVMS Print Queues

The LPS client uses LPS print symbionts to control where document formatting and device control occurs: locally or remotely. The two symbionts (shown in the below table) provide different kinds of print format support, configurable during the CNFNET procedure.

Symbiont...	Supports the command...	And forms definition...
TCPWARE_VMSLPRSMB (provides local print formatting control)	PRINT/FORM	All DEFINE/FORM qualifiers
TCPWARE_LPRSMB (provides remote print formatting control)	PRINT/PARAMETERS and some PRINT/FORM	DEFINE/FORM /SETUP

		/STOCK /DESCRIPTION only

See the *TCPware User's Guide*, Chapter 5, *Network Printing*, and your OpenVMS documentation for information about the PRINT command and its qualifiers.

You can also configure the OpenVMS print queues to support additional qualifiers available with the OpenVMS INITIALIZE /QUEUE command. Use the /LIBRARY=LN03DEVCTL qualifier to enable the device control library. The device control library contains device control modules and resides in SYS$LIBRARY.

Print Forms

You must configure the print forms specifically for LPS to control the local or remote printer's setup for each print job. Use the OpenVMS DEFINE /FORM command format, with the qualifiers listed in the below table, as supported for each of the print symbionts:

```
$ DEFINE/FORM form-name form-number qualifier...
```

Qualifiers

/LENGTH=*n*
/MARGIN=(BOTTOM=*n*,LEFT=*n*,RIGHT=*n*,TOP=*n*)
/PAGE_SETUP=*module*
/SHEET_FEED,
/TRUNCATE,
/WIDTH=*n*,
/WRAP
Sets various page setup print parameters. (Use with TCPWARE_VMSLPRSMB only.)

/SETUP=*module*
Device control module or modules (separated by commas) that contain the print control sequences for the remote printer. (Use with TCPWARE_VMSLPRSMB or TCPWARE_LPRSMB.)

/STOCK=*string*
Type of paper stock to associate with the form (if not the same as the form-name); *string* can be up to 31 characters. (Use with TCPWARE_VMSLPRSMB or TCPWARE_LPRSMB.)

/DESCRIPTION=*string*
Operator information about the form (the default is the form-name) that appears when you issue the SHOW QUEUE/FORM command; use quotes to preserve case or if using spaces; *string* can be up to 255 characters. (Use with TCPWARE_VMSLPRSMB or TCPWARE_LPRSMB.)

PRINTCAP Database

During TCPware configuration, you can select whether to use the PRINTCAP (printer capability) database, if it exists, to start your local OpenVMS queues for the PRINT command. (Make any subsequent queue definitions during the usual LPS configuration.) The PRINTCAP database is the equivalent of the UNIX /etc/printcap file and resides locally in the TCPWARE:PRINTCAP file. If you do not have or opt not to use the PRINTCAP database to define local print queues, you must define these queues one by one during configuration.

The PRINTCAP database requires a special syntax. Each entry in the database describes one printer. According to the UNIX convention, each entry in the file is one or more lines consisting of fields separated by colon (:) characters. The first entry for each printer gives the name or names under which the printer is known, separated by vertical bar (|) characters. Entries can continue onto multiple lines by adding a backslash (\) after the last character of a line. You can include empty fields for readability.

You can use a number of Boolean, numeric, and string type options (or *capabilities*) with each database entry, although TCPware only supports three string type capabilities:

lp	Local printer's device name
rm	Remote machine name
rp	Remote printer name

An equal sign (=) separates the capability code from its value.

The below example shows a sample entry in the PRINTCAP. file.

```
#
# LOCAL PRINTERS
#
test_printer:\
  :lp=test_printer:\
  :rm=test1_printer:\
  :rp=eng2_printer:
```

In this example, the name of the local printer (`lp`) is `test_printer`. The remote machine (`rm`) is `test1_printer`, and the remote printer (`rp`) is `eng2_printer`. Lines starting with the pound sign (`#`) and blank lines are ignored.

To print to `TEST_PRINTER`, users specify:

```
$ PRINT/QUEUE=TEST_PRINTER filespec
```

The output appears on node EXAMPLE's `ENG2_PRINTER` printer.

> **Note:** The PRINTCAP database is not dynamic. To institute any changes you make to it, you must reconfigure the OpenVMS print queue using the configuration procedure.

LPS System Logicals

LPS uses several system logicals. TCPware defines only those LPS logicals required for features that you enable during CNFNET in the `TCPWARE_SPECIFIC:TCPWARE_CONFIGURE.COM` file. STARTNET uses the information in this file to create the logicals when you start the network. For example, TCPware defines logicals related to the LPD server only if you enable the server during CNFNET. Change features that you enable by rerunning CNFNET.

After you start the network, use the `SHOW LOGICAL` command in OpenVMS to examine the logical definitions. To set up a generic LPS client queue to print to a machine, set up the `TCPWARE_LPR_`*`qname`*`_PRINTER` logical for both the generic and server queues. The server queue automatically sets up the logical after you define it.

Client Logicals

The below table explains the purpose of each LPS client logical.

LPS logical	Description
`TCPWARE_LPR_QUEUES`	Lists the names of all TCPware print symbiont queues. Defined only if you defined one or more print queue.
`TCPWARE_LPR_`*`qname`*`_PRINTER`	Defines the absolute printer for the `PRINT` command. You cannot override this logical when submitting a

	print job. Use to restrict printing to one printer per queue.
TCPWARE_LPR_*qname*_PRINTER_DEFAULT	Defines a default remote printer for the PRINT command. Used if neither TCPWARE_LPR_*qname*_PRINTER nor the PRINT command specify a remote printer. You must define either TCPWARE_LPR_*qname*_PRINTER or TCPWARE_LPR_*qname*_PRINTER_DEFAULT for each queue for the PRINT command.
TCPWARE_LPR_SPOOL	Points to the work directory for the PRINT command. This directory holds temporary files.
TCPWARE_LPR_PRINTER	Defines the default remote printer for the LPR, LPRM, and LPQ commands. You can define your own TCPWARE_LPR_PRINTER logical in a LOGIN.COM file.

VMSLPRSMB Tuning Logicals

The TCPWARE_VMSLPRSMB print symbiont provides the retry interval and timeout tuning logicals (all are executive mode system logicals) listed in the below table.

VMSLPRSMB logical...	**Description**
TCPWARE_VMSLPRSMB_*_RETRY_INTERVAL	Defines the interval at which the symbiont retries to make a connection to a printer after an attempt fails. The connection can fail either by the remote printer rejecting it due to a busy state, or by the network timing out. The default value for a retry interval is 2 minutes (:2 in delta time). Note that a connection failure can take 1.5 minutes to time out, which is not included in this interval value.

TCPWARE_VMSLPRSMB_*qname*_RETRY_INTERVAL	Same as TCPWARE_VMSLPRSMB_*_RETRY_INTERVAL, but for a specific queue only, and overrides TCPWARE_VMSLPRSMB_*_RETRY_INTERVAL.
TCPWARE_VMSLPRSMB_*_TIMEOUT	Defines the time it takes for a print job to abort if the connection to the printer is never established. The default timeout is infinite (it never times out).
TCPWARE_VMSLPRSMB_*qname*_TIMEOUT	Same as TCPWARE_VMSLPRSMB_*_TIMEOUT, but for a specific queue only, and overrides TCPWARE_VMSLPRSMB_*_TIMEOUT.
TCPWARE_VMSLPRSMB_*qname*_PRECONN	Makes the connection to the printer *before* processing the file. Normal behavior is to make the connection to the printer *after* processing the file.

LPRSMB Tuning Logicals

The TCPWARE_LPRSMB print symbiont provides similar retry interval and timeout tuning logicals as those for TCPWARE_VMSLPRSMB (see the descriptions in the previous section). The TCPWARE_LPRSMB logicals are:

- TCPWARE_LPRSMB_*_RETRY_INTERVAL
- TCPWARE_LPRSMB_*qname*_RETRY_INTERVAL
- TCPWARE_LPRSMB_*_TIMEOUT
- TCPWARE_LPRSMB_*qname*_TIMEOUT
- TCPWARE_LPRSM_*qname*_PRECONN

Troubleshooting LPS

You may have a SETUP module in your LPS print queues (TCPWARE_VMSLPRSMB or TCPWARE_TSSYM) that causes the OpenVMS print symbiont to insert unexpected form feeds (<FF>).

You can remove these form feeds by adding an undocumented escape sequence to the SETUP module, as follows:

1. Start the SETUP module with the special escape sequence:

```
<ESC>]VMS;2<ESC>
```

2. Enclose the setup text with:

```
<ESC>Pstring<ESC>
```

For example, if you want to send the setup text `setup` to the printer, the `SETUP` module could look like this (or you could have two setup modules, one with the `<ESC>]VMS;2<ESC>F0>` text and the other with the `<ESC>Psetup<ESC>F0>` text):

```
<ESC>]VMS;2<ESC>Psetup<ESC>F0>
```

LPD Server

The LPD server on the local host accepts print requests from remote users. It then places the remote files in local OpenVMS print queues. You must define and initialize these OpenVMS print queues before you configure the TCPware LPD server.

Sending files from remote UNIX systems to a local OpenVMS printer requires the UNIX system to have an entry in an `/etc/printcap` file. Some UNIX systems do not have this file and rely on another method. (See your UNIX documentation for more information.) Here is a sample entry in an `/etc/printcap` file:

```
rpl | remote printer:
:lp=:
:sd=/usr/spool/lpd:
:rm=daisy:
:rp=ln03r$print
```

The following UNIX command puts `myfile` in the `ln03r$print` queue on daisy:

```
> lpr -Prpl myfile
```

Server Supported Options

The LPD server supports the options listed in the below table. (It does not support other options and ignores print requests you issue with such options, without issuing an error message.)

For command...	This option...	Does...
`LPQ`	`-l`	Displays the status of each job on more than one line

LPR	-C	Prints the job classification on the burst page (like the PRINT/NOTE command in OpenVMS)
	-f	Interprets the first character of each line as a standard FORTRAN carriage control character
	-h	Prevents the burst page from printing (like the PRINT/NOFLAG command in OpenVMS)
	-J	Prints the job name on the burst page (like the PRINT/NAME command in OpenVMS)
	-l	Prints control characters and suppresses page breaks (like the PRINT/PASSALL/NOFEED/NOHEADER command in OpenVMS)
	-m	Notifies the OpenVMS user when printing has completed for the job (like the PRINT/NOTIFY command in OpenVMS)
	-o	Indicates the file contains PostScript input
	-p	Prints the file with page headers (like the PRINT/HEADER command in OpenVMS)
	-v	Prints the Sun raster format file as a binary file with no formatting
	-x	Specifies not to require filtering before printing (like the PRINT/PASSALL/NOFEED/HEADER command in OpenVMS)
	-#	Prints multiple copies (like the PRINT/COPIES command in OpenVMS)
LPRM	-	Removes all jobs that only you own

Data and Control Files

The LPD server accepts only data files and control files from clients. Data files contain copies of the data you want printed or executed. Control files store the commands that specify how you want the data printed.

LPD receives the data and control files in STREAM-LF format unless you use `lpr -l` to send the print job to the printer. It stores the files in the spooling directory until the job ends. The `TCPWARE_LPD_SPOOL` logical points to the spooling directory.

LPD Access File

The LPD server requires an LPD access file. It checks this file before accepting any requests from remote clients. This file:

- Determines which remote hosts can access the local LPD server.
- Maps remote users to OpenVMS usernames.

You can create the LPD access file during or after TCPware configuration. Use any text editor to enter data in the file. If you create the file after configuring TCPware, give it the name `TCPWARE_COMMON:[TCPWARE]LPD_USERS.DAT`. Use the following format:

```
vms-username remote-host remote-user
```

vms-username	Username defined in the OpenVMS User Authorization File. Upper- or lowercase characters are acceptable. You can enter an asterisk (*) as a wildcard to designate the incoming user as the *vms-username*. Use a hyphen (-) to specifically disallow access to printing services.
remote-host	Host on which the remote user resides. Enter the full or partial domain name, or the IP address. (Using the IP address improves performance.) Upper or lowercase characters are acceptable. You can enter an asterisk (*) as a wildcard to designate all remote hosts. Do not partially wildcard the host name.
remote-user	Username on the remote host. Enter the username in the same case (upper- or lowercase) as the remote host uses to define it. You can enter an asterisk (*) as a wildcard. The wildcard maps all remote users to this *vms-username* account entry.

CAUTION! Use wildcards cautiously. They can cause severe security problems.

Use the exclamation point (`!`) or pound sign (`#`) as the first character of a line to indicate a comment line.

Include at least one space or tab between each item.

When the remote user tries to access the LPD server, LPD looks at `LPD_USERS.DAT` for valid username mapping. If a valid username mapping is not found, LPD checks the system logical `TCPWARE_LPD_DEFAULT_USER` to determine the OpenVMS username. If this system logical is not found, the LPD server discards and never prints the file. You define the OpenVMS username for this logical during network configuration.

When LPD receives a job from a remote system, the format of the print job's `NOTE` is:

```
remote-ID: user@host - note
```

- *`user`* - Remote username
- *`host`* - Remote hostname
- *`note`* - Note as specified on the LPR command, or the default (often the hostname)

Here is a sample LPD access file:

```
!vms-username    remote-host     remote-user
!-----------------------------------------
smith            daisy           smith
jones            daisy           jones
jones            rose            jones
harrington       192.168.95.1    harrington
harrington       tulip           harrington
wallace          *               wallace
harrington       iris            *
```

It is recommended you place wildcard entries later in the file, as the first acceptable mapping will be used.

The following example illustrates an access file which provides a specific mapping for remote user gertrude. It allows access to all users with matching names on both systems, and provides a default mapping for all other users on node daisy.

```
!vms-username   remote-host   remote-user
!--------------------------------------
-               daisy         thorn
rose            daisy         gertrude
*               daisy         *
daisy_default   daisy         *
```

In the first line, user thorn on system daisy is denied access to printing services.

In the second line, the remote user gertrude on daisy is mapped to the OpenVMS username rose.

In the third line, the LPD server is instructed to map, as is, usernames having corresponding OpenVMS accounts.

In the fourth line, if the remote user on daisy does not have a corresponding OpenVMS account on the local system, it is mapped to account DAISY_DEFAULT.

Batch Queues

The LPD server can place jobs in batch queues for execution. You enable this feature during network configuration. To send a job to a batch queue, specify the batch queue name instead of the print queue name when you enter the `PRINT` or `LPR` command.

The LPD server does not support qualifiers or options that are analogous to the following OpenVMS `SUBMIT` command qualifiers: `/CLI`, `/CPUTIME`, `/LOG_FILE`, `/PRINTER`, `/WSDEFAULT`, `/WSSEXTENT`, and `/WSQUOTA`.

LPD Logicals

The below table explains the purpose of each LPD server logical.

LPD logical...	**Description**
`TCPWARE_LPD_DEFAULT_USER`	Defines a default OpenVMS username for remote users connecting to the local LPD server. Used only when you define a remote host in the LPD access file and the remote username is not mapped to a specific OpenVMS username.
`TCPWARE_LPD_OPTIONS`	Determines if the server handles batch queues.
`TCPWARE_LPD_`*`qname`*`_FORM*`	Defines the form used for print jobs. This logical is similar to `TCPWARE_LPD_`*`qname`*`_PARAMETER`. Use the `TCPWARE_LPD_*_FORM` logical to define the form for all queues. Note that a specific queue setting overrides the global setting for that queue.

TCPWARE_LPD_*qname*_OPTION	Specifies additional PRINT command qualifiers to pass to the specified print queue: /BURST, /FEED, /FLAG, /FORM, /HEADER, /LOWERCASE, /PASSALL, /PRIORITY, /RESTART, /SPACE, /TRAILER Use the TCPWARE_LPD_*_OPTION logical to define the option for all queues. Note that a specific queue setting overrides the global setting for that queue.
TCPWARE_LPD_*qname*_PARAMETER*	Defines the specified parameters when the remote user submits a print request to the OpenVMS print system (*qname* is the queue name). The first equivalence string for the logical (if defined) is the first parameter; the second is the second parameter; and so on, for up to eight parameters. See *Using the LPD Logicals to Support a Printer*. Use the TCPWARE_LPD_*_PARAMETER logical to define the parameter for all queues. Note that a specific queue setting overrides the global setting for that queue.
TCPWARE_LPD_*qname*_QUEUE*	Defines the print queues for an alias queue name (*qname*). Typically supports clients that may not allow standard OpenVMS queue names as the remote printer (such as IBM's AIX, which restricts remote printer names to seven characters). See *Using the LPD Logicals to Support a Printer*.
TCPWARE_LPD_SPOOL	Points to the work directory for the LPD server. This directory holds temporary files.

* STARTNET does not define the TCPWARE_LPD_*qname*_QUEUE logical. Define this system logical in the system startup file.

Be aware of security when you define the TCPWARE_LPD_DEFAULT_USER logical. Remote users can submit batch jobs to your local OpenVMS host. To prevent unauthorized users from submitting batch jobs, do not define a username belonging to a privileged account (such as the SYSTEM username). Use AUTHORIZE instead to create a special user account with restricted access.

The below example shows how to use the TCPWARE_LPD_*qname*_PARAMETER and TCPWARE_LPD_*qname*_QUEUE logicals to support an LN03R PostScript printer. These logical definitions define two alias queues (LP0 and PS0) for the LN03R printer queue, $LN03R1, and define the parameters for these queues. The LP0 queue prints jobs submitted as ASCII files. The PS0 queue prints jobs submitted as PostScript files.

```
$ DEFINE/SYSTEM/EXEC TCPWARE_LPD_LP0_QUEUE $LN03R1
$ DEFINE/SYSTEM/EXEC TCPWARE_LPD_PS0_QUEUE $LN03R1
$ DEFINE/SYSTEM/EXEC TCPWARE_LPD_LP0_PARAMETER "DATA=ANSI"
$ DEFINE/SYSTEM/EXEC TCPWARE_LPD_PS0_PARAMETER "DATA=POST"
```

Troubleshooting LPD

Facilities to aid in resolving problems you may encounter with LPD include:

- OPCOM error messages (OPCOM)
- LPD log files

LPD sends messages to OPCOM under some error conditions.

Access error messages help by entering **HELP TCPWARE MESSAGES**.

LPD also writes the OPCOM messages and several other informational messages to the following LPD log file, shared by all LPD servers: TCPWARE:LPDSERVER.LOG. It is often useful to review the messages in this log file.

You can also obtain more details about LPD processing by using the Network Control Utility (NETCU) to specify an output file for SYS$OUTPUT for the LPD server. Normally, LPD's output goes to NLA0: (the null device) and is, therefore, lost. By redirecting the output to a log file, you can examine a detailed trace of LPD's execution:

```
$ NETCU MODIFY SERVICE printer TCP /OUTPUT=file
```

Terminal Server Print Services

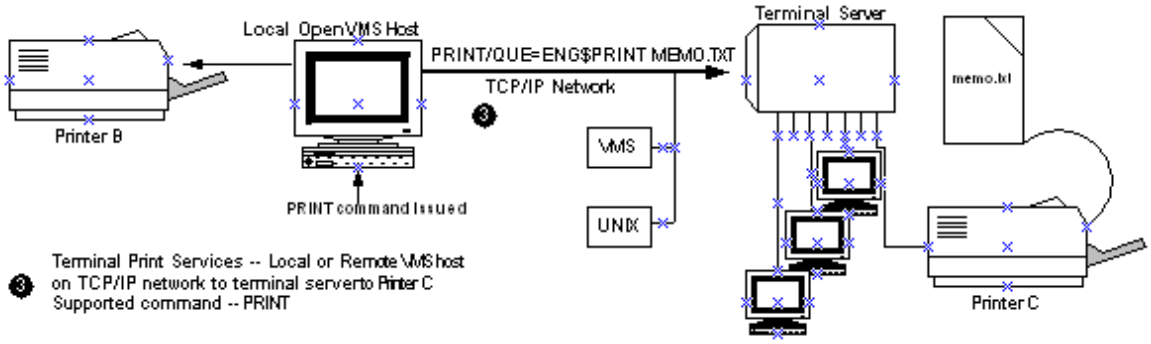
Terminal Server Print Services (TSSYM) provide an efficient way for OpenVMS users to send print requests to printers attached to TCP/IP-based terminal servers. Users on the host can easily access printers attached to a terminal server as if they were any other OpenVMS printer.

You can configure the print queues to the remote printer using standard OpenVMS print operations. Users can initiate print jobs with the usual OpenVMS commands. The below diagram shows using the `PRINT` command with a TSSYM configuration.

A special print symbiont sends the print request over the network instead of writing it to a local printer. The symbiont performs all the necessary device-related functions on the remote terminal server. These include allocating the device, assigning a channel to it, obtaining its device characteristics, and determining its device class.

For further information on setting up print queues and initiating print commands on the OpenVMS host, see your OpenVMS documentation.

The symbiont processes the data so that the terminal server can pass the job to the printer. Unless you keep the connection open, the symbiont also closes the TCP connection to the terminal server after every print job and opens a connection with a new terminal server.



TSSYM Print Queue

Initializing the terminal server print queue requires OPER privileges. Starting the queue requires OPER privileges or execute (E) access to the queue.

You can set up the terminal server print queue as you would any other OpenVMS printer queue, using standard OpenVMS queue commands. If you want to modify the terminal server printer queue, specify any standard printer queue commands and options the `INITIALIZE/QUEUE`, `START/QUEUE`, and `SET QUEUE` commands support. When you set up the queues, the terminal server queues must specify:

`/PROCESSOR=TCPWARE_TSSYM`	`TCPWARE_TSSYM` identifies the special print symbiont.
`/ON=`*`string`*	*`string`* specifies the terminal server information using the *`host`*, *`port`*, *`options`*, and *`qname`* parameters.

/AUTOSTART_ON	implements an autostart queue (if enabled) on one or more nodes if used together with the TCPWARE_TSSYM_*qname* logical.

Initialize the print queue and then start it. Add the command lines shown below to your printer startup commands or in your system startup file (SYSTARTUP_V5.COM or SYSTARTUP_VMS.COM).

```
$ INIT/QUEUE/PROCESSOR=TCPWARE_TSSYM/ON="host,port[,options]" qname
$ START/QUEUE qname
```

host	With /ON, Internet address or host name (if in the HOSTS. file or resolvable using DNS) of the terminal server
port	With /ON, port number on which the terminal server accepts incoming connections for the printer port (A below table shows some common port numbers)
options	Specify the /ON options listed in the below table singly or in the combinations given (each option separated by a comma).

/ON options:

Option	**Description**
EXPNTAB	Expands <TAB> characters to be the equivalent number of <SPACE> characters in print files. TSSYM normally ignores <TAB> characters.
KEEP	Keeps the connection open between jobs and closes it only on errors or when exiting. Prevents several systems from sharing the printer. This eliminates opening and closing the TCP connection with every print job, thereby not tying up network resources. Do not combine with NOFF.
LOWER	Marks the queue as supporting lowercase printing.

NOCRNUL	The TELNET standard (RFC 856) requires that a carriage return character (<CR>) not followed by a line feed character (<LF>) be converted to <CR><NULL>. TCPware supports this standard. Use the NOCRNUL option to disable the character sequence conversion for printers that do not support the TELNET standard.
NOFF	Does not send a form feed to the printer for each new connection. However, NOFF still sends both a form feed and a carriage return for the first file printed after you initialize the queue. Do not combine with KEEP.
NOINIFF	Suppresses an initial form feed before the very first job after queue startup. Uses the sequence <CR><CR> instead of <CR><FF>.
NOOPCOM	Suppresses OPCOM messages the terminal server print symbiont may produce.
RAW	Makes the connection a raw, binary connection, not a TELNET connection. TCPWARE_TSSYM does not double IAC characters (ASCII 255) in the data stream. Also, <CR> is not converted to <CR><NULL>.
TRIMFF qname	A print job normally ends with a carriage return (<CR>) and form feed (<FF>). Using the TRIMFF option, you can prevent the symbiont from adding these to the end of the print job. TRIMFF also replaces <CR> and <FF> with <CR><CR> at the beginning of the print job. Name of the print queue.

Common printer port numbers:

Printer or server...	**Is given port number...**
Emulex NetQue Print Server	2501

Emulex NetQue Serial Card	2502
HP Jet Direct Card	9100
Lantronics	20*nn*, where *nn* is the port number
Racal	100*n*, where *n* is the port number
Xylogic	70*nn*, where *nn* is the physical port on the terminal server
Xyplex 720 Terminal Server	2000 + (100 *nn*), where *nn* is port number; for example, port 14 would be 3400 for the TCP/IP listener port

Here is a typical command sequence to set up a standard ANSI printer:

```
$ INIT/QUEUE/PROCESSOR=TCPWARE_TSSYM/ON="192.168.25.50,2005" PR1
$ START/QUEUE PR1
```

If you use more than 31 characters for `/ON` qualifier parameters (including the quotes), the message `%QUEMAN-F-INVQUAVAL, value '"host,port,options"' invalid for /ON qualifier` appears. If you need to use more than 31 characters, define the `TCPWARE_TSSYM_qname` logical, described in *Autostart Queue*.

The standard OpenVMS qualifiers for the `INITIALIZE` and `START` commands are available. You can also set up a generic print queue where you can move print jobs to compatible execution queues, so that you can print to the first available printer on a `SYS$PRINT` request.

You need to start TCPware to print to the printer connected to the terminal server. If you do not start TCPware, the printer is down, or the system does not recognize the domain name, the print symbiont waits until you resolve the problem. This puts the print queue into a "stalled" state. In this case, you can either correct the problem while the queue is up, or stop and restart the queue using `STOP/QUEUE/RESET` and `START/QUEUE`.

Spool Device

You can set up a spool device so that you can use TSSYM to associate the device with a print queue and then perform operations such as copying a file to the device. HP DATATRIEVE is an application that could use this functionality:

1. Set up a print queue, such as with the typical command sequence shown earlier.
2. Use SYSGEN to set up the spool device. Select a new device (such as QPA0:) and use it in the CONNECT command with the /DRIVER=FTDRIVER and /NOADAPTER qualifiers. Then specify the queue name from step 1 in the SET DEVICE /SPOOLED command, as in the following example:

```
$ RUN SYS$SYSTEM:SYSGEN
SYSGEN>CONNECT QPA0: /DRIVER=FTDRIVER /NOADAPTER
SYSGEN>EXIT
$ SET DEVICE QPA0: /SPOOLED=PR1
```

3. Copy a file to the device, or define the system output to be the device and run a program whose output goes to it, such as the following:

```
$ COPY FOOBAR QPA0:
$ DEFINE SYS$OUTPUT QPA0:
$ RUN PROGRAM1       !output from PROGRAM1 goes to the device
```

Autostart Queue

You can set up an autostart queue on a node that automatically starts up again after it stops. You can also set up such a queue to autostart on other failover nodes.

Starting an autostart queue requires the /AUTOSTART_ON qualifier for the INITIALIZE/QUEUE command. Since you cannot use /AUTOSTART_ON together with the /ON qualifier to initialize a terminal server print queue, you need to define the TCPWARE_TSSYM_*qname* logical for this purpose. This logical defines the parameters normally set with the /ON qualifier.

The format of the logical definition is:

```
$ DEFINE/SYSTEM TCPWARE_TSSYM_qname "host,port[,option...]"
```

The format of the /AUTOSTART_ON qualifier (use the parentheses when specifying multiple nodes):

```
/AUTOSTART_ON=(node::[,node::,...])
```

The below example shows a typical command sequence to define the TCPWARE_TSSYM_*qname* logical, initialize and start up an autostart queue (QUEUE1) on two nodes, and enable autostart on these nodes. You can also add the commands to your startup command procedure. Note that there are two nodes: NODE2 can be a failover node in case NODE1 goes down.

```
$ DEFINE/SYSTEM TCPWARE_TSSYM_QUEUE1 "192.168.25.50,2005,KEEP"
$ INIT/QUEUE /START /PROCESSOR=TCPWARE_TSSYM -
_$ /AUTOSTART_ON=(NODE1::,NODE2::) QUEUE1
$ ENABLE AUTOSTART /QUEUES /ON=NODE1
$ ENABLE AUTOSTART /QUEUES /ON=NODE2
```

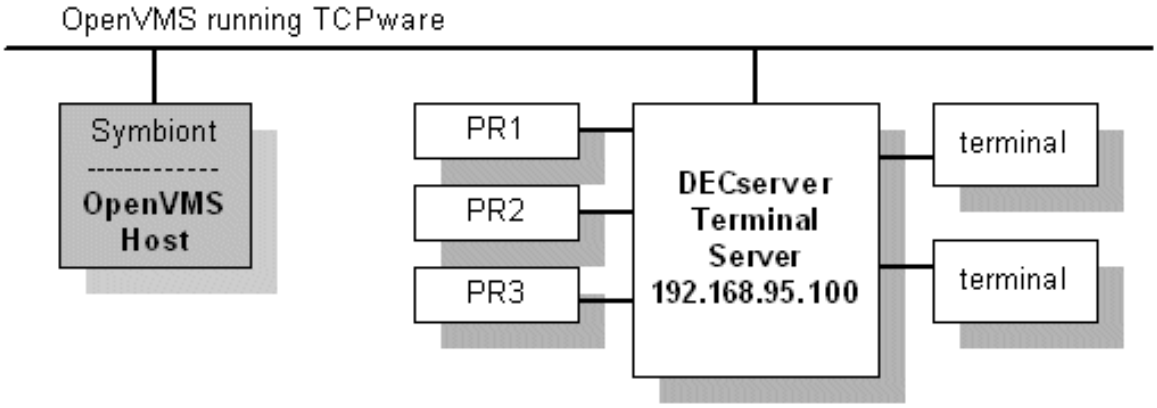
Sample TSSYM Configuration

A sample configuration includes a host connected to a DECserver 300 terminal server, as shown in the below diagram.

The following example shows how to configure the OpenVMS host to process print requests to the PR1 printer on the DECserver 300. The procedure then shows how to set up the queue and execute a print request.

For details on configuring a TELNET port and the recommended settings for access through a TELNET listener for a specific printer, see HP's *DECserver 300 Management* guide.

The setup values in the next example are for the DECserver 300 terminal server only. For setup values specific to your terminal server, see your server documentation.



The steps in the sample configuration and startup are as follows:

1. Initialize the DECserver 300 terminal server.
2. At the `Local>` prompt, enable privileged status to access all terminal server commands.
3. Configure the terminal server port for the printer. For example, if printer PR1 is connected to port 5:

```
Local>define port 5 access remote autobaud disabled
Local>define port 5 break disabled
Local>define port 5 character size 8 dedicated none
Local>define port 5 dsrlogout disabled
Local>define port 5 flow control xon inactivity logout enabled
Local>define port 5 parity none
Local>define port 5 password disabled preferred none
Local>define port 5 signal check enabled
Local>define port 5 signal control disabled speed 9600
Local>define port 5 type hardcopy
Local>logout port 5
```

4. Configure the TELNET server characteristics of the terminal server port for the printer. For example, to set up TELNET server carriage control handling:

```
Local>define port 5 telnet server newline to host <CRLF>
Local>set port 5 telnet server newline to host <CRLF>
```

5. Configure the TELNET listener port to associate the listener with the printer port. The valid TCP listener port numbers for the DECserver 300 are 2001 through 2016. For example:

```
Local>define telnet listener 2005 ports 5 enabled
Local>set telnet listener 2005 ports 5 enabled
Local>define telnet listener 2005 identification "PR1"
Local>set telnet listener 2005 identification "PR1"
Local>define telnet listener 2005 connection enabled
Local>set telnet listener 2005 connection enabled
```

6. On the OpenVMS host, initialize and start up the print queue, as follows:

```
$ INIT/QUEUE/PROCESS=TCPWARE_TSSYM/ON="192.168.95.100,2005" PR1
$ START/QUEUE PR1
```

OpenVMS users can now issue print commands to the printer, such as:

```
$ PRINT/HEADER/QUEUE=PR1/COPIES=10 TEST1, TEST2, TEST3
```

TSSYM Tuning Logicals

TSSYM provides the retry interval and timeout tuning logicals (all are executive mode system logicals) listed in the below table.

TSSYM logical...	**Description**
`TCPWARE_TSSYM_*_RETRY_INTERVAL`	Defines the interval at which the symbiont retries to make a connection to a printer after an attempt fails. The format must be `D h:mm:ss`. The default is `0 0:00:15` (15 seconds delta time).
`TCPWARE_TSSYM_qname_RETRY_INTERVAL`	Same as `TCPWARE_TSSYM_*_RETRY_INTERVAL` but for a specific queue only, and overrides `TCPWARE_TSSYM_*_RETRY_INTERVAL`

TCPWARE_TSSYM_*_TIMEOUT	Defines the time it takes for a print job to abort if the connection to the printer is never established. The default timeout is infinite (it never times out).
TCPWARE_TSSYM_qname_TIMEOUT	Same as TCPWARE_TSSYM_*_TIMEOUT, but for a specific queue only, and overrides TCPWARE_TSSYM_*_TIMEOUT.

Troubleshooting TSSYM

OPCOM can send a number of status messages that can help you troubleshoot TSSYM.

Access error messages help by entering HELP TCPWARE MESSAGES.

Internet Printing Protocol (IPP)

The IPP print symbiont is an OpenVMS print symbiont working with the OpenVMS printing subsystem to implement an IPP client. It allows printing over a network to printers and servers that support the IPP v1.0 network printing protocol. The user interface is similar to other print symbionts in that it uses PRINT commands or system library calls to submit jobs to print queues. The IPP protocol has specific qualifier values and queue settings that must be present to allow the symbiont to function. This section describes both the configuration of IPP print queues and the use of the PRINT command. For information on submitting jobs to print queues using system library calls, see the appropriate OpenVMS documentation.

IPP Protocol Background

The Internet Printing Protocol solves a number of problems in existing network printing protocols; the most common is the LPD protocol, originally implemented on UNIX operating systems.

From RFC 2568:

> "The Internet Printing Protocol (IPP) is an application level protocol that can be used for distributed printing on the Internet. This protocol defines interactions between a client and a server. The protocol allows a client to inquire about capabilities of a printer, to submit print jobs and to inquire about and cancel print jobs. The server for these requests is the Printer; the Printer is an abstraction of a generic document output device and/or a print service provider. Thus, the Printer could be a real

printing device, such as a computer printer or fax output device, or it could be a service that interfaced with output devices."

IPP has a better error reporting capability than LPD or TELNET. It supports multi-sided printing, landscape/portrait layouts, and multiple pages per physical sheet ("number-up") printing. Because not all printer models that support IPP will support all capabilities, the IPP protocol includes a way for the symbiont to query the printer as to its capabilities before a job is sent. If the printer cannot handle a given request, the job is aborted with an error status. The error status appears in the system accounting log.

IPP uses the HTTP 1.1 protocol as its transport layer; however, it has its own reserved port number, port 631. You can use the IPP Symbiont to print to other port numbers, including the standard HTTP port (80), but you need to specify the port number as part of the printer URL if the port number is not the default IPP port number. If you are printing through a firewall this could be a factor to consider. For a full description of the IPP protocol, see the relevant RFCs listed below.

Relevant RFCs

The RFCs related to IPP v1.0 are:

Design Goals for an Internet Printing Protocol	RFC 2567
Rationale for the Structure and Model and Protocol for the Internet Printing Protocol	RFC 2568
Internet Printing Protocol/1.0: Model and Semantics	RFC 2566
Internet Printing Protocol/1.0: Encoding and Transport	RFC 2565
Internet Printing Protocol/1.0: Implementer's Guide	RFC 2639
Mapping between LPD and IPP Protocols	RFC 2569

Additional RFCs are referenced by these, such as the ones describing HTTP v1.1, MIME Media Types, etc. The specific RFCs are called out in the above documents.

Limitations of this Implementation

The IPP symbiont implements a subset of the IPP v1.0 protocol consisting of all required portions and several selected optional features. Note that not all features are available on all printers; most printers implement a subset of the available protocol capabilities.

Not all printers claiming to support IPP implement IPP correctly. Some use supersets of HTTP 1.0, rather than the required HTTP 1.1. Some do unusual things with TCP/IP connections, such as having extremely short timeouts. The symbiont has been adapted to support as many of these inconsistencies as possible (see the EXPECT_LINK_CLOSURES option for an example). The symbiont may or may not behave as expected with such printers depending on your particular network characteristics and exactly what the printer manufacturer has done differently from what is specified in the RFCs. The symbiont should work with any fully compliant IPP v1.0 printer or server.

Configuration

The IPP symbiont has a flexible configuration. You can supply the information in the queue setup itself, as the /DESCRIPTION string which is supported by OpenVMS as part of the INITIALIZE/QUEUE command. You can supply the information in a "configuration" system logical name that the symbiont checks. You can use both, putting some information on one place, and some in the other. You can also put configuration information in one or more files and reference those files from the /DESCRIPTION string and/or "configuration" logical name (see the /INCLUDE option), or even from other such files. If you have large numbers of queues making up complicated groupings with similar requirements, this flexibility can help reduce the time required to set up and maintain queues.

In addition to the basic configuration information, there are several optional logical names used to control specific behaviors. Note that the default behaviors may be adequate to your needs.

The following sections describe the various logical names, queue settings, and PRINT command options available with the IPP symbiont. In many cases there is a global version, that affects all IPP symbiont queues on the system, as well as a queue-specific version that affects only a specified queue. PRINT command options affect only the job being submitted.

Global Settings

These logical names establish configuration values for all queues on the system, not on a queue-by-queue basis. Where there are queue-specific settings related to these, these become the default values, overriding any built-in defaults.

TCPWARE_IPP_CONFIG

Specifies one or more of the qualifiers described in the *Queue-specific Settings* section. These qualifiers are not case sensitive. Underscores (_) in the qualifier names are optional. Each may be abbreviated as long as the result is not ambiguous. There is no default. This logical provides defaults that may be overridden by the queue-specific configuration logical, TCPWARE_IPP_*queuename*_CONFIG, for a given queue.

TCPWARE_IPP_DEFAULT_DOCUMENT_FORMAT
Specifies a string to use as the document format, unless specified differently for a given queue or print job. The actual document format used on a given job must be a valid MIME media type, supported by the printer to which the job is sent. The default is `text/plain`.

TCPWARE_IPP_DOCALIAS_FILE
Specifies document format name aliasing. Rather than having to specify long mime-media-type names for document formats, you can define local names that are equivalent, and the symbiont will do the replacement. For example, you can define **PS** as equivalent to `application/postscript`, and use it in print commands as `/DOCUMENT_FORMAT=PS`. There is an escape mechanism in case a local name is ever made into a different MIME-media-type. Prefixing the document format name with `%` prevents alias translation. `%PS` means just send it as `PS`, do not translate `PS` into `APPLICATION/POSTSCRIPT` in the request.

To use aliasing, define the system logical name `TCPWARE_IPP_DOCALIAS_FILE` with the filename of the alias file as the equivalence string. The format of the alias file is:

```
MIME_media-type: alias, alias, alias...
```

Blank lines are ignored. Lines starting with # are treated as comments and are ignored.

TCPWARE_IPP_IGNORE_DESCRIPTION
TCPWARE_IPP_*queuename*_IGNORE_DESCRIPTION
If this logical is defined, the symbiont ignores the `/DESCRIPTION` strings for all IPP queues. This allows use of `/DESCRIPTION` for other information without affecting the symbiont. Configuration of the symbiont must be done through use of the `TCPWARE_IPP_CONFIG` logical, or the queue-specific logical, `TCPWARE_IPP_queuename_CONFIG` if `TCPWARE_IPP_IGNORE_DESCRIPTION` is defined. The value of the equivalence string for `TCPWARE_IPP_IGNORE_DESCRIPTION` is not important. The existence or non-existence of the logical is all that is checked. This logical provides defaults that may be overridden by the queue-specific configuration logical, `TCPWARE_IPP_queuename_IGNORE_DESCRIPTION`, for a given queue.

TCPWARE_IPP_JOB_RETRY_DELAY
Specifies, as an OpenVMS delta time specification, the length of time to hold a job when it is re-queued due to a temporary problem. The default value is "0 00:10:00.00" (10 minutes).

TCPWARE_IPP_MAX_LOG_BYTES

Specifies how many bytes of data will be logged by the send and receive routines when running with logging level set to DETAILED_TRACE. The value is an integer. A negative value sets the limit to infinite (all data will be logged). A value of zero turns off inclusion of data to the log file. A positive value sets the actual number of bytes logged, and any additional data is ignored. The default action is to log all data.

TCPWARE_IPP_MAX_STREAMS
Specifies the number of streams (queues) that each IPP symbiont process can handle. This is an integer from 1 to 16. The default is 16.

TCPWARE_IPP_LOG_LEVEL
Specifies one of the logging levels values listed in *Logging Levels*, and is used to determine how serious a message must be before it is written to the log file. Only those messages marked as this level, or as a more serious level, are logged. The default is JOB_TRACE.

TCPWARE_IPP_LOGFILE
Specifies the name of the log file. All queues for a given symbiont process will share this file unless there are individual queue overrides. The default is to create the log file in the default spool directory, with the name IPP_SYMBIONT_*pid*.LOG.

TCPWARE_IPP_OPCOM_LEVEL
Specifies one of the logging levels values listed in *Logging Levels*, and is used to determine how serious a message must be before it is sent to OPCOM. Only those messages marked as this level, or as a more serious level, are sent. The default is INFO.

TCPWARE_IPP_OPCOM_TERMINAL
Specifies the OPCOM operator "terminal" to send OPCOM messages to. Permissible values are listed later in this section. The default is the PRINT operator.

Queue-specific Settings

These items are specified as qualifiers in the queue's /DESCRIPTION string, and/or in the TCPWARE_IPP_*queuename*_CONFIG logical equivalence string. The two are concatenated before

being processed. These qualifiers are not case sensitive. The underscores in the qualifier names are optional. Each may be abbreviated as long as the result is not ambiguous. The two sections below contain the complete list of qualifiers.

Queue-specific Required Qualifier

/PRINTER_URI
A valid URI, or list of URIs, for the printer or printers to be sent to from this queue. Wildcards are allowed ("*" to match one or more characters, "?" for a single character). The individual URIs in the list are separated from each other with the vertical bar ("|") character. The first URI in the list that does not include any wildcards is the default printer for the queue. If there are no default printer URIs and you have not specified a particular printer URI with the PRINT command, the job is aborted. Any printer URI specified with the PRINT command must match at least one of the URIs listed for the queue or the job will be aborted.

Queue-specific Optional Qualifiers

/COMMENT=*quoted string*
Allows inclusion of a quoted string of text that the symbiont will ignore, other than to write to the log file and/or OPCOM if the logging level is set to SYMBIONT or a more detailed setting.

/COPIES_DEFAULT=*number*
Specifies the number of copies of each document to print unless specified otherwise on the PRINT command. The default value is 1.

/DEBUG
Causes the symbiont to retain all spool files and to force DETAILED_TRACE logging to the log file, regardless of what other settings might be specified. Note that /DEBUG forces the setting for MAX_LOG_BYTES to a minimum of 512 bytes. You can set it higher, but any setting lower than 512 bytes will be ignored when /DEBUG is used.

/DEFAULT_DOCUMENT_FORMAT=*formatspec*
/DOCUMENT_FORMAT_DEFAULT=*formatspec*
Specifies the default document format for the queue. This value will be a MIME media type that is supported for the printer or printers served by this queue. It could also be the string ***printer_default***, which will result in whatever the target printer defines as its default when no document format is specified on the PRINT command.

/EXPECT_LINK_CLOSURES
Specifies that the printer is not fully HTTP 1.1 compliant because it does not support persistent connections, and does not send a "Connection: Close" header line in its last response. Therefore, the symbiont should assume that such a line was sent in every response, using a new link for each request, closing the old one, and not treating it as an error if the other end closes the link after sending a response.

/FINISHINGS_DEFAULT=*keyword*
Specifies finishing operations to be performed on the printed documents. May or may not be supported by a given IPP server. Any or all of the four available finishings may be specified. Case is ignored. Keywords are:

- NONE
- STAPLE
- PUNCH
- COVER
- BIND

/[NO]FLAG_DEFAULT
Specifies whether a “job-sheets” attribute will be specified for jobs by default. If /FLAG_DEFAULT is used, job-sheets will be requested as “standard”. If /NOFLAG_DEFAULT is used, job-sheets will be requested as “none”.

/INCLUDE=*filename*
Specifies a sequential access text file containing additional qualifiers from this list. These qualifiers are read and processed at the point where the /INCLUDE qualifier is encountered, and share the precedence of that point.

/JOB_PRIORITY_DEFAULT=*integer*
Specifies the priority of the print job. 1 is the lowest, 100 is the highest.

/JOB_RETRY_DELAY=*deltatime*
Specifies, as an OpenVMS delta time specification, the length of time to hold a job when it is re-queued due to a temporary problem. The default value is "0 00:10:00.00" (10 minutes).

/LOG_FILE=*filename*
Specifies the name of the queue log file to write messages to for this queue. The default is in TCPware's default spool directory, unless overridden by a global setting, as described in *Global Settings*. The default filename is `IPP_SYMBIONT_Process_PID.LOG`.

/LOG_LEVEL=*logging_level*
Specifies one of the logging levels values listed in *Logging Levels* and is used to determine the severity of a message before it is written to the queue log file. Only those messages marked as this level, or a more serious one, are logged. The default is `JOB_TRACE` unless overridden by a global `TCPWARE_IPP_OPCOM_LEVEL` logical.

/MAX_LOG_BYTES=*number*
Specifies how many bytes of data will be logged by the send and receive routines when running with logging level set to `DETAILED_TRACE`. The value is an integer. A negative value sets the limit to infinite (all data will be logged). A value of zero turns off inclusion of data to the log file. A positive value sets the actual number of bytes logged, and any additional data is ignored. The default action is to log all data.

/MEDIA_DEFAULT=*name*
This attribute identifies the medium that the printer uses for all pages of the job.

The values for "media" include medium-names, medium-sizes, input-trays and electronic forms. See your printer documentation for details concerning what values are supported for your printer.

Standard keyword values are taken from ISO DPA and the Printer MIB and are listed in Section 14 of RFC 2566. Some servers may support definition of locally created names as well. See the *Input Tray Name*s table below for standard values for input trays. The below table contains examples of standard names. These names include, but are not limited to the following:

Name	Description
`default`	The default medium for the output device
`iso-a4-white`	Specifies the ISO A4 white medium

`iso-a4-colored`	Specifies the ISO A4 colored medium
`iso-a4-transparent`	Specifies the ISO A4 transparent medium
`na-letter-white`	Specifies the North American letter white medium
`na-letter-colored`	Specifies the North American letter colored medium
`na-letter-transparent`	Specifies the North American letter transparent medium
`na-legal-white`	Specifies the North American legal white medium
`na-legal-colored`	Specifies the North American legal colored medium
`na-9x12-envelope`	Specifies the North American 9x12 envelope medium
`monarch-envelope`	Specifies the Monarch envelope
`na-number-10-envelope`	Specifies the North American number 10 business envelope medium
`na-7x9-envelope`	Specifies the North American 7x9 inch envelope
`na-9x11-envelope`	Specifies the North American 9x11 inch envelope
`na-10x14-envelope`	Specifies the North American 10x14 inch envelope
`na-number-9-envelope`	Specifies the North American number 9 business envelope
`na-6x9-envelope`	Specifies the North American 6x9 inch envelope
`na-10x15-envelope`	Specifies the North American 10x15 inch envelope
`executive-white`	Specifies the white executive medium
`folio-white`	Specifies the folio white medium
`invoice-white`	Specifies the white invoice medium

`ledger-white`	Specifies the white ledger medium
`quarto-white`	Specified the white quarto medium
`iso-a0-white`	Specifies the ISO A0 white medium
`iso-a1-white`	Specifies the ISO A1 white medium
`a`	Specifies the engineering A size medium
`b`	Specifies the engineering B size medium
`c`	Specifies the engineering C size medium
`d`	Specifies the engineering D size medium
`e`	Specifies the engineering E size medium

The following standard values are defined for input-trays:

Name	**Description**
`top`	The top input tray in the printer.
`middle`	The middle input tray in the printer.
`bottom`	The bottom input tray in the printer.
`envelope`	The envelope input tray in the printer.
`manual`	The manual feed input tray in the printer.
`large-capacity`	The large capacity input tray in the printer.
`main`	The main input tray

side	The side input tray

/MULTIPLE_DOCUMENT_HANDLING_DEFAULT=*keyword*

This qualifier is relevant only for jobs consisting of two or more documents, and when the IPP server supports jobs with multiple documents. The qualifier controls finishing operations and the placement of one or more pages onto media sheets. When printing multiple copies, it also controls the order in which the copies that result are produced. Standard keyword values are listed in the below table.

single-document	If a job has multiple documents, say, the documents are called A and B, then the result printing the data (A and then B) will be treated as a single sequence of media sheets for finishing operations; that is, finishing would be performed on the concatenation of the two documents. The printer will not force the data in each document to start on a new page. If more than one copy is requested, the ordering of the pages resulting from printing will be A, B, A, B, ..., and the printer will force each copy (A, B) to start on a new media sheet.
separate-documents-uncollated-copies	If a job has multiple documents, say, the documents are called A and B, then the result of printing each document will be treated as a single sequence of media sheets for finishing operations; that is, the documents A and B would each be finished separately. The printer will force each copy of the data in a single document to start on a new sheet. If more than one copy is made, the ordering of the pages will be A, A, ..., B, B ... .

`separate-documents-collated-copies`	If a job has multiple documents, say, A and B, then the result will be that each document will be treated as a single sequence of media sheets for finishing operations; that is, A and B would each be finished separately. The printer will force each copy of the result of processing the data in a single document to start on a new sheet. If more than one copy is made, the ordering of the documents will be A, B, A, B,... .
`single-document-new-sheet`	Same as `single-document`, except that the printer will ensure that the first page of each document in the job is placed on a new media sheet.

The `single-document` value is the same as `separate-documents-collated-copies` with respect to ordering of print-stream pages, but not media sheet generation, since `single-document` will put the first page of the next document on the back side of a sheet if an odd number of pages have been produced so far for the job, while `separate-documents-collated-copies` always forces the next document or document copy on to a new sheet. In addition, if the `finishings` attribute specifies `staple`, then with `single-document`, documents A and B are stapled together as a single document with no regard to new sheets, with `single-document-new-sheet`, documents A and B are stapled together as a single document, but document B starts on a new sheet, but with `separate-documents-uncollated-copies` and `separate-documents-collated-copies`, documents A and B are stapled separately.

> **Note:** None of these values provide means to produce uncollated sheets within a document, i.e., where multiple copies of sheet n are produced before sheet *n*+1 of the same document.

/NUMBER_UP_DEFAULT=*number*
Specifies the number of page images to be placed on each side of each sheet of paper. The number must be an integer that is acceptable to the IPP server. If the number specified is not a value supported by the server, the job aborts.

/OPCOM_LEVEL=*logging_level*
Specifies one of the logging levels value listed in *Logging Levels*, and is used to determine the severity of a message before it is sent to OPCOM. Only those messages marked as this level, or at a more serious level, are sent. The default is `INFO` unless overridden by a global `TCPWARE_IPP_OPCOM_LEVEL` logical.

/OPCOM_TERMINAL=*opcom_term*
Specifies which OPCOM operator terminal to send OPCOM messages to. The default is the `PRINT` operator. See the OpenVMS documentation for the `REPLY/ENABLE` command for more information on OPCOM terminals.

/ORIENTATION_DEFAULT=*keyword*
Specifies the default page orientation. Case is ignored. Supported values are:

- `PORTRAIT`
- `REVERSE_PORTRAIT`
- `LANDSCAPE`
- `REVERSE_LANDSCAPE`

/PAGE_RANGE_DEFAULT="*range[,range]...*"
Specifies the page numbers to print. *`range`* is either a single integer page number, or a pair of page numbers, separated by a hyphen. Multiple range specifications are separated by commas. For example:

```
$ PRINT/QUEUE=IPP_QUEUE/PARAM=(PAGE_RANGES="1,3-6,9,12-14") TEST.TXT
```

The example specifies the pages: 1, 3, 4, 5, 6, 9, 12, 13, and 14. Note that embedded spaces are allowed, and ignored.

/QUALITY_DEFAULT=*keyword*
Specifies the quality of the printed material. Case is ignored. The keywords are:

- DRAFT
- NORMAL
- HIGH

/SIDES_DEFAULT=*keyword*
Specifies how the printing is to be placed on the paper.

- ONE-SIDED: prints each consecutive page upon one side of consecutive media sheets.
- TWO-SIDED-LONG-EDGE: prints each consecutive pair of pages upon the front and back sides of consecutive media sheets, with the orientation of each pair of pages on the long edge. This positioning is called duplex or head-to-head also.
- TWO-SIDED-SHORT-EDGE: prints each consecutive pair of pages upon front and back sides of consecutive media sheets, with the orientation of each pair of print-stream pages on the short edge. This positioning is called tumble or head-to-toe also.

/SPOOL_DIRECTORY=*dirspec*
Specifies the directory to use for storing temporary files used while processing print jobs for the queue. The default is TCPware's default spool directory.

Order of Processing

The various logicals and qualifiers described in the previous two sections sometimes define the same configuration item. The operation has been defined, but the precedence has not. The order, from lowest precedence to highest, is:

1. Built-in hard coded default values.
2. Global logicals, as described in the first section.
3. Queue-specific qualifiers found in the /DESCRIPTION string of the queue.
4. Queue-specific qualifiers found in the queue-specific CONFIG logical.

The queue-specific qualifiers are parsed second, allowing for an override of the global settings on a queue-by-queue basis when that behavior is desired.

PRINT Command Options

Print command options are specified using the OpenVMS standard `/PARAMETERS` qualifier. The list of options is enclosed in parenthesis. For example,

```
$ PRINT /QUEUE=IPP_PRINTER_1 /PARAMETER=(COPIES=3,
ORIENTATION=LANDSCAPE) FILE.TXT
```

These options are not case sensitive. The underscores in the option names are optional. Each may be abbreviated as long as the result is not ambiguous.

The available print command options are:

PRINTER=*printer_uri*
Specifies the target printer when the queue default is not desired, or when there is no queue default. The printer URI specified must match at least one of the defined printer URI's for the print queue.

Wildcards cannot be used in the printer URI.

COPIES=*number*
Specifies the number of copies of each document to print. The default value is 1.

SIDES=*keyword*
Specifies how the printing is to be placed on the paper. The *keyword* must be one of the following:

- `ONE-SIDED` or `1sided`: prints each consecutive page upon one side of consecutive media sheets.
- `TWO-SIDED-LONG-EDGE` or `two-long-edge` or `2long_side`: prints each consecutive pair of pages upon the front and back sides of consecutive media sheets, with the orientation of each pair of pages on the long edge. This positioning is called duplex or head-to-head also.
- `TWO-SIDED-SHORT-EDGE` or `two-short-edge` or `2short_side`: prints each consecutive pair of pages upon front and back sides of consecutive media sheets, with the orientation of each pair of print-stream pages on the short edge. This positioning is called tumble or head-to-toe also.

ORIENTATION=*keyword*
Specifies the page orientation. The *keyword* must be one of:

- `PORTRAIT`
- `REVERSE_PORTRAIT`

- LANDSCAPE
- REVERSE_LANDSCAPE

These can be abbreviated to any non-ambiguous prefix. Case is ignored.

[NO]FLAG
Requests, or suppresses, the printing of an IPP flag page for the job. The printer may, or may not, respond to this request. The exact format of this flag page is up to the IPP Server (printer) implementation.

NUMBER_UP=*number*
Specifies the number of page images to be placed on each side of each sheet of paper. The number must be an integer that is acceptable to the IPP server. If the number specified is not a value supported by the server, the job aborts.

DOCUMENT_FORMAT=*MIME-media-type*
DOCUMENT_FORMAT=*printer_default***
Specifies the document format of the files in the job, or specifies use of the printer's built-in default. The default for this qualifier is the default for the queue. Also, if the queue configuration does not specify a default document format, the hard coded default is `text/plain`.

JOB_PRIORITY=*integer*
Specifies the priority of the print job at the IPP server (not to be confused with the OpenVMS queue priority). 1 is the lowest, 100 is the highest.

FINISHINGS="*keyword[,keyword]...*"
Specifies finishing operations to be performed on the printed documents. May or may not be supported by a given IPP server. Any or all of the four available finishings may be specified. Case is ignored.

- BIND
- COVER
- PUNCH
- STAPLE

MULTIPLE_DOCUMENT_HANDLING=*keyword*

Specifies how you want the printer to print your job. The *keyword* is one of the following:

- Single_Document or 1Document
- Separate_Documents_Uncollated_Copies or UncollatedSeparate
- Separate_Documents_Collated_Copies or CollatedSeparate
- Single_Document_New_Sheet or NewSheet

Case is ignored. See /MULTIPLE_DOCUMENT_HANDLING_DEFAULT in this chapter for information on single document, separate-documents-uncollated-copies, separate-documents-collated-copies, and single-document-new-sheet handling.

PAGE_RANGES="*range[,range]...*"
Specifies the page numbers to print. *range* is either a single integer page number, or a pair of page numbers, separated by a hyphen. Multiple range specifications are separated by commas and enclosed in double quotes.

For example:

```
$ PRINT/QUEUE=IPP_QUEUE/PARAM=(PAGE_RANGES="1,3-6,9,12-14") FILE.TXT
```

Note that embedded spaces are allowed, and ignored. The example specifies the pages: 1, 3, 4, 5, 6, 9, 12, 13, and 14.

MEDIA=*name*
This attribute identifies the medium that the printer uses for all pages of the Job.

The values for *name* include medium-names, medium-sizes, input-trays and electronic forms. See your printer documentation for details concerning what values are supported for your printer.

Standard keyword values are taken from ISO DPA and the Printer MIB and are listed in section 14 of RFC 2566. Some servers may support definition of locally created names as well.

See *Standard Media Names* and *Input Tray Names* for the standard media names.

QUALITY=*keyword*
Specifies the quality of the printed material. Case is ignored. The keyword choices are:

- DRAFT
- HIGH
- NORMAL

Using Logicals to Define Queue Configurations

This section provides examples of using logicals to define queue configuration prior to queue initialization. This method can be used both as an alternative to and in addition to the `/DESCRIPTION` string shown in the previous examples. See the *Configuration* section for a complete description of all available qualifiers.

Setting Up IPP Symbiont Queues

Creating an IPP symbiont queue is done using the OpenVMS `INITIALIZE/QUEUE` command. All standard qualifiers are allowed, but the `/DESCRIPTION` qualifier has special use with the IPP symbiont. See the *Configuration* section.

Setting up IPP Symbiont Queues Using Queue-Specific Logicals

Set up an IPP symbiont queue named `ENG_PRINTER` to obtain its configuration information from a queue specific configuration file and to print a flag page with each job.

```
$ DEFINE/SYSTEM TCPWARE_IPP_ENG_PRINTER_CONFIG -
_$ "/INCLUDE=SYS$SYSTEM:ENG_PRINTER.SETUP/FLAG_DEFAULT"
$ INITIALIZE/QUEUE/PROCESSOR=TCPWARE_IPP_SYMB ENG_PRINTER
```

The file `SYS$SYSTEM:ENG_PRINTER.SETUP` contains:

```
/printer="ipp://engprinter.mynet.com:631/ipp"
```

Setting Up an IPP Symbiont Queue to Print Only to a Specific Printer

Set up the IPP symbiont queues named `IPP_PRINT_QUEUE` and `IPP_PRINT_2` to print only to the iprinter.example.com printer on port 631. Additionally, `IPP_PRINT_2` will always print two copies of each submitted file if copies are supported by the printer.

```
$ DEFINE/SYSTEM TCPWARE_IPP_CONFIG -
_$ "/PRINTER_URI=""ipp://iprinter.example.com:631/ipp"""
$ INITIALIZE/QUEUE /PROCESSOR=TCPWARE_IPP_SYMB IPP_PRINT_QUEUE
$ INITIALIZE/QUEUE /PROCESSOR=TCPWARE_IPP_SYMB -
_$ /DESCRIPTION="/copies_default=2" IPP_PRINT_2
```

Setting Up to Print to Multiple Printers Using Wildcards

Set up an IPP symbiont queue to print to any IPP printer in the exmaple.com domain, with the default printer being iprinter.example.com:

```
$ INITIALIZE/QUEUE /PROCESSOR=TCPWARE_IPP_SYMB -
```

```
_$ /DESCRIPTION="/printer=
""http://iprinter.example.com:631/ipp|*.example.com"""
IPP_PRINT_QUEUE
```

Setting Up Two Queues Using a Disk File for Queue Settings

Set up two IPP symbiont queues to print to any printer in the example.com domain, with the default printer being iprinter.example.com for one queue, and oprinter.example.com for the other. Log all possible messages to the log file, but send only messages more severe than `FILE_TRACE` to OPCOM. Use a 5 minute retry delay, and make the document format default the same as the printer's default. Use a disk file for the configuration information common to both queues:

```
$ INITIALIZE/QUEUE /PROCESSOR=TCPWARE_IPP_SYMB -
_$ /DESCRIPTION="/printer=
""http://iprinter.example.com:631/ipp|*.example.com""
/include=SYS$SYSTEM:IPP_QUEUE.SETUP" IPRINTER_QUEUE
$ INITIALIZE/QUEUE /PROCESSOR=TCPWARE_IPP_SYMB -
_$ /DESCRIPTION="/printer=
""http://oprinter.example.com:631/ipp|*.example.com""
/include=SYS$SYSTEM:IPP_QUEUE.SETUP" OPRINTER_QUEUE
```

The file `SYS$SYSTEM:IPP_QUEUE.SETUP` contains:

```
/log_level=DETAILED_TRACE
/opcom_level=FILE_TRACE
/job_retry_delay="0 00:05:00.00"
/default_document_format=***printer_default***
```

Setting Up Two Queues with no Configuration Values in the INITIALIZE Command

Do the same as the prior example, but put as much of the configurations in disk files as possible to allow changes to queue characteristics without having to re-initialize the queues:

```
$ INITIALIZE/QUEUE /PROCESSOR=TCPWARE_IPP_SYMB -
_$ /DESCRIPTION="/INCLUDE=SYS$SYSTEM:IPP_IPRINTER.SETUP"
IPRINTER_QUEUE
$ INITIALIZE/QUEUE /PROCESSOR=TCPWARE_IPP_SYMB -
_$ /DESCRIPTION="/INCLUDE=SYS$SYSTEM:IPP_OPRINTER.SETUP"
OPRINTER_QUEUE
```

The file `SYS$SYSTEM:IPP_IPRINTER.SETUP` contains:

```
/printer="http://iprinter.example.com:631/ipp|*.example.com"
/include=SYS$SYSTEM:IPP_QUEUE.SETUP
```

The file `SYS$SYSTEM:IPP_OPRINTER.SETUP` contains:

```
/printer="http://oprinter.example.com:631/ipp|*.example.com"
/include=SYS$SYSTEM:IPP_QUEUE.SETUP
```

The file `SYS$SYSTEM:IPP_QUEUE.SETUP` contains:

```
/log_level=DETAILED_TRACE
/opcom_level=FILE_TRACE
/job_retry_delay="0 00:05:00.00"
/default_document_format=***printer_default***
```

Submitting Jobs to IPP Symbiont Print Queues

This section describes how to submit jobs to the IPP symbiont print queues.

Printing a Single Text File to an IPP Queue

Print the file `FOO.TXT` to the `IPRINTER` (default destination printer) set up in the prior examples:

```
$ PRINT/QUEUE=IPRINTER_QUEUE foo.txt
```

Specifying the Destination Printer on the Print Command

Print a single text file to a non-default printer on a queue with a wild carded printer URL:

```
$ PRINT /QUEUE=iprinter_queue -
_$ /PARAM=(printer="ipp://another.example.com/ipp/port1") foo.txt
```

> **Note:** The above will fail unless the queue specifies another.example.com as a legal URL, either explicitly or by using wildcards.

Using Other Print Qualifiers

Print a text file to a default printer on a queue but specify the document format and additional copies:

```
$ PRINT /QUEUE=iprinter_queue -
_$ /PARAM=(document="text/plain" copies=3) foo.txt
```

IPP SHOW Command

The `IPP SHOW` utility allows a user to learn the capabilities supported by an IPP server. This utility queries the server and displays the supported attributes. The program can be used to see what a given server supports, by a program to gather information about a number of printers, or by a DCL or other program to check the capabilities of a given server before submitting a print job to a queue. The command syntax is:

```
$ TCPWARE IPP SHOW server_URI /qualifiers...
```

Qualifiers

/ATTRIBUTE=*attribute*

Puts the program into a mode suitable for use from a DCL command procedure. Not compatible with the `/FORMAT` or `/OUTPUT` qualifiers or those associated with them. It causes the program to return the value of a single attribute as a character string in a DCL symbol. This is for a procedure to check to see if, for example, a given server supports color printing before submitting a job to a queue that requires color output. Allowable values for *attribute* are:

```
Charset_Configured                Orientation_Requested_Default
Charset_Supported                 Orientation_Requested_Supported
Color_Supported                   Page_Ranges_Default
Compression_Supported             Page_Ranges_Supported
Copies_Default                    PDL_Override_Supported
Copies_Supported                  Print_Quality_Default
Document_Format_Default           Print_Quality_Supported
Document_Format_Supported         Printer_Current_Time
Finishings_Default                Printer_Driver_Installer
Finishings_Supported              Printer_Info
Gen_Natural_Language_Supported    Printer_Is_Accepting_Jobs
Job_Hold_Until_Default            Printer_Location
Job_Hold_Until_Supported          Printer_Make_and_Model
Job_Impressions_Supported         Printer_Message_From_Operator
Job_K_Octets_Supported            Printer_More_Info
Job_Media_Sheets_Supported        Printer_More_Info_Manufacturer
Job_Priority_Default              Printer_Name
Job_Priority_Supported            Printer_Resolution_Default
Job_Sheets_Default                Printer_Resolution_Supported
Job_Sheets_Supported              Printer_State
Media_Default                     Printer_State_Message
Media_Supported                   Printer_State_Reasons
Multiple_Doc_Handling_Default     Printer_Uptime
Multiple_Doc_Handling_Supported   Printer_URI_Supported
Multiple_Operation_Timeout        Queued_Job_Count
Natural_Language_Configured       Reference_URI_Schemes_Supported
```

```
Number_Up_Default                Sides_Default
Number_Up_Supported              Sides_Supported
Operations_Supported             URI_Security_Supported
```

/[NO]APPEND
Specifies that output should be appended to an existing output file if possible. /NOAPPEND is the default.

/FORMAT=*style*
Specifies what print style to use. *style* is either

- SCREEN (default) which writes in a human-friendly screen-formatted mode, or
- LIST which writes an easy to parse, *name=value* format, one name/value pair per line.

/[NO]FULL
Causes all IPP attributes to be included in the display, whether the server supports them or not. Those not supported are marked as such. /NOFULL is the default.

/[NO]GLOBAL
Specifies whether the named symbol should be created as a DCL global symbol. Used only with /ATTRIBUTE. If specified as /NOGLOBAL, the symbol will be local to the calling procedure level. /GLOBAL is the default.

/OUTPUT=*file*
Specifies a file to write output to. SYS$OUTPUT: is the default.

/SYMBOL=*symbolname*
Specifies a DCL symbol name that should be set to the value of the specified attribute. Used only with /ATTRIBUTE. The default is IPP SHOW_RESULT if /SYMBOL is not specified.

EXAMPLES of IPP SHOW Use and Output

1. Basic operation with all defaults:

```
$ TCPWARE IPP SHOW LILLIES.EXAMPLE.COM
LILLIES.EXAMPLE.COM as of Tue Aug  10 16:08:43 2020

CURRENT INFO:
    Printer State:       Idle
    State Reasons:       none
    Accepting Jobs?:     Yes
    Queued Job Count:    0

PRINTER INFO:
    Name:                Lexmark Optra T610
    Make & Model:        Lexmark Optra T610


DEFAULTS:
    Document Format:     application/octet-stream
    Orientation:         Portrait
    Number-Up:           1
    Copies:              1
    Job Media Sheets:    none
    Character Set:       utf-8
    Natural Language:    en-us

SUPPORTED FEATURES AND ALLOWED VALUES:
    Color?:              No
    Orientation:         Portrait, Landscape
    Document Formats:    application/octet-stream, application/postscript,
                         application/vnd.hp-PCL, text/plain
    Job Sheets:          none, standard
    Number-Up:           1:16
    Copies:              1:999
    PDL Override:        not-attempted
    Character Sets:      utf-8, us-ascii
    Natural Languages:   en-us
    Operations:          Print_Job, Validate-Job, Cancel-Job,
                         Get-Job_Attributes, Get-Jobs,
                         Get-Printer_Atrributes, Unknown: 18

URIs Supported and associated security options:
        URI:      http://192.168.50.2/
        Security: none

        URI:      http://192.168.50.2:631/
       Security: none
```

2. Operation with `/FULL` and output to a file (note that the "`/`" character in the URI requires use of quotes around the server URI parameter):

```
$ TCPWARE IPP SHOW "LILLIES.EXAMPLE.COM/IPP" /FULL /OUTPUT=FOO.BAR
```

```
FOO.BAR contains:
LILLIES.EXAMPLE.COM/IPP as of Tue Aug 10 16:11:54 2020
CURRENT INFO:
    Printer State:       Idle
    State Reasons:       none
    State Message:       <not supported>
    Accepting Jobs?:     Yes
    Queued Job Count:    <not supported>
    Uptime (seconds):    <not supported>
    Printer Time:        <not supported>

 PRINTER INFO:
    Name:                LILLIES
    Printer Location:    <not supported>
    Printer Info:        MANUFACTURER:Hewlett-Packard;COMMAND SET:PJL,ML -
                           C,PCL,PCLXL,POSTSCRIPT;MODEL:HP LaserJet 2100  -
                           Series;CLASS:PRINTER;DESCRIPTION:H
    URL for more info:   <not supported>
    URL for driver:      <not supported>
    Make & Model:        <not supported>
    URL for Maker:       <not supported>

DEFAULTS:
    Document Format:     application/octet-stream
    Orientation:         <not supported>
    Number-Up:           <not supported>
    Sides:               <not supported>
    Copies:              <not supported>
    Mult. Doc. Handling: <not supported>
    Media:               <not supported>
    Job Media Sheets:    <not supported>
    Finishings:          <not supported>
    Job Priority:        <not supported>
    Job Hold Until:      <not supported>
    Print Quality:       <not supported>
    Printer Resolution:  <not supported>
    Character Set:       us-ascii
    Natural Language:    en-us
    Mult. Op. Timout:    <not supported>

SUPPORTED FEATURES AND ALLOWED VALUES:
    Color?:              <not supported>
    Orientation:         <not supported>
    Document Formats:    text/plain, text/plain; charset=US-ASCII,
                           application/postscript, application/vnd.hp-PCL,
                           application/octet-stream
    Job Sheets:          <not supported>
    Number-Up:           <not supported>
    Sides:               <not supported>
    Copies:              <not supported>
    Mult. Doc. Handling: <not supported>
    Media Names:         <not supported>
```

```
    Job Media Sheets:    <not supported>
    Finishings:          <not supported>
    Job Priority:        <not supported>
    Job Hold Until:      <not supported>
    Page Ranges?:        <not supported>
    Print Qualities:     <not supported>
    Resolutions:         <not supported>
    Compression Modes:   <not supported>
    Job K-octets:        <not supported>
    Job Impressions:     <not supported>
    PDL Override:        not-attempted
    Character Sets:      us-ascii, utf-8
    Natural Languages:   en-us
    URI Schemes:         <not supported>
    Operations:          Print_Job, Validate-Job, Cancel-Job,
                           Get-Job_Attributes, Get-Jobs,
                           Get-Printer_Atrributes

URIs Supported and associated security options:

        URI:      /ipp
        Security: none

        URI:      /ipp/port1
        Security: none

MESSAGE FROM OPERATOR:

<no Message>
```

3. Operation with `/ATTRIBUTE` and `/SYMBOL` and `/GLOBAL` to get a single attribute into a DCL symbol:

```
$ TCPWARE IPP SHOW LEXIM /ATTRIB=NUMBER_UP_SUPPORTED /SYMBOL=NUMUP
/GLOBAL
$ SHO SYM NUMUP
  NUMUP == "1:16"
$
```

15. Managing R Commands

Introduction

This chapter describes how to:

- Manage and troubleshoot the R commands services, and set up host equivalence files on the server.
- Manage the R Shell (RSH) server, which handles Remote Copy Program (RCP) requests.
- Manage the Remote Magnetic Tape (RMT) Server.

R Services

Configure the Berkeley R Services as part of TCPware configuration (CNFNET). The following are the specific steps in the process:

1. Specify whether you want to enable the login, shell, and exec services. Specify `YES` or `NO` in each case.
2. For login service, also specify whether you want `NORMAL` or `SECURE` login authorization (the default is `SECURE`).
3. If desired, create a service access list.
4. Set up a host equivalence file.

Service Access Lists

You may want to set up a service access list to control which hosts, group of hosts, or network can access the service.

Use the `ADD ACCESS_LIST` command in TCPware's Network Control Utility (`NETCU`) that lets you specify a list number and `PERMIT` or `DENY` condition for a specific network internet address (and optional network mask). Here is the command format:

```
$ NETCU ADD ACCESS_LIST list condition ia mask
```

For details on the `ADD ACCESS_LIST` command, see the *NETCU Command Reference*.

Host Equivalence Files

Host equivalence files are security access files on the server host used to authorize access to services by other hosts or users. The files list hostnames (and, optionally, usernames) and indicate which remote hosts and users have equivalent access as local users.

(Host equivalence files authorize access to login and shell services only, since the exec service relies on direct username and password authorization.)

Two types of equivalence files are available: `HOSTS.EQUIV` and `.RHOSTS`. These files serve slightly different needs and are in different locations on the OpenVMS host, although they use the same data format:

- The `HOSTS.EQUIV` file defines which remote hosts or users can have equivalent access to the server host, and is analogous to the `/etc/hosts.equiv` file in UNIX. Place the `HOSTS.EQUIV` file in either the `TCPWARE_COMMON:[TCPWARE]` or `TCPWARE_SPECIFIC:[TCPWARE]` directory, depending on your configuration.
- The `.RHOSTS` file lets remote users access local accounts beyond what the `HOSTS.EQUIV` file specifies, and is analogous to the UNIX `~/.rhosts` file. Place the `.RHOSTS` file in the local account's login directory. Here are some things to keep in mind:
    - A remote user specified in a `.RHOSTS` file can access the local account only if the account owns the file.
    - Access to the SYSTEM (root) account, or one that has a system UIC group, requires a `.RHOSTS` file, and does not work in the `HOSTS.EQUIV` file.
    - To disable user-specified `SYS$LOGIN:.RHOSTS` files (and use the `HOSTS.EQUIV` file only), set the `TCPWARE_RCMD_FLAGS` system logical to `1` (it is `0` by default).

Both the `HOSTS.EQUIV` and `.RHOSTS` files contain line entries for hostnames and optional usernames, in the following format, where *`host`* is the name of the remote host allowed access and the optional *`username`* is the name of a specific user on that host:

```
host [username]
```

Here are some sample lines in a host equivalence file, with the following considerations:

```
ALPHA
BETA smith
GAMMA Jones
DELTA +
+ JackSprat
```

- If you specify only *`host`* on a line (omitting *`username`*), users having accounts with the same names on both hosts can access the local system.

- If you specify both *host* and *username* on a line, the specified user on that host can access the local system or account. The *username* must match case exactly with the incoming username (`JackSprat` does not match with `jacksprat`).
- You can use the plus symbol (+) as a wildcard for *host* and *username*, but do not use an asterisk (*) since UNIX does not recognize it as a wildcard. A wildcarded *host* entry grants access to any host, provided username checks pass. A wildcarded *username* entry grants access to any user, provided the hostname passes.
- The `HOSTS.EQUIV` and `.RHOSTS` files may handle some entries differently, as described in the below table.

Sample entry...	**In the `HOSTS.EQUIV` file...**	**In the `.RHOSTS` file (with account owner Betty)...**
`ALPHA`	Accepts anyone from ALPHA for access to a local account with the same name as the remote username	Accepts only username Betty from ALPHA
`ALPHA Betty`	Accepts only username Betty from ALPHA for any account other than SYSTEM	Accepts only username Betty from ALPHA
`ALPHA David`	Accepts only username David from ALPHA for any account other than SYSTEM	Accepts only username David from ALPHA who requests username Betty (such as with `rlogin /user="Betty"`)
`ALPHA +`	Accepts anyone from ALPHA for any requested account	Accepts anyone from ALPHA who requests username Betty
`+ Betty`	Accepts only username Betty from any host for any account other than SYSTEM	Accepts only username Betty from any host
`+ David`	Accepts only username David from any host for any account other than SYSTEM	Accepts only username David from any host who requests username Betty

+ +	Accepts anyone from any host for any requested account other than SYSTEM	Accepts anyone from any host who requests username Betty

Caution! Use *username* values or wildcards in the HOSTS.EQUIV file with security in mind. Because HOSTS.EQUIV files apply system-wide, any remote user (or group of users) can masquerade as a local user. Especially avoid using double pluses (+ +) in either file.

Host equivalence files grant access authorization differently depending on how you configure each R service during the regular TCPware configuration:

- For login service, CNFNET offers NORMAL and SECURE options for login authorization:

When...	With NORMAL login...	With SECURE login...
Hostname and username appear in the .RHOSTS or HOSTS.EQUIV file	User is logged in	User is prompted for password
Hostname or username does not appear in the .RHOSTS or HOSTS.EQUIV file	User is prompted for username and password (standard login sequence)	Access is denied

- The shell service checks the host equivalence files for command execution. If the check fails, shell denies access.
- The exec service performs a standard username and password check before allowing command execution on the host.

  The exec service also tracks break-in attempts using the OpenVMS Intrusion database. If a remote user enters an invalid login, TCPware creates an intrusion record based on the remote source's IP address. If TCPware reaches the threshold number of invalid login attempts, no one at the remote IP address can use the exec service. The intrusion database shows who the offending addresses are; you can re-enable login attempts by deleting the intrusion records.

See the SHOW INTRUSION and DELETE/INTRUSION_RECORD commands in HP's *VMS DCL Dictionary* for details.

Customizing the Shell and Exec Services

Incoming shell and exec services invoke the TCPWARE:TCPWARE_RSERVICE.COM procedure to perform the requested operation. You can customize this command procedure to map an incoming command to an OpenVMS command. (Just make sure that you do not destroy mapping for TCPware operations such as RCP and RMT.)

The REXEC and the RSHELL servers have the functionality to set up a DECwindows display to the client's IP address if the TCP/IP transport is loaded. To enable this functionality, define the logical as:

```
$ DEFINE/SYSTEM/EXEC TCPWARE_RCMD_ENABLE_DISPLAY "TRUE"
```

R Services Log File

You can set up a log file for incoming R Services such as RCP and RSH by defining the TCPWARE_RCMD_OUTPUT logical to log messages in the RCMD.LOG file, as follows:

```
$ DEFINE/SYSTEM/EXEC TCPWARE_RCMD_OUTPUT RCMD.LOG
```

RCP Server

Use the SHOW SERVICES command in NETCU to make sure that the R shell daemon (rshd) is operating before client users can use the RCP command. On TCPware hosts, the service should show up as shell under the Port column. If it does not, use CNFNET to enable the R Services.

You also need to use the rexec server if client users need to specify the RCP command's /USER, /PASSWORD, and /TRUNCATE qualifiers. On TCPware hosts, the service shows up as exec under the Port column.

> **CAUTION!** The /PASSWORD qualifier requires entry of a plain text password, which could cause a security problem. You can avoid having users specify the /USER or /PASSWORD qualifier by checking that remote hosts include your hostname entry in their host equivalence files (such as the /etc/hosts.equiv file in UNIX systems, or the TCPWARE:HOSTS.EQUIV or SYS$LOGIN:.RHOSTS file locally).

Make sure that your users' login files (as well as the `~/.login` files on UNIX system clients) do not contain commands that generate output.

You can log RCP activity by defining the `TCPWARE_RCMD_OUTPUT` logical. (See *R Services Log File*.)

Troubleshooting RCP

Here are some steps to follow to ensure that the RCP command given from a UNIX system works correctly with TCPware's R Services. First make sure that the R services were configured by running `@TCPWARE:CNFNET RCMD` to make sure that RCP is at the very least enabled and the image installed. Then follow these steps:

On the TCPware system:

1. Set the default to the login directory of the user account to which you will be copying files.
2. Edit the `SYS$LOGIN:.RHOSTS` file and add the UNIX hostname and username.
3. Edit the `TCPWARE:HOSTS.` file and add the IP address and hostname of the UNIX system from which you will be copying files.
4. Check the user account's UAF record (using `SET DEFAULT SYS$SYSTEM` followed by `MCR AUTHORIZE`).
5. Check that the owner of the login directory and `.RHOSTS` file is the same as the UAF record's `IDENTIFIER/UIC` (using `DIRECTORY/OWNER`). If different, you must set the ownership of the login directory and `.RHOSTS` file to match the `IDENTIFIER/UIC`.

On the UNIX system:

1. Issue the `rcp` command. If you get a `Login information not recognized at remote node` message, run `NETCU DEBUG /TCP /DATA=1500 /DIA=`*`unix-host-ip-address`*, preferably with the additional `/OUTPUT=RCPDEBUG.LOG`.
2. If there is a debug trace, look for the *`unix.username.vax-username.rcp filename.ext`*. If you notice that the case of the username is different from how it appears in the `.RHOSTS` file, change the case in the `.RHOSTS` file to match.

A `permission denied` message usually indicates a protection error on the UNIX side.

RMT Server

Here are some preliminary tasks to make sure the remote client can access the tape or CD-ROM drive:

1. Make sure the shell and exec services and the RMTSETUP image are installed during TCPware configuration at the Do you want to activate ... service: and Do you want to INSTALL the ... image: prompts, or make sure that at least RCMD_SERVICES == ":SHELL:EXEC" and RCMD_CLIENTS == ":RMTSETUP" appear in your TCPWARE:TCPWARE_CONFIGURE.COM file.
2. Make sure the magnetic tape or CD-ROM, recognizable to the system, is loaded in the device. With a tape device, the client essentially mounts and allocates the tape; you do not need to perform this task. With a CD-ROM device, you need to make the device accessible by issuing a MOUNT command.
3. Make sure the rsh command works from the UNIX system user's root directory to the OpenVMS user SYSTEM's directory.
4. For SYSTEM accounts, RMT ignores the TCPWARE:HOSTS.EQUIV file and uses an explicit entry in the SYS$LOGIN:.RHOSTS file to grant access.
5. Make sure nothing is output from SYS$SYLOGIN procedure or from SYS$LOGIN:LOGIN.COM when issuing the rsh command to SYSTEM.
6. Make sure you suppress output by including the following commands in the SYS$SYLOGIN and LOGIN.COM procedures:

```
$ RMT_VERIFY = 'F$VERIFY(0)

$ IF (F$MODE() .NES. "OTHER") THEN RMT_VERIFY = F$VERIFY(RMT_VERIFY)
```

RMT Client Utilities

On the remote host, a user can employ the rdump utility to dump files to OpenVMS tapes, or the rrestore utility to restore files from OpenVMS tapes. The functionality of rdump and rrestore depends entirely on the type of UNIX system you use and not on TCPware's RMT service. For example, not all UNIX systems let the user restore files selectively using rrestore.

When remote users use these remote dump and restore commands, they must specify either the OpenVMS device name or a filename. If they specify a device name, it must be a valid OpenVMS type of magnetic tape device name.

See your particular client system's documentation for details. Users should be careful about the order in which they specify options on the command line.

Here is an example of an rdump command:

```
>/etc/rdump 0f lilac:mua0:/nomount /usr
```

The remote user requests to remotely dump the /usr file system onto device mua0: on system lilac, and specifies the nomount qualifier and a tape density of 1600 bits per inch.

TCPware's RMT server lets you specify the qualifiers with magnetic tape device names indicated in the below table. The qualifiers are available with the `RMTSETUP` command.

Qualifier...	**Defines...**
`/ASSIST` `/NOASSIST`	Whether to use operator assistance to mount the volume. `/ASSIST` is the default.
`/BLOCKSIZE=`*`n`*	Default block size for magnetic tape volumes. The default is 65534 bytes.
`/CD`	Indicates that the remote device is a CD-ROM device.
`/COMMENT="`*`string`*`"`	Additional information included with the operator request when the mount operation requires operator assistance (`/ASSIST`). The comment appears in the OPCOM message for the operator.
`/DENSITY=`*`n`*	Density (in bits per inch) at which to write a foreign or unlabeled magnetic tape. The default is the current density.
`/MOUNT` `/NOMOUNT`	Whether to use the OpenVMS MOUNT service to mount the tape. `/NOMOUNT` gains access to the tape directly without mounting it. Use this for UNIX utilities that expect the tape drive to hold its position (not rewind) if the utility closes it. The default is `/MOUNT`. With `/NOMOUNT`, the qualifiers `/ASSIST`, `/BLOCKSIZE`, `/COMMENT`, and `/DENSITY` are not allowed. With `/NOMOUNT`, the defaults are `/NOUNLOAD`, `/NOREWIND`, `/STREAM`, and `/WRITE`.
`/REWIND` `/NOREWIND`	Whether to rewind the drive when it is closed. The default is `/REWIND`.
`/STREAM` `/NOSTREAM`	Whether to read the tape in record (`/NOSTREAM`) or byte-stream (`/STREAM`) mode. The default is `/STREAM`.

/UNLOAD /NOUNLOAD	Whether to unload the drive when it is closed. The default is /UNLOAD.
/WRITE /NOWRITE	Whether you can write to the volume. The default is /WRITE.

Client Examples

The following steps perform rdump and rrestore from a Linux client system. They dump two directories to the tape by issuing separate rdump commands. They then restore files selectively from the tape to the client system:

1. Put the directories on the tape by issuing two rdump commands:

```
>rdump 0f homer:mkb600/nomount /
>rdump 0f homer:mkb600/nomount/rewind /usr
```

Include the /nomount qualifier with the first rdump to keep OpenVMS from rewinding the tape (even though rdump on Linux reports to the contrary). The additional /rewind qualifier for the second rdump actually rewinds the tape.

2. Restore the files selectively from the tape using rrestore. In this example, rrestore extracts .rlogin from the second (2) dump file on the tape:

```
>rrestore fsx homer:mkb600/nomount/rewind 2 .rlogin
```

In this example, rrestore invokes the interactive utility to let the user specify particular files that were put on the tape in the first (1) dump file. The add command then adds the files to the extraction list and the extract command restores them:

```
>rrestore fis homer:mkb600/unload 1
restore>add /users
restore>extract
```

The rrestore command may display messages such as You have not read any volumes yet and ask you to specify the next volume. Although the messages may appear, rrestore should work properly.

16. Managing Mail Services

This chapter describes how to configure the TCPware SMTP (Simple Mail Transport Protocol) server to send and receive electronic mail.

The chapter also includes information about the Internet Message Access Protocol (IMAP) and the Post Office Protocol (POP).

If you are running PMDF or another mail system that provides its own SMTP support, refer to that mail system's documentation.

Modifying the TCPware SMTP Configuration File

TCPware's SMTP configuration is stored in the `START_SMTP.COM` and `START_SMTP_LOCAL.COM` startup command procedures. TCPware provides the `MAIL-CONFIG` utility for editing these files. You can invoke the utility with the `TCPWARE CONFIGURE /MAIL` command.

```
Configuring SMTP:

For detailed information on the following parameters, refer to the
TCPware for OpenVMS Management Guide.

Do you want to use the SMTP Mail Transfer Agent? [NO]: Y

One user on this system must act as the local postmaster. This person
will receive mail sent to the postmaster. The person's username must
always be valid while SMTP is operating.

Enter the username of the local postmaster [SYSTEM]: RETURN
Port for SMTP to use [25]? RETURN
Do you want to enable the SMTP RFC2789 MIB [No]? Y
Do you want to enable SMTP accounting [No]? Y
Name of the host that will run the accounting collection program:
BIGBOOTE.EXAMPLE.COM
Port number that accounting collection program listens on: 2222

For further configuration options, please see the procedure described in
the TCPware for OpenVMS Installation & Configuration Guide to configure
SMTP.
```

The last two questions are optional depending on the value of the one before them.

After using these configuration utilities, stop and restart the mail queues with `@TCPWARE:START_SMTP.COM` to update the OpenVMS cluster or with `@TCPWARE:START_SMTP_LOCAL.COM` to update the local host only.

If you have implemented virtual domains (that is, your system receives mail for multiple domain names), you must do the following:

- Create the file `TCPWARE:SMTP_HOST_ALIASES.` and specify, one per line, all the virtual domains for which this node serves as mail processor.
- Define the logical `TCPWARE_SMTP_ALLOW_VIRTUAL_DOMAIN` as a system-wide, executive-mode logical.

The file `TCPWARE:SMTP_LOCAL_LOGICALS.COM` can be created to define all local SMTP-related logicals. It executes each time SMTP starts. The available SMTP logicals are:

`TCPWARE_LOCALDOMAIN`	Specifies the default local domain name to be used when building To: addresses on outgoing messages. For example, to have messages sent to `SMTP%"Joe@bigboote"` to be delivered to `SMTP%"Joe@bigboote.example.com"`, `TCPWARE_LOCALDOMAIN` would be defined as `bedrock.com`.
`TCPWARE_NAMESERVERS`	List of IP addresses for DNS lookups.
`TCPWARE_SMTP_A1_NAME`	Used in forming the username portion of return addresses for ALL-IN-1 users.
`TCPWARE_SMTP_ACCEPT_UNIX_LF`	Tells the SMTP agents to accept lines sent by some UNIX systems that are terminated with a linefeed only (instead of the proper carriage-return, linefeed combination).
`TCPWARE_SMTP_ALLOW_USER_FROM`	Allows users to override their From: address on outgoing mail by specifying

	/FROM=*xxx*@*yyy* as the first line of outgoing mail messages.
TCPWARE_SMTP_ALLOW_VIRTUAL_DOMAIN	Allows the use of virtual domains in TCPware SMTP environment. Without this logical defined, incoming aliases are assumed to be local addresses only. If your system supports multiple virtual domains and uses in the alias file to reroute traffic based on those domains, you must define this logical.
TCPWARE_SMTP_AM_DOMAIN	Domain name used when forming return addresses for ALL-IN-1 users.
TCPWARE_SMTP_AM_NAME	Used in forming the username portion of return addresses for ALL-IN-1 users.
TCPWARE_SMTP_APPEND_FORWARDER_TO_MX	Specifies that the default SMTP forwarder, if defined, is appended to the end of an MX list for a target host when delivering outgoing mail.
TCPWARE_SMTP_BATCH_QUEUE	Points to the TCPware SMTP queue.
TCPWARE_SMTP_DECNET_DOMAIN	Specifies a DECnet name used in the creation of return addresses.
TCPWARE_SMTP_DELIVERY_RECEIPTS	Enables or disables delivery receipts (value is TRUE or FALSE).
TCPWARE_SMTP_DISABLE_DELIVERY_RECEIPT_DISCLAIMER	When deliver receipts are enabled, a disclaimer is included in all such receipts telling the sender that the message has been delivered, but not necessarily read. Defining this logical prevents the disclaimer from being included.
TCPWARE_SMTP_DISABLE_FOLDER_DELIVERY	Disables TCPware SMTP's ability to deliver messages to user-defined folders in their VMS Mail files.

TCPWARE_SMTP_DISABLE_OPTYPE_SANITY_CHECK	If this logical is defined then the SMTP symbiont will default to MAIL for the operation not specified in the files in the queue.
TCPWARE_SMTP_DISABLE_PSIMAIL	If defined, causes mail sent to PSI% users to be returned with NOSUCHUSER.
TCPWARE_SMTP_ENVELOPE_FROM_HOST	Specifies the host name to be used in the SMTP envelope MAIL FROM: line. If not defined, the default system host name is used.
TCPWARE_SMTP_FORWARDER	Specifies the domain name of the system to which all outgoing mail is forwarded for further delivery.
TCPWARE_SMTP_FROM_HOST	Specifies the local hostname used when forming From: address on outgoing messages. If this logical is not defined, the system host name is used.
TCPWARE_SMTP_HEADER_ORG	Specifies the text for an Organization: header in outgoing mail.
TCPWARE_SMTP_HEADER_RETURN_RECEIPT_TO	Generates a Return-Receipt-To: header in outgoing mail. Requires the TCPWARE_SMTP_RETURN_RECEIPT_TO_HEADER_ENABLE logical to be defined.
TCPWARE_SMTP_HEADER_SYS	Specifies the text for a System: header in outgoing mail.
TCPWARE_SMTP_HOST_ALIAS_FILE	Points to the file containing a list of all the host names that should be considered local for this node for incoming mail delivery.
TCPWARE_SMTP_HOST_NAME	Specifies all the local host names for this node. Used to specify all virtual domains handled by this node. Alternatively, the node names can be stored in the file TCPWARE:SMTP_HOST_ALIASES.

TCPWARE_SMTP_INCOMING_MSGSIZE_LIMIT	This logical can be defined to cause the SMTP server to reject messages that are larger than a desired size. The defined values are: S - 1 MB M - 10 MB L - 100 MB X - 1000 MB
TCPWARE_SMTP_LOG	Specifies the output filename. If not defined, the name defaults to TCPWARE:TCPWARE_SMTP_LOG.*queuename*.
TCPWARE_SMTP_MAXIMUM_822_TO_LENGTH	Sets the maximum length of the RFC822 To: header line when sending outgoing mail. The default is 1024. The range can be set from 256 to 65535.
TCPWARE_SMTP_MRGATE_NAME	Specifies the name of the Message Router gateway.
TCPWARE_SMTP_NON_LOCAL_ FORWARDER	Specifies the name of a forwarder system for non-local outgoing mail.
TCPWARE_SMTP_NO_USER_REPLY_TO	Disallows the use of user-defined Reply-To: headers in outgoing mail.
TCPWARE_SMTP_POSTMASTER	Specifies the address of the system-wide postmaster.
TCPWARE_SMTP_REJECT_INVALID_DOMAINS	Tells the SMTP server to reject mail from domains whose names and addresses cannot be resolved in a reverse lookup.
TCPWARE_SMTP_REPLY_TO	Specifies an address for a Reply-To: header in outgoing mail.

TCPWARE_SMTP_RESENT_HEADERS	Causes the inclusion of "Resent-*" headers in mail forwarded from a VMS Mail account using SET FORWARD in VMS Mail.
TCPWARE_SMTP_RETRY_INTERVAL	Specifies the retry interval for messages waiting for an attempted redelivery. The time is specified as a delta time.
TCPWARE_SMTP_RETURN_INTERVAL	Specifies the amount of time a given message delivery should be retried before giving up and bouncing the message back to the sender. The time is specified as a delta time.
TCPWARE_SMTP_RETURN_MSG	Specifies an input filename for the return message SMTP sends when a mail message bounces.
TCPWARE_SMTP_RETURN_RECEIPT_TO_HEADER_ENABLE	Enables the Return-Receipt-To: header if the TCPWARE_SMTP_HEADER_RETURN_RECEIPT_TO logical is also defined.
TCPWARE_SMTP_SEND_CLASS	Specifies the VMS broadcast class for "New mail" notifications. The default is USER16.
TCPWARE_SMTP_SERVER_DISABLE_VRFYEXPN	Disables the VRFY and EXPN commands in bitmask format to the SMTP server. Bit 0 = VRFY; Bit 1 = EXPN.
TCPWARE_SMTP_SERVER_LOG	Enables debug logs for the SMTP server.
TCPWARE_SMTP_SERVER_RCPT_CHECK_HOST	The host name to be used in checking for local host when passing messages through the reject rules.
TCPWARE_SMTP_SERVER_REJECT_FILE	Points to the file containing the rejection rules.
TCPWARE_SMTP_SERVER_REJECT_INFO	Specifies the level of OPCOM messages generated by the rejection rules for incoming SMTP mail. If not defined, no messages are generated.

TCPWARE_SMTP_SUPPRESS_VENDOR	Suppresses the vendor's name in the SMTP server welcome banner. Define this logical to hide the fact that the system is a VMS system running TCPware.
TCPWARE_SMTP_SYMBIONT_LOG	Enables debug logs for the SMTP symbiont.
TCPWARE_SMTP_SYMBIONT_PURGWS_TIMER	Specifies how often the SMTP symbiont purges its working set to free up unneeded memory. The time is specified as a delta time.
TCPWARE_SMTP_WINDOW_SIZE	Specifies the window size used in TCP connections when delivering mail.
TCPWARE_VMSMAIL_HEADER_CONTROL	Specifies how many RFC822 headers are included in mail delivered to VMS Mail users. Values can be ALL, MAJOR, and NONE.
TCPWARE_VMSMAIL_LOCASE_USERNAME	Lowercases the username portion of outgoing addresses.
TCPWARE_VMSMAIL_NO_EXQUOTA	Delivers incoming mail to local VMS Mail users without using EXQUOTA.
TCPWARE_VMSMAIL_REPLY_CONTROL	Specifies which header to use to determine the sender of a message ("Reply-To:" or "From:").
TCPWARE_VMSMAIL_USE_RFC822_TO_HEADER	Specifies that addresses in the RFC822 To: and CC: headers should make up the VMS Mail To: and CC: headers.

Delivering Mail to Specific Folders

The SMTP server supports mail delivery to folders other than the NEWMAIL folder. The folder names are restricted to UPPERCASE characters only, the pound sign (#), and the underscore (_). Use of the comma (,) in a folder name causes an error. Mail addressed to *user+folder@host* is delivered to the specified *folder*. You can disable this mechanism by defining the system-wide logical name TCPWARE_SMTP_DISABLE_FOLDER_DELIVERY.

Using the New Mail Delivery Mechanisms

The current release of SMTP supports alias file extensions that request mail delivery to a file or specify addresses in a separate file. You must use the SMTP aliases file, specified with `TCPWARE CONFIG/MAIL`, to list all of these new mail delivery mechanisms. The default is `TCPWARE:SMTP_ALIASES`. The syntax for these aliases follows the form of those described in *Configuring Mail Aliases* found later in this chapter. It is necessary to use the colon and semicolon in the command lines as shown in the examples.

<device:[directory]address.list	Delivers mail to the list of addresses in the specified file. `alias1 : "<filespec" ;`
| device:[directory]procedure.com parameter(s)	Submits the specified command procedure to the queue identified by the logical name `TCPWARE_SMTP_BATCH_QUEUE`, `SYS$BATCH` by default. The first parameter (P1), passed to the submitted procedure, is always the name of a temporary file containing the mail message that the procedure must delete. Any *parameter(s)* specified in the alias file are passed to the submitted procedure in a single string as its second parameter (P2). `alias2 : "|filespec p2 p3" ;`
>device:[directory]mail.file	Appends mail to the specified file. If no filetype is specified, the default is *.yyyy-mm* (*yyyy* and *mm* are set to the current year and month, respectively). `alias3 : ">filespec" ;`

Rejecting Mail Messages

The SMTP server supports a set of rules for rejecting mail messages received by itself based on the mail header contents or any combination of MAIL FROM, RCPT TO, and source IP address values. Mail matching the criteria can be quietly ignored or rejected with a message to the SMTP client or delivered to an address rewritten according to the rule specification. This capability can be useful for controlling SPAM and preventing your system from being used as a mail relay.

The file `TCPWARE:SMTP_SERVER_REJECT.` contains the rejection and rewrite rules. You may specify an alternate file via the logical name `TCPWARE_SMTP_SERVER_REJECT_FILE`. A rejection file line of the form `#include` *`device`*`:[`*`directory`*`]`*`reject.file`* temporarily suspends processing of the current file and begins processing of the specified file. Rejection files can be nested to arbitrary depth. Comments may be included in rejection files by placing any of the characters `;` or `!` or `#` in the first column of a line.

The following is a sample rejection file:

```
!
! This is a sample reject file for the company EXAMPLE.COM.
!
! This file is processed sequentially. In other words, processing ends on
! the first rule that the message applies to. So if you have a wildcard
! accept at the top of this file, then no other rules will be processed.
!
! Entries can have one of the following formats:
!
!       from_user [from_ip to_user action action-data]
!
!       :rfc822 header
!
! Wildcards can be used in FROM_USER, FROM_IP, and TO_USER. ACTION is the
! reject action, which is one of:
!
!       n       Don't reject, but rewrite TO address to be ACTION-DATA.
!               If ACTION-DATA is blank then we simply deliver to TO_USER.
!
!        y       Reject and use optional ACTION-FIELD as a rejection message
!               format that can contain up to three %s formatting
!               designators for mail from, mail to, and local domainname.
!
!       q       Reject quietly -- don't inform Sending SMTP Client that
!               message will be discarded. If only FROM_USER is specified
!                other fields default to FROM_IP=*, TO_USER=*, and ACTION=n.
!
! Don't rewrite or reject any mail to "postmaster*"
!
!       *       * postmaster*     n
!
! Accept all messages with MAIL FROM:<> (bounce messages)
```

```
! This rule is commented out because you probably don't want it, although
! We're _supposed_ to always accept it. This is the main method relay
! attacks use, by always saying they are from <> to take advantage of that
! RFC hole.
!
! <>             *          *          n
!
! Reject anything with a Message-ID that appears to have originated from
! cyberpromo.com or nowhere.com
!
:Message-ID: <*@cyberpromo.com>
:Message-ID: <*@nowhere.com>
!
! Reject mail from well-known SPAM sites with sample non-standard error
! messages.
!
<*answerme.com>   *  *  y "Spam from <%s> rejected"
<*cyberpromo.com> *  *  y "Spam from <%s> to <%s> rejected"
<*pleaseread.com> *  *  y "Spam rejected;%.0s%.0s Contact postmaster@%s"
!
! Disallow percent-hacks (e.g, joe%somewhere.com@example.com)
!
*    *  *@*@*example.com        y   "No forwarding-path relaying allowed"
!
! Disallow "!" UUCP hacks (e.g. somewhere.com!joe@example.com)
!
*    *  *!*                     y   "No UUCP relaying allowed"
!
! Rewrite all mail to webmaster to the postmaster
!
*    *  webmaster@*example.com  n   postmaster@example.com
!
!
! Disallow relaying through our mailer, and only allow users on our
! networks to claim to be from our company (example.com)
!
*       *     *@example.com    n
*       *     *@daisy.example.com    n
*       *     *@[10.0.0.1]     n
!
<*example.com> 10.0.0.*      * n
<*example.com> 10.115.140.*  * n
<*example.com> 10.115.141.*  * n
!
! Allow our internal systems to bounce mail out.
!
<>    10.0.0.*       *   n
<>    10.115.140.*   *   n
<>    10.115.141.*   *   n
!
! If a message has slipped through all the tests above, then we want to
! reject it, as they are either relaying through us or it's not a valid
```

```
! MAIL FROM.
!
*@*        *        *@*       y    "no relaying through this site"
*          *        *@*       y    "missing domain name in MAIL FROM"
!
!end of sample file
```

Mail rejection rules have two formats:

- `:`*`RFC822_header pattern`*

  This format causes rejection of any mail in which a line with the specified header matches the given *`pattern`*. The following rejection message is sent to the client:

  ```
  554 Message rejected due to header contents
  ```

  > **Note:** Use caution when rejecting mail based on header contents. No other criteria are considered during rejection processing.

- *`from_user ip_address to_user action action_data`*

  This format causes rejection or alternate delivery of all messages that match all of the patterns specified. The *action* item can be as follows:

`n`	Means do not reject the mail, but deliver it to the address specified as the *`action_data`*. If *`action_data`* is not specified, deliver the message to its intended recipient.

`y`	Means reject the mail, sending the *`action_data`* string to the SMTP client as a rejection message. The *`action_data`* item is used as a format string and may contain from one to three `%s` formatting designators to include the *`from_user`*, the *`to_user`*, and the SMTP server name, **in that order**. If *`action_data`* is missing, the default rejection message is
	
	`553-Mail to user not allowed;`
	`553 Contact Postmaster@host.com to remove block`

`q`	Means reject the mail, but do not give the SMTP client any indication that it has been rejected. Use caution when rejecting messages quietly.

Each of the pattern specifications *pattern*, *from_user*, *ip_address*, and *to_user* may contain the OpenVMS * and % wildcard characters.

You can represent *from_addr* expressions in the SMTP_SERVER_REJECT filter with the <> syntax. So, *@*domain.com and <*@*domain.com> are the same expression. To allow all bounce mail in without the filter rules being applied, add the following line to the top of your SMTP_SERVER_REJECT file:

```
<> * * n
(Accept any mail with a MAIL FROM: of <>)
```

When comparing the RCPT TO: address with the SMTP_SERVER_REJECT file expressions, any % signs in the RCPT TO: address are changed to @. You can write filter rules in the SMTP_SERVER_REJECT files that can match against forward-path relays. You can add the rule of

```
* * *@*@localdomain y "No forward-path relaying allowed"
```

to your SMTP_SERVER_REJECT file above the rules that accept mail with the destination of your domains. RCPT TO.: addresses will replace any % character with the @ character for matching purposes only so you can filter with *@*@*-type rules. So, RCPT TO.:<*xxx*%*yyy*@*zzz*> is changed to *xxx*@*yyy*@*zzz*. You can use the logical name TCPWARE_SMTP_SERVER_REJECT_INFO to control debug and informational OPCOM messages produced during rejection processing. You should define it to have some non-zero value to request OPCOM messages. The following values may be combined to control message quantity and content:

Values	To show...
1	mail rejected due to *actiony*
2	rewritten addresses (*actionn* with *action_data*)
4	the reject message sent to the remote system
8	configuration file parsing
16	non-written addresses (*actionn* and no *action_data*)
32	mail rejected due to *actionq*
64	mail rejected due to header rules

The value 65 is appropriate for auditing rejection activity.

The remainder of this chapter describes the configuration tasks.

Network Service Monitoring

Partial support for RFC 2789 (Mail Monitoring MIB) has been added to SMTP. This feature can be enabled from `@TCPWARE:CNFNET SMTP`. Information is maintained only while the service is active. The following items from the Mail Monitoring MIB are available in the enterprises.105.3.25 MIB:

`MtaReceivedMessages`	The number of messages received since Message Transfer Agent (MTA) initialization. (enterprises.105.3.25.1)
`MtaStoredMessages`	The total number of messages currently stored in the MTA. (enterprises.105.3.25.2)
`MtaTransmittedMessages`	The number of messages transmitted since MTA initialization. (enterprises.105.3.25.3)
`MtaReceivedVolume`	The total volume of messages received since MTA initialization, measured in kilo-octets. (enterprises.105.3.25.4)
`MtaStoredVolume`	The total volume of messages currently stored in the MTA, measured in kilo-octets. (enterprises.105.3.25.5)
`MtaTransmittedVolume`	The total volume of messages transmitted since MTA initialization, measured in kilo-octets. (enterprises.105.3.25.6)
`MtaReceivedRecipients`	The total number of recipients specified in all messages received since MTA initialization. (enterprises.105.3.25.7)
`MtaStoredRecipients`	The total number of recipients specified in all messages currently stored in the MTA. (enterprises.105.3.25.8)

MtaTransmittedRecipients	The total number of recipients specified in all messages transmitted since MTA initialization. (enterprises.105.3.25.9)
MtaSuccessfulConvertedMessages	The number of messages that have been successfully converted from one form to another since MTA initialization. (enterprises.105.3.25.10)
MtaFailedConvertedMessages	The number of messages for which an unsuccessful attempt was made to convert them from one form to another since MTA initialization. (enterprises.105.3.25.11)
MtaLoopsDetected	A message loop is defined as a situation where the MTA decides that a given message will never be delivered to one or more recipients and instead will continue to loop endlessly through one or more MTAs. This variable counts the number of times the MTA has detected such a situation since MTA initialization (enterprises.105.3.25.12)

This information can be displayed with the `NETCU SHOW SNMP MIB_VARIABLE` command. See the `SHOW SNMP` command in the *TCPware NETCU Command Reference*.

This feature requires the SNMP Agent X functionality; to use this SNMP must be configured to have Agent X service enabled, and to allow 127.0.0.1 to be an AGENTX_PEER. See Chapter 7 for more information on SNMP and Agent X.

The file `TCPWARE:SMTP_LOCAL_LOGICALS.COM` can be created to define all local SMTP-related logicals. It executes each time SMTP starts.

Session Accounting

TCPware 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 from `@TCPWARE:CNFNET ACCOUNTING` and reads `TCPWARE:ACCOUNTING.CONF` for additional configuration information. The format of the accounting records is described in `TCPWARE_ROOT:[TCPWARE.EXAMPLES]ACCOUNTING.H`

A sample program using this is in TCPWARE_ROOT:[TCPWARE.EXAMPLES]ACC_DUMP.C

Configuring Session Accounting

To configure session accounting, follow these steps:

1. Edit the accounting configuration file, as described in the next section.
2. Invoke the CNFNET procedure by entering the following command at the DCL prompt:

```
$ @TCPWARE:CNFNET ACCOUNTING
```

3. Restart TCPware or accounting.

Configuration File

The accounting configuration file is TCPWARE: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.
- 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.

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 TCPWARE: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 TCPWARE: device is assumed if a device is not specified as part of the file specification.

Enabling the Session Accounting Facility

```
$ @TCPWARE:CNFNET ACCOUNTING
```

Configuring the Accounting listener

TCPware accounting consists of two components: The accounting record logger, which this procedure configures and controls, and the services that can use the accounting process.

This procedure controls the startup of the accounting record logger. The details such as the name of the accounting file, the port that the accounting record logger listens on, and the list of IP addresses that can use the accounting logger are controlled by TCPWARE:ACCOUNTING.CONF:

```
Do you want to activate the Accounting listener on this host [NO]: Y
```

Displaying the Contents of the Logging File

To view accounting information, do the following:

```
$ TCPWARE ACCOUNTING/INPUT=data_file [/output=output file] -
 $ [/since=start_date] [/before=end_date] [/protocol={SMTP, FTP,
MAIL}] [/CSV]
```

- *data_file* is the name of the logging file you want to see.
- *output file* 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. 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 generates Comma Separated Values, for input to products like Excel.

Accounting file record format

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

```
struct accountingPDU {
    char version;
    char type;              /* type of record */
/*
 * FTP: C - Client
 *      S - Server
 * SMTP: N - Network delivery (send)
 *       L - Local delivery (received)
 *       R - Rejected message
 */
    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];     /* VMS time that session started */
    int  end_time[2];       /* VMS 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 */
};
```

Configuring Mail Parameters

The parameters that control the operations of the TCPware mailer are described in the table at the beginning of this chapter.

Configuring Mail Parameters with MAIL-CONFIG

To configure mail parameters with the MAIL-CONFIG utility:

1. Start MAIL-CONFIG with the TCPWARE CONFIGURE /MAIL command.
2. Use the SET *parameter_name* commands.
3. Save the configuration with the SAVE command.
4. Quit MAIL-CONFIG with the QUIT command.

The modified configuration takes effect the next time your system reboots or the queues are restarted.

Mail Parameters

The below table describes all the mail parameters you can set with the MAIL-CONFIG utility.

Parameter	Description
ALIAS-FILE	File in which SMTP aliases are stored; see the *Configuring Mail Aliases* section.
DECNET-DOMAIN	Domain name for DECnet gateway function; see the *Configuring the SMTP-DECnet Mail Gateway* section.
DELIVERY-RECEIPTS	When TRUE, causes the SMTP processor to honor Delivery-Receipt-To: headers.
DISABLE-PSIMAIL	When TRUE, the SMTP symbiont looks for messages addressed through PSImail (usually of the form PSI%*address*::*user*) and returns messages with those addresses to the sender marked "user unknown."
DISALLOW-USER-REPLY-TO	When TRUE, prevents OpenVMS MAIL users from setting a Reply-To: header address with the logical name TCPWARE_SMTP_REPLY_TO.
FORWARDER	Identifies a host (known as a *mail hub*) to which mail should be forwarded if a host name cannot be resolved for an address. The

	specified name must resolve to an IP address; it must not resolve to an MX record.
FORWARD-LOCAL-MAIL	Forwards all mail designated for local users to the mail hub instead of delivering it locally. Can be overridden by entries in the SMTP_ALIASES file.
FORWARD-REMOTE-MAIL	Forwards all SMTP-delivered mail to the mail hub instead of directly to the destination host. Can be overridden by a GATEWAY or LOCAL-DOMAIN entry.
HEADER-CONTROL	Controls which RFC-822 headers appear in messages delivered to OpenVMS MAIL users.
HOST-ALIAS-FILE	Contains a list of host names considered aliases for the local host name.
LOCAL-MAIL-FORWARDER	Identifies a host to which mail should be forwarded when a local mail delivery fails because the username is unknown.
POSTMASTER	Identifies the username of the system postmaster.
QUEUE-COUNT	Specifies the number of mail processing queues to create on a particular system.
REPLY-CONTROL	Specifies how Internet mail headers should be mapped to the OpenVMS MAIL from header. Permitted values are FROM and REPLY-TO. You may specify both as a comma-separated list.
RESENT-HEADERS	When FALSE, the SMTP symbiont omits the Resent-From, Resent-To, and Resent-Date headers that are usually included when a message is forwarded using an OpenVMS MAIL forwarding address.
RETRY-INTERVAL	Specifies the amount of time (in minutes) that should elapse after a failed attempt before another attempt is made.
RETURN-INTERVAL	Specifies the amount of time (in hours) a message can remain in the processing queue before it is returned to sender.

SEND-BROADCAST-CLASS	Controls the OpenVMS broadcast class used by SMTP for SEND-type messages (which are sent to a terminal).
SMTP-HOST-NAMES	A list of up to 16 host names to consider as aliases for the local host. See the *Specifying SMTP Host Aliases* section.
START-QUEUE-MANAGER	Determines whether START_SMTP.COM starts the OpenVMS queue manager if it is not already running.

Configuring the SMTP Server for Inbound Mail

The TCPware SMTP server accepts mail from remote hosts and delivers it to users' mailboxes.

By default, the SMTP server is enabled when you install TCPware.

Translating UNIX-Style Linefeeds to SMTP-Compliant End-of-Line Character Sequences

TCPware provides a logical name to solve the problem of systems sending messages containing lines terminated by an LF character only instead of the proper CR/LF sequence. The following command tells TCPware to accept the bare LF as the end-of-line indicator:

```
$ DEFINE/SYSTEM/EXEC TCPWARE_SMTP_ACCEPT_UNIX_LF TRUE
```

TCPware lets you validate the contents of the envelope-from field by defining the system-wide logical name TCPWARE_SMTP_REJECT_INVALID_DOMAINS. Use the equivalence string STRICT to require the presence of a host in those addresses. For example, require MAIL FROM: *user@host* rather than MAIL FROM: *user*. The host specified in the MAIL FROM: address must exist in the DNS database.

Configuring the SMTP Symbiont and Mail Queues for Outbound Mail

TCPware lets users send mail to remote destinations by submitting outbound messages to mail queues that are processed by TCPware's SMTP symbiont.

You can configure the SMTP symbiont to:

- Control users' ability to specify their own REPLY-To: headers (see the *Specifying the REPLY_TO Header* section.)

- Provide more than one server queue for each cluster node. By default, TCPware provides one server queue for each cluster node running TCPware (see the *Configuring Mail Queues* section.).
- Forward mail through a central mail hub (see the *Forwarding Mail through a Mail Hub* section.)
- Use gateways to reach specific hosts, domains, or "virtual" domains (see the *Configuring Mail Gateways* section.)
- Use host aliases (see the *Specifying SMTP Hosts Aliases* section.)
- Use mail aliases (see the *Configuring Mail Aliases* section.)

You can also write your own SMTP dispatcher by modifying and compiling the SMTP user exit TCPWARE_ROOT:[TCPWARE.EXAMPLES]USER_SMTP_DISPATCH.C. Instructions for modifying the dispatcher are outside the scope of this manual.

For outbound mail, TCPware SMTP eases the 255-character limitation on RFC-822 To: and CC: header lengths. The limit of 255 characters was imposed because some mail applications cannot handle headers longer than 255 characters.

The default header length is 1024 characters. The maximum length is configurable by defining the logical name TCPWARE_SMTP_MAXIMUM_822_TO_LENGTH to be the maximum number of characters to allow in the header. If that maximum is exceeded, only as many addresses as will fit are put into the header. OpenVMS MAIL always creates X-VMSmail-To: and -CC: headers that contain all of the given addresses.

To automatically lower the case of usernames in outbound messages, define the logical name TCPWARE_VMSMAIL_LOCASE_USERNAME.

Specifying the REPLY_TO Header

The TCPWARE_SMTP_REPLY_TO logical name lets you specify the value for the RFC822 REPLY-TO: header. For example, to set your Reply-To: header to FNORD@EXAMPLE.COM, use the command:

```
$ DEFINE TCPWARE_SMTP_REPLY_TO "FNORD@EXAMPLE.COM"
```

This logical name only affects mail agents that use the SMTP% interface (for example, OpenVMS and DECwindows mail). The system manager can disable the use of this logical name with the SET DISALLOW-USER-REPLY-TO command of the TCPWARE CONFIGURE /MAIL utility.

Disabling VRFY and EXPN

To disable VRFY and EXPN processing, use the logical name TCPWARE_SMTP_SERVER_DISABLE_VRFYEXPN. Define it to have some non-zero value to disable the requisite functions. The following values may be used to specify which function to disable:

Value	Function
1	to disable VRFY
2	to disable EXPN
3	to disable both VRFY and EXPN

Configuring Mail Queues

TCPware uses OpenVMS server queues for SMTP processing. Initially, TCPware configures each cluster node running TCPware with a server queue and configures a generic queue for the entire cluster. New messages are placed in the generic queue for processing, which distributes mail processing to the first available server queue.

For example, if three clustered nodes, Huey, Louey, and Dewey, are running TCPware, TCPware creates three server queues and one generic queue. The queue names are:

```
SMTP_HUEY         [Execution queue]
SMTP_LOUEY        [Execution queue]
SMTP_DEWEY        [Execution queue]
TCPWARE_SMTP      [Generic queue]
```

The following example lists the queues for node Huey:

```
$ SHOW QUEUE TCPWARE_SMTP/FULL
Generic server queue TCPWARE_SMTP
       /GENERIC=(SMTP_HUEY,SMTP_LOUEY,SMTP_DEWEY) /OWNER=[SYSTEM]
       /PROTECTION=(codes)

$ SHOW QUEUE SMTP_HUEY/FULL
Server queue SMTP_HUEY, idle, on HUEY::, mounted form DEFAULT
     /BASE_PRIORITY=4 /DEFAULT=(FEED,FORM=DEFAULT) /OWNER=[SYSTEM]
     /PROCESSOR=TCPWARE_SMTP_SYMBIONT /PROTECTION=(codes)
```

The queues `SMTP_LOUEY` and `SMTP_DEWEY` are also created, and are similar to the `SMTP_HUEY` queue shown. Note that a standalone (non-clustered machine) has two queues created by default; that is, one generic queue (`TCPWARE_SMTP`) and one execution queue (`SMTP_`*`nodename`*).

Configuring Multiple Queues

If mail traffic is heavy on your system, you can configure multiple server queues on one or more nodes using `MAIL-CONFIG`. To configure multiple queues with the `MAIL-CONFIG` utility:

1. Start MAIL-CONFIG with the TCPWARE CONFIGURE /MAIL command.
2. Use the SET QUEUE-COUNT command to specify the number of queues on the node.
3. Save the configuration with the SAVE command.
4. Quit MAIL-CONFIG with the QUIT command.

The modified configuration takes effect the next time your system reboots.

Configuring Queue Groups

In heterogeneous cluster environments, you may need to partition mail processing by grouping homogeneous subsets of your cluster into queue groups using MAIL-CONFIG.

> **Note:** Queue groupings can be displayed by starting MAIL-CONFIG and issuing the SHOW command.

To configure queue groups with the MAIL-CONFIG utility:

1. Start MAIL-CONFIG with the TCPWARE CONFIGURE /MAIL command.
2. Use the ADD QUEUE-GROUP and DELETE QUEUE-GROUP commands to add or delete queues.
3. Save the configuration with the SAVE command.
4. Quit MAIL-CONFIG with the QUIT command.

The modified configuration takes effect the next time your system reboots.

Forwarding Mail through a Mail Hub

Many sites provide outbound e-mail access to the Internet through a single system known as a *mail hub* to deliver all outbound mail on behalf of the other hosts at the site. A mail hub typically implements a single-address scheme for e-mail users at the site, so that all users have addresses of the form *username@sitename* rather than *username@hostname.sitename*. Site administrators often configure mail hubs to provide Internet e-mail access to hosts that do not have direct access to the Internet. To forward mail through a mail hub:

1. Specify the host that will serve as a mail hub.
2. Specify the conditions under which TCPware forwards mail to the mail hub.

The following sections describe these procedures.

Specifying a Mail Hub

To specify the host that will serve as a mail hub for your TCPware host:

1. Start MAIL-CONFIG with the TCPWARE CONFIGURE /MAIL command.
2. Modify the FORWARDER parameter using the SET FORWARDER *mailhub_hostname* command.
3. If desired, set any of the following conditions for forwarding mail to the mail hub:
   - Forward mail addressed to users on remote hosts (see the *Forwarding Mail Addressed to Remote Hosts* section.)
   - Exclude hosts in specific domains from remote mail hub forwarding (see the *Excluding Hosts in Specific Domains from Mail Forwarding* section.)
   - Forward mail addressed to users on the local host (see the *Forwarding Local Mail* section.)
   - Exclude specific local users from mail hub forwarding (see the *Excluding Specific Local Users from Mail Forwarding* section.)
4. Save the configuration with the SAVE command.
5. Quit MAIL-CONFIG with the QUIT command.
6. Make the changes take effect immediately by stopping and restarting the mail queues. To update the OpenVMS cluster, use the @TCPWARE:START_SMTP.COM command. To update the local host only, use the @TCPWARE:START_SMTP_LOCAL.COM command. Otherwise, your changes take effect the next time you reboot your system.

Forwarding Mail Addressed to Remote Hosts

To configure TCPware to forward mail addressed to remote users via a mail hub:

1. Make sure the FORWARDER parameter specifies the host you want to use as a mail hub (see the *Specifying a Mail Hub* section.).
2. Start MAIL-CONFIG with the TCPWARE CONFIGURE /MAIL command.
3. Modify the FORWARD-REMOTE-MAIL parameter using the SET FORWARD-REMOTE-MAIL TRUE command.
4. If desired, exclude hosts in specific domains from mail hub forwarding (see the *Excluding Hosts in Specific Domains from Mail Forwarding* section.).
5. If desired, specify other conditions under which TCPware forwards mail to the mail hub (see the *Specifying a Mail Hub* section.).
6. Save the configuration with the SAVE command.
7. Quit MAIL-CONFIG with the QUIT command.
8. Make the changes take effect immediately by stopping and restarting the mail queues with @TCPWARE:START_SMTP.COM to update the OpenVMS cluster or with @TCPWARE:START_SMTP_LOCAL.COM to update the local host only. Otherwise, your changes take effect the next time you reboot your system.

Excluding Hosts in Specific Domains from Mail Forwarding

If you configure TCPware to forward mail addressed to remote users via a mail hub (see the *Forwarding Local Mail* section), you can exclude hosts in specific domains from the mail forwarding system by adding the domain to a list of "local domains." To modify the local domain list:

1. Make sure remote mail forwarding is enabled (see the *Forwarding Mail Addressed to Remote Hosts* section.).
2. Start `MAIL-CONFIG` with the `TCPWARE CONFIGURE /MAIL` command.
3. Add a domain to the list using the `ADD LOCAL-DOMAIN` *`domain_name`* command. If *`domain_name`* begins with a dot, it specifies a domain name. Otherwise, *`domain_name`* specifies a host name.
4. Delete a domain from the list using the `DELETE LOCAL-DOMAIN` *`domain_name`* command.
5. Save the configuration with the `SAVE` command.
6. Quit `MAIL-CONFIG` with the `QUIT` command.
7. Make the new configuration take effect immediately by stopping and restarting the mail queues with `@TCPWARE:START_SMTP.COM` to update the OpenVMS cluster or with `@TCPWARE:START_SMTP_LOCAL.COM` to update the local host only. Otherwise, your changes take effect the next time you reboot your system.

Forwarding Local Mail

To configure TCPware to forward mail addressed to local users via a mail hub:

1. Make sure the `FORWARDER` parameter specifies the host you want to use as a mail hub (see the *Specifying a Mail Hub* section.).
2. Start `MAIL-CONFIG` (`TCPWARE CONFIGURE /MAIL`).
3. Modify the `FORWARD-LOCAL-MAIL` parameter using the `SET FORWARD-LOCAL-MAIL TRUE` command.
4. If desired, exclude specific local users from mail hub forwarding (see the *Excluding Specific Local Users from Mail Forwarding* section.).
5. If desired, specify other conditions under which TCPware forwards mail to the mail hub (see the *Specifying a Mail Hub* section.).
6. Save the configuration with the `SAVE` command.
7. Quit `MAIL-CONFIG` with the `QUIT` command.
8. Make the changes take effect immediately by stopping and restarting the mail queues with `@TCPWARE:START_SMTP.COM` to update the OpenVMS cluster or with `@TCPWARE:START_SMTP_LOCAL.COM` to update the local host only. Otherwise, your changes take effect the next time you restart your system.

Excluding Specific Local Users from Mail Forwarding

If you configure TCPware to forward local mail via a mail hub (see the *Forwarding Local Mail* section), you can exclude specific local users from the mail forwarding system by creating mail aliases for them in the TCPWARE:SMTP_ALIASES file. Each users' alias must be in the following format:
username: *;

For more information on configuring mail aliases, see the *Configuring Mail Aliases* section.

Configuring Mail Gateways

You can configure TCPware with gateways to particular hosts or domains to override the normal host lookup used by SMTP or to configure "virtual" domains not actually present on the network. You can use MAIL-CONFIG. To configure mail gateways with MAIL-CONFIG:

1. Start MAIL-CONFIG with the TCPWARE CONFIGURE /MAIL command.
2. Use the ADD GATEWAY and DELETE GATEWAY commands.
3. Save the configuration with the SAVE command.
4. Quit MAIL-CONFIG with the QUIT command.

The modified configuration takes effect the next time your system reboots.

Specifying SMTP Host Aliases

If your system is a member of a OpenVMS cluster, you can define *host aliases*, which are host names interpreted by the mailer as aliases for the actual local host name. You can specify these aliases in return addresses for individual users.

Setting Host Aliases

TCPware relies on two logical names as parameters to obtain its list of host aliases:

SMTP-HOST-NAMES	Is a comma-separated list of up to 16 host aliases. If defined, the first alias in the list is the name used for outgoing mail. Any aliases are names for which your host accepts incoming mail.
HOST-ALIAS-FILE	Is the complete file specification of a file containing an unlimited list of host alias entries (one entry per line). The HOST-ALIAS-FILE value defaults to TCPWARE:SMTP_HOST_ALIAS.

To change your host aliases with MAIL-CONFIG:

1. Use the SET SMTP-HOST-NAMES command or the SET HOST-ALIAS-FILE command.
2. Save the modified configuration with the SAVE command.
3. Quit MAIL-CONFIG with the QUIT command.

The new configuration takes effect the next time you reboot the system or the queues are restarted.

Specifying Host Aliases for Individual Users

The logical name TCPWARE_SMTP_FROM_HOST lets you change the host name that appears in your return address on outgoing mail.

Normally, the host name you choose must be a "local" host name; that is, it must be one of the registered SMTP host name aliases on the system (either from the SMTP-HOST-NAMES setting or the HOST-ALIAS-FILE). If it is not a known alias, the setting is ignored.

If you define the host name in executive mode, however, TCPWARE_SMTP_FROM_HOST can be any arbitrary host name. The name is not checked against the SMTP host name.

This feature lets users from different administrative entities within an organization have return addresses that reflect the names of those entities. To enable this feature:

1. Set up MX records in DNS so mail is routed to the local host for each separate host name. For information about MX records, see the discussion of zone files in Chapter 6.
2. Set up SMTP-HOST-NAMES or the HOST-ALIAS-FILE with a list of host names.
3. Define the logical name TCPWARE_SMTP_FROM_HOST for each user. Base the value for this logical name on some aspect of the department or organization to which the user belongs.

Configuring Mail Aliases

The TCPware SMTP system supports system-wide mail aliases, system-wide mailing lists, and per-user mail aliases. The default system-wide alias file is TCPWARE:SMTP_ALIASES. You can configure this name or specify a list of alias file names.

Per-user mail aliases are kept in the file SMTP_ALIASES in each user's login directory. The format for alias entries is: *alias*: *real_address*[,...];

- *alias* is an alphanumeric string.
- *real_address* is either a local or remote electronic mail address.

You can specify multiple addresses by separating them with commas; the alias definition may span multiple lines, if needed, and must always be terminated with a semicolon (;).

For example, a local user has the user name "JB134A", but wants to receive SMTP mail sent to the address "john". The system manager adds the following line to the system-wide alias file:
```
john: jb134a;
```

You can both forward a mail message and deliver it to a local mailbox by adding the mailbox name, preceded by an underscore, to the TCPWARE:SMTP_ALIASES file.

The following example shows such an alias entry:

```
FNORD: FNORD@SOMEWHERE.EXAMPLE.COM, _FNORD;
```

The leading underscore on the second address (_FNORD), tells the SMTP symbiont to skip any further alias processing.

Mailing Lists

Mailing lists are a special form of mail alias and are supported only in the system-wide alias files. The format for specifying a mailing list is: *list-name*:: *owner-address*, *file-spec*;

- A double-colon (::) signifies that this alias is a mailing list.
- *owner-address* is the address of the mailing list owner. Messages sent to this mailing list go to each subscriber on the list with the return-path set to this address. The owner address can be an actual user's address or an alias, if desired.
- *file-spec* is the file specification for the file containing the subscribers to the mailing list. Specify a complete path name for this file, including the device and directory.

For example, you might want to set up a mailing list called OPERATIONS-STAFF for your operations staff, and have your operations manager, user OPER1, manage that list. You might set up the mailing list this way:

```
Operations-Staff:: Operations-Manager, USERS:[OPER1]STAFF.LIST;
Operations-Manager: OPER1;
```

Mail sent to OPERATIONS-STAFF is forwarded to the addresses listed in USERS:[OPER1]STAFF.LIST. Because this file is in OPER1's area, the operations manager has control over who is included in the list. The list is set up in this example so the return-path on list messages is set to "Operations-Manager" instead of user OPER1; setting up the list owner as an alias makes it easier to change list owners later.

Specifying the System-Wide Mail Alias File

By default, the TCPware SMTP system obtains system-wide mail aliases from the TCPWARE:SMTP_ALIASES file. You can configure TCPware to use any other file, or to use multiple files. To change the SMTP aliases file with MAIL-CONFIG:

1. Use the SET ALIASES-FILE command.
2. Save the modified configuration with the SAVE command.

The new configuration takes effect the next time you reboot the system.

Using Mail Aliases and Mailing Lists From OpenVMS MAIL

If you want aliases configured within the TCPware SMTP alias file to be accessible to local OpenVMS MAIL users (or those connected via DECnet), specify the address using the OpenVMS MAIL foreign mail protocol interface.

For example, a local user wanting to send mail to the "gcc-users" mailing list would specify the address SMTP%"gcc-users". Note that you can, however, define an OpenVMS MAIL alias containing the SMTP% specification.

To define the OpenVMS MAIL alias "Operations-Staff," use the OpenVMS `MAIL SET FORWARD` command:

```
MAIL> SET FORWARD "SMTP%""Operations-Staff-USERS""" /USER=Operations-Staff
```

TCPware SMTP uses the RFC-822 To: and CC: headers to provide the contents of the OpenVMS MAIL To: and CC: fields. To enable this processing, define the logical name `TCPWARE_VMSMAIL_USE_RFC822_TO_HEADER`.

IMAP Server

The Internet Message Access Protocol (IMAP) server lets an IMAP-compliant client mail program access remote message storage as if the storage were local. TCPware's implementation is based on IMAP Version 4, Revision 1.

IMAP and the Post Office Protocol (POP3), described in the next section, operate differently. IMAP retains the message on the server, whereas POP3 retrieves the message and stores it "off-line" on the client, thereby deleting it from the mail server. IMAP does not delete the mail message and lets you access your mail from more than one client host at a time.

IMAP was designed to:

- Be fully compatible with Internet messaging standards, such as MIME.
- Allow message access and management from more than one computer.
- Allow access without relying on less efficient file access protocols.
- Provide support for "online," "offline," and "disconnected" access modes
- Support concurrent access to shared mailboxes.
- Eliminate the need for the client software to know about the server's file storage format.

The IMAP protocol includes operations for:

- Creating, deleting, and renaming mailboxes

- Checking for new messages
- Permanently removing messages
- Setting and clearing flags
- Server-based RFC-822 and MIME parsing and searching
- Selective fetching of message attributes, texts, and portions thereof, for efficiency

Use `CNFNET`, `@TCPWARE:CNFNET IMAP` to enable or disable the IMAP server.

`CNFNET` asks if you want to use the IMAP server, and if so, provides additional prompts:

- Enter the user (account) the IMAPD (daemon) process should execute as. The default is SYSTEM. Whatever user you choose must have SYSNAM, TMPMBX, NETMBX, SYSPRV, and BYPASS privileges.
- Do you want to enable full message caching? The default is `NO`.

  The IMAP Server usually caches only the text of the last accessed message, in addition to the attributes of all messages in the currently selected folder. Enabling message caching causes the server to cache the text of all messages once seen and until the folder is closed. This can increase server performance but requires considerably more memory.

- What is the desired logging level? The options are:
  - `NONE` - Does no error logging (the default)
  - `ERROR` - Logs errors only
  - `INFO` - Logs errors and informational messages
  - `DEBUG` - Complete debug logging

  TCPware creates a log file, `TCPWARE_SPECIFIC:[TCPWARE]IMAPSERVER.LOG`, when the server is started.

By default, the User Authorization File (UAF) record for the IMAP user is not affected by a user checking for mail. However, if you want the `LAST-NON-INTERACTIVE-LOGIN-TIME` field in the UAF record for each IMAP user to be updated with the current system date and time, the system administrator can define the following logical system-wide:

```
$ DEFINE/SYSTEM TCPWARE_IMAP_UDPATE_LOGIN_TIME  1
```

This update takes place at the time the authentication process is successfully completed.

IMAP Mail Folders

In contrast to POP3, IMAP allows you to access server mail folders (message stores) other than INBOX. In OpenVMS MAIL, for example, if you create a NOTES folder, you can access mail in that folder via

the client mail program. This NOTES folder can be in a mail file other than the default `MAIL.MAI` file. In fact, you can set a configuration parameter that determines the way mail folders are presented to the client so that you can use folders in these other mail files.

Your default mail directory includes a `.IMAPRC` file in which you can set certain configuration directives (described more fully in the *IMAP Directives File* section that follows). Among these directives is `allow-subfolders`. This directive specifies that folder names are comprised of a directory (optional), mail file, and folder. For example, the NOTES folder in `MAIL.MAI` is represented as `mail/notes` (as opposed to just `notes` if the directive were not set). This would distinguish it from another NOTES folder in the `OLD.MAI` mail file, for example, which would be named `old/notes`.

Each level beyond the second in this hierarchy represents a subdirectory of the default mail directory. For example, the NOTES folder in `[.ARCHIVED]MAIL.MAI` has the IMAP equivalent of `archived/mail/notes`.

Because of this folder syntax ambiguity, directory names, file names, and folders can overlap, such as the examples in the following table.

This mail file...	Containing this folder...	Has this IMAP equivalent...
MAIL.MAI	NOTES	mail/notes
[.MAIL]NOTES.MAI	STUFF	mail/notes/stuff
[.MAIL.NOTES]STUFF.MAI	BOBS	mail/notes/stuff/bobs

Entries in the syntax can at different times be mail files, directories, subdirectories, or folders. Because of this overlap, the server must keep an internal representation of the hierarchy and mark what each level of the folder name means. This information is critical when renaming or deleting folders. Mail file definitions do not have to match.

One restriction is that a first level folder (MAIL, for example) cannot be a message store, since it represents only a file and not a mail folder. INBOX, however, is a special case. INBOX is always INBOX, cannot be deleted or renamed (a rename moves messages to the renamed folder but does not delete INBOX), and never goes away. In IMAP, INBOX is `mail/NEWMAIL` by default, and is hidden to the user.

You can change the mail inbox from INBOX to another folder by defining the `file-inbox-messages-to-folder` directive in the `.IMAPRC` file. See the next section for details.

You can also access mail files in your login directory (SYS$LOGIN) by prefixing the folder name with a tilde (~). The ~ folder is reserved and cannot be used by other folders.

IMAP Directives File

Users can set certain preferences by creating a file in their default mail directory called .IMAPRC and including directives. The following table lists these directives along with their meanings. Each directive must be on its own line and in lowercase.

This directive...	Does the following...
set allow-child-folders	Enables or disables the use of subfolders and the way that folders are presented to the client. By default, this value is false. If you want to set the value to be true, put this line in the .IMAPRC file: set allow-child-folders true (See the previous section for details.)
set autofile-messages-to-folder *foldername*	Moves read messages from INBOX to the specified folder in the user's default mail file. By default, this option is disabled.
set case-insensitive-folders	Specifies that folder names are case-insensitive. Otherwise, two folders with the same name but with different cases could become inaccessible to the IMAP client. Newly created folders are created in uppercase on the server. By default, this value is false. If you want to set the value to be true, put this line in the .IMAPRC file: set case-insensitive-folders true
set do-purge-reclaim	Enables or disables purge-reclaim operations upon closing a folder. By default, this value is true.

`set folder-timer` *`delta_time`*	Specifies the frequency the IMAP server will check for externally created folders. By default, this value is `00:02:00` (2 minutes).
`set inbox-folder` *`foldername`*	Maps INBOX to the specified folder in the user's default mail file. By default, this value is `NEWMAIL`.
`set new-mail-timer` *`delta_time`*	Specifies, using OpenVMS delta time format, how often the server checks a folder for new mail. (Note that checking for new mail is time consuming for large folders.) The default is `0:0:30` (30 seconds).

IMAP Options in the Global IMAPD.CONF file

These options are valid in the global `IMAPD.CONF` file (`TCPWARE:IMAPD.CONF`) as shown in the below table:

This directive...	**Does the following...**
`set decnet-address` *`nodename namespace domainname`*	Maps the specified DECnet node name and/or namespace to a given domain name. Use " " to represent either a blank node name or namespace. Multiple entries are allowed. By default, this option is disabled. Example: `set decnet-address knob " " door.com`
`set enable-full-cache`	Enables or disables full message caching. By default, this value is `false`.
`set max-ping-count` *`integer`*	Specifies, in number of messages, the threshold at which the server will no longer attempt to check for new messages. By default, this value is `20000`.
`set smtp-transport-prefix` *`string`*	Specifies the SMTP transport prefix. By default, this value is `SMTP`. If you want to change it to `MX`, put this line in the `IMAPD.CONF`:

	`set smtp-transport-prefix MX`
`set trailing-header-marker` *`string`*	Specifies a text string used to indicate the start of RFC822 message headers if the system does not place them at the start of the message. By default, this option is disabled.

IMAP State Information Files

The IMAP server includes files created in the user's mail directory where it maintains state information, as shown in the following table.

This file...	**Stores...**
`.MAILBOXLIST`	Folders to which the user subscribes.
`.NEWMAILBOXES`	List of folders known to be empty. OpenVMS MAIL deletes folders once it deletes the last message, so that the server must "remember" these folders.
`*.MAI-UID`	UIDVALIDITY information for folders in a mail file. Persistent UIDs allow clients to operate in “offline” and “disconnected” modes, and can improve performance if clients implement sensible caching schemes.
`mailfolder.foldernameuidvalidity`	For each folder, the UIDs for all the messages. The file name is composed of the folder name and its UIDVALIDITY code. For example: `MAIL.NEWMAIL3B3200E6` In the example, the folder name is `NEWMAIL` and the UID validity code is `3B3200E6`.

POP3 Server

The Post Office Protocol Version 3 (POP3) is a multithreaded server that can handle up to 31 simultaneous client connections. It does not perform any mail delivery functions but simply allows clients (mostly PCs) to retrieve new mail from OpenVMS MAIL inboxes.

Use `CNFNET`, `@TCPWARE:CNFNET POP3` to enable or disable the POP3 server.

`CNFNET` asks if you want to use the POP3 server, and if so, asks the following additional questions:

- What is the maximum number of new mail messages to return per connection? The default is 32.
- What is the desired logging level? The options are:
  - `ERROR` - Logs errors only (the default)
  - `INFO` - Logs errors and informational messages
  - `THREAD` - Logs errors, informational messages, and detailed thread logging
  - `DEBUG` - Complete debug logging

  TCPware creates a log file, `TCPWARE_SPECIFIC:[TCPWARE]POP3SERVER.LOG`, when the server is started.

- Do you want to do a `MAIL PURGE/RECLAIM` operation for each mailbox after its use?

  If `TRUE`, this not only purges (deletes all messages in) the wastebasket folder, but releases deleted message space back to RMS for reuse. The default is `TRUE`.

POP3 runs on the local mail server so that client software on a PC can check the server for incoming mail and display it on the PC. The POP3 server downloads the incoming mail on its machine to the PC, and deletes the original message. This is known as an “off-line” mail operation, where all message processing is local to the client and distinct from the server.

POP3 processes only new, incoming mail and does not manipulate folders other than INBOX.

The POP3 server first listens on TCP port 110 for a client request (in the form of a greeting) to download mail. The POP3 session then passes through the following three states:

AUTHORIZATION - The client identifies itself to the POP3 server and the server acquires resources associated with the client’s mail drop. By default, the User Authorization File (UAF) record for the POP3 user is not affected by a user checking for mail. However, if you want the `LAST-NON-INTERACTIVE-LOGIN-TIME` field in the UAF record for each POP3 user to be updated with the current system date and time, the system administrator can define the following logical system-wide:

```
$ DEFINE/SYSTEM TCPWARE_POP3_UDPATE_LOGIN_TIME  1
```

This update takes place at the time the authentication process is successfully completed.

TRANSACTION - The client acquires resources and signs off.

UPDATE - The POP3 server releases resources and signs off.

The client sends commands to the POP3 server in each state and the server responds with an affirmative `+OK` (or negative `−ERR`) status. TCPware's POP3 server supports all the minimal commands described in RFC 1939 *Post Office Protocol - Version 3* as well as the optional `TOP` command:

USER	STAT	DELE	QUIT
PASS	LIST	NOOP	UIDL
QUIT	RETR (or TOP)	RSET	

The `APOP` optional command is not supported. For more information on the POP3 commands, see RFC 1939.

Configuring SMTP Service for ALL-IN-1 Users

The TCPware mailer supports users of HP's ALL-IN-1 office automation environment (often referred to as ALL-IN-1 IOS or ALL-IN-1 Classic) and ALL-IN-1 MAIL via an interface to Message Router, the backbone of HP's MAILbus product line. This interface allows both ALL-IN-1 IOS and ALL-IN-1 MAIL users to send and receive SMTP mail. Message Router V3.1 or later is required for this feature to function properly. For information on sending and receiving SMTP mail from within ALL-IN-1 IOS and ALL-IN-1 MAIL, see Chapter 10 in the *User's Guide*.

Before Configuration

You must have Message Router V3.1 or later installed on your system before configuring the TCPware SMTP to Message Router (SMTP/MR) interface. If you want to support automated conversion of WPS and DX documents in ALL-IN-1 messages to ASCII, you must also have the Message Router OpenVMS MAIL Gateway (MRGATE) installation kit.

You do not need to install MRGATE on your system; however, certain object libraries in the MRGATE kit are needed to provide the necessary document conversion functions. The SMTP/MR gateway software functions even if the document conversion is not built. It does, however, cause all WPS and DX message body parts to be discarded as the ALL-IN-1 message passes through SMTP/MR.

Configuring SMTP/MR

The `MR_CONFIGURE.COM` command procedure in the TCPware: directory is used to configure the SMTP/MR gateway software. Execute this procedure with the DCL command:

```
$ @TCPWARE:MR_CONFIGURE
```

The command procedure presents a series of prompts. Enter a question mark (`?`) at any time to display more information about that prompt. The configuration command procedure prompts you for the following information:

1. Whether to display a detailed explanation before each question. It is recommended that you answer `YES` to this question the first time you run the configuration procedure.
2. The domain name of the gateway system. This is a domain-style host name used to refer to the gateway from the Internet. Be sure the name you choose is within a domain or subdomain over which you have administrative authority and is not currently being used for another host. If your local domain is EXAMPLE.COM, a reasonable choice for this domain name would be MR.EXAMPLE.COM. This name is largely for internal use and should not be needed to address mail to ALL-IN-1 users.
3. The domain name to be used for local ALL-IN-1 IOS users. This is a domain-style host name used to indicate to the TCPware mailer that it should pass the message to the Message Router for delivery to an ALL-IN-1 user. Be sure the name you choose is within a domain or subdomain over which you have administrative authority and is not currently being used for another host. If your local domain is EXAMPLE.COM, a reasonable choice for this domain name would be A1.EXAMPLE.COM. Note that if you are not running ALL-IN-1 IOS, you should specify NONE.
4. The domain name to use for local ALL-IN-1 MAIL users. This is a domain-style host name used to indicate to the TCPware mailer that it should pass the message to Message Router for delivery to an ALL-IN-1 user. Be sure the name you choose is within a domain or subdomain over which you have administrative authority and is not currently being used for another host. If your local domain is EXAMPLE.COM, a reasonable choice for this domain name would be AM.EXAMPLE.COM. Note that if you are not running ALL-IN-1 MAIL, you should specify NONE.
5. The Message Router mailbox name used for the gateway. This Message Router mailbox name is used by ALL-IN-1 users to send outbound SMTP mail. You are directed later to create this mailbox with the Message Router MRMAN utility. The default value of "SMTP" should be used.

   Both ALL-IN-1 IOS and ALL-IN-1 MAIL users use the address form *user@host@SMTP* to specify remote SMTP recipients. Each ALL-IN-1 mail utility places this outbound mail in the Message Router mailbox named SMTP. The TCPware mailer "picks up" mail from this mailbox and sends it via the normal SMTP delivery mechanism.

   The current version of TCPware allows both ALL-IN-1 IOS and ALL-IN-1 MAIL users to reply to messages imported with the V command or forwarded into ALL-IN-1 using an address of the form MRGATE:"A1::*user*" or MRGATE:"AM::*user*". Return addresses are translated from Message Router format to a more standard RFC-822 format in a fashion analogous to the DECnet to SMTP gateway conversion.

The following logical names can be used to customize the translation:

`TCPWARE_SMTP_MRGATE-NAME` - Change the name of the Message Router gateway mailbox. The default value is `MRGATE`.

`TCPWARE_SMTP_A1_NAME` - Change the name of the ALL-IN-1 IOS gateway mailbox. The default value is `A1`.

`TCPWARE_SMTP_AM_NAME` - Change the name of the ALL-IN-1 MAIL gateway mailbox. The default value is `AM`.

`TCPWARE_SMTP_A1_DOMAIN` - Specify the RFC-822 domain associated with the ALL-IN-1 IOS gateway. Use the domain you specified when configuring your SMTP/MR gateway.

`TCPWARE_SMTP_AM_DOMAIN` - Specify the RFC-822 domain associated with the ALL-IN-1 MAIL gateway. Use the domain you specified when configuring your SMTP/MR gateway.

6. The Message Router mailbox name used for delivery to ALL-IN-1 MAIL users. This Message Router mailbox is used to deliver inbound SMTP mail to ALL-IN-1 MAIL users. Normally this mailbox is named `AM`, but the name is configurable. If your site uses a name other than AM, enter that name.
7. The Message Router mailbox name used by default when mail is sent to the domain name of the gateway system (for example, MR.EXAMPLE.COM). Specify the default value, `MRGATE`, to indicate the Message Router OpenVMS MAIL Gateway mailbox.
8. The Message Router mailbox name for the local system. Specify the DECnet node name of the local system. SMTP/MR uses this name to generate addresses that are valid on remote systems.
9. The names of any null routes. Specify the name of the local DECnet node and any other nodes in a homogeneous OpenVMS cluster (including the cluster alias). Message Router routing information often includes local DECnet node names or a cluster alias which is superfluous in outbound SMTP mail. To determine the names of any null routes, use the `MRMAN` utility `SHOW *` command. Null routes have the general form:

```
nullname,      Replace=
```

   The names you specify at this point are automatically stripped from addresses passing through SMTP/MR, greatly simplifying them when they pass into the SMTP world. Use a blank entry to terminate the list.

10. The SMTP address of the local postmaster, which should be the full domain-style email address of the user who should receive error messages generated by SMTP/MR.
11. The password associated with this gateway's Message Router mailbox (SMTP by default). Message Router protects each mailbox with a password to ensure privacy of mail messages.

12. Whether you want to have messages with WPS and DX body parts automatically converted to ASCII. SMTP/MR can perform these conversions if it is linked against libraries provided as part of various Message Router products, notably MRGATE. If you intend to have messages in these special formats converted to ASCII, answer `YES`. If you answer `NO`, these body parts are not converted, and may be discarded.
13. You are then prompted for the names of several SMTP/MR configuration files. Accept the defaults for each of these file names.
14. You are then asked if you want to generate the configuration files. If you are satisfied with all of your responses, type `YES`.

In the following example, which shows an `MR_CONFIGURE.COM` session, the local host's DECnet node name is MIGUEL, while its TCP/IP host name is MIGUEL.EXAMPLE.COM. The DECnet cluster alias is IRISDN; and other hosts in the homogeneous VAXcluster are named WHARFIN, BIGBTE, and YAYA. The e-mail address of the local postmaster is POSTMASTER@EXAMPLE.COM.

```
$ @TCPWARE:MR_CONFIGURE

SMTP-MR Configuration Utility

This utility creates an initial pair of databases for mapping SMTP
RFC 822-style addresses to Message Router X.400-style addresses and
back again. Only minimal mappings are created to support ALL-IN-1
users sending and receiving SMTP mail.

Important note: No changes are made to existing SMTP-MR database
information until all questions have been answered. This utility can
be aborted at any prompt by entering a CTRL/Z. The files output by
this utility may optionally be redirected to a different location so
they will have no impact on the existing SMTP-MR databases.

Do you wish to continue [Y]? Return
Do you wish to have a detailed explanation printed before each question [N]?
Return
Domain name of the gateway: MR.EXAMPLE.COM
Domain name for local ALL-IN-1 IOS users [A1.EXAMPLE.COM]: Return
Domain name for local ALL-IN-1 MAIL users [AM.EXAMPLE.COM]: Return
Message Router mailbox name for the gateway [SMTP]: Return
ALL-IN-1 IOS mailbox known to Message Router [A1]: Return
ALL-IN-1 MAIL mailbox known to Message Router [AM]: Return
Message Router mailbox used by default [MRGATE]: Return
Local system's Message Router mailbox [MIGUEL]: Return

Local node name or other null routes [RETURN if no more]: MIGUEL
Local node name or other null routes [RETURN if no more]: WHRFIN
Local node name or other null routes [RETURN if no more]: BIGBTE
Local node name or other null routes [RETURN if no more]: YAYA
```

```
Local node name or other null routes [RETURN if no more]: IRISDN
Local node name or other null routes [RETURN if no more]: Return

RFC822 address of local PostMaster: postmaster@example.com
Password for the Message Router mailbox: example
Convert WPS-PLUS and DX messages to ASCII automatically [Y]? Return
SMTP to MR mapping text file [TCPWARE:TO_MR.]: Return
MR to SMTP mapping text file [TCPWARE:FROM_MR.]: Return
Gateway options file [TCPWARE:MR_OPTIONS.]: Return
SMTP-MR checklist file name [TCPWARE:SMTP-MR.CHECKLIST]: Return

All configuration questions have been answered.
Do you wish to generate the configuration files [Y]? Return
Generating SMTP to MR mapping text file...
SMTP to MR mapping text file is complete.

Generating MR to SMTP mapping text file...
MR to SMTP mapping text file is complete.

Generating the setup checklist...
Checklist file is complete.

Generating options file...
Options file is complete
*****************************************************************
*   To complete your SMTP-MR configuration, carry out the steps
*   detailed in the setup checklist  TCPWARE:SMTP-MR.CHECKLIST.
*****************************************************************
$
```

Configuring SMTP/MR Document Conversion

The DCF document conversion utility is used by SMTP/MR to convert WPS and DX message body parts to ASCII text. This utility is built from document conversion library routines supplied as part of the Message Router OpenVMS MAIL Gateway (MRGATE) distribution kit. SMTP/MR can function without this utility, but cannot convert WPS and DX body parts to ASCII without it. Body parts in outbound SMTP mail that cannot be converted to ASCII are discarded.

The DCF utility is not supplied in executable form with SMTP/MR; it must be built after SMTP/MR is configured. The command procedure `TCPWARE:DCF_BUILD.COM` is provided to accomplish this. This procedure prompts for two items:

- The location of the save set from which to extract the necessary conversion libraries.
- The name of a directory into which the libraries should temporarily be placed while the utility is being linked.

You need not copy the saveset from the distribution media for DCF_BUILD.COM to work. For example, if you want to access the libraries on a TK50 containing MRGATE while on a VAXstation 3100, you would specify the saveset name as MKA500:MRGATE031.A.

The following example shows how to build the DCF utility from a saveset located in the SYS$MANAGER directory:

```
$ @TCPWARE:DCF_BUILD
Directory to put libraries in [SYS$SCRATCH:]: Return
File specification of save set from which to extract libraries:
SYS$MANAGER:MRGATE031.A
Extracting libraries...
%BACKUP-S-CREATED, created SYS$SYSROOT:[SYSMGR]KOALA.OLB;1
%BACKUP-S-CREATED, created SYS$SYSROOT:[SYSMGR]DCF_BASE.OLB;1
%BACKUP-S-CREATED, created SYS$SYSROOT:[SYSMGR]DCF_TRANSLATE.OLB;1
%BACKUP-S-CREATED, created SYS$SYSROOT:[SYSMGR]DCF_MAIL_CONVERSIONS.OLB;1
%BACKUP-S-CREATED, created SYS$SYSROOT:[SYSMGR]DCF_DSAF.OLB;1
%BACKUP-S-CREATED, created SYS$SYSROOT:[SYSMGR]WPADOC.OLB;1
%BACKUP-S-CREATED, created SYS$SYSROOT:[SYSMGR]WPABASE.OLB;1
%BACKUP-S-CREATED, created SYS$SYSROOT:[SYSMGR]XPORT.OLB;1
Linking DCF utility...
Cleaning up...
Done
$
```

The DCF utility is never invoked interactively. It is always invoked automatically by the SMTP/MR gateway whenever it has mail containing WPS or DX body parts it needs to send via SMTP.

> **Note:** You must use the A save set from MRGATE V3.1 or V3.2 to build DCF. Versions V3.3 and later do not contain the object files needed to link to DCF. If you upgrade to MRGATE V3.3, save your V3.1 or V3.2 distribution media.

Completing SMTP/MR Configuration

In addition to the SMTP/MR configuration data files, the file TCPWARE:SMTP-MR.CHECKLIST is created by the MR_CONFIGURE.COM command procedure. This file contains the steps needed to complete the SMTP/MR configuration, which include:

1. Adding the SMTP/MR gateway mailbox to your Message Router configuration. Run the MRMAN utility exactly as documented in the checklist file. The Message Router mailbox name and password must be exactly the same as you specified to MR_CONFIGURE.COM.

2. Building the WPS and DX document conversion utility. See the previous section for details on building this utility.
3. Configuring your Domain Name System (DNS) name server for SMTP/MR operation. You must add a Mail eXchanger (MX) record to your name server configuration for the following:
    - The domain name of the gateway itself (MR.EXAMPLE.COM in the example in the *Configuring SMTP/MR* section)
    - The domain name used to direct mail to your ALL-IN-1 IOS users (A1.EXAMPLE.COM in the example in the *Configuring SMTP/MR* section)
    - The domain name used to direct mail to your ALL-IN-1 MAIL users (AM.EXAMPLE.COM in the example in the *Configuring SMTP/MR* section).

    If the host running SMTP/MR is named MIGUEL.EXAMPLE.COM, the DNS Resource Records (RRs) you would use in the DNS configuration file for the domain EXAMPLE.COM are:

    ```
    MR.EXAMPLE.COM.   IN  MX  0  MIGUEL.EXAMPLE.COM.
    A1.EXAMPLE.COM.   IN  MX  0  MIGUEL.EXAMPLE.COM.
    AM.EXAMPLE.COM.   IN  MX  0  MIGUEL.EXAMPLE.COM.
    ```

    For more detailed information on configuring a DNS name server, see Chapter 6, *Host Tables and DNS*.

4. If you are not running a DNS name server locally, you must add additional host records to the `TCPWARE:HOSTS.LOCAL` file for the host names of the gateway and your ALL-IN-1 users. Using the names from the above example, and assuming that the IP address for MIGUEL.EXAMPLE.COM is 128.0.0.1, you would add the following lines to `TCPWARE:HOSTS.LOCAL`:

    ```
    HOST : 128.0.0.1 : MR.EXAMPLE.COM : : : :
    HOST : 128.0.0.1 : A1.EXAMPLE.COM : : : :
    HOST : 128.0.0.1 : AM.EXAMPLE.COM : : : :
    ```

    Note that you should:

    - Place these lines after the entry for MIGUEL.EXAMPLE.COM.
    - Specify each name on a line by itself. Merging entries in the hosts file prevents the gateway from functioning properly.
    - Recompile and re-install the host tables after adding the new entries.

    For detailed information on adding entries to your host tables, see chapter 6.

5. Submitting the command procedure `TCPWARE:MR_TO_TCPWARE.COM` to the appropriate batch queue on your system. This command procedure runs periodically to transfer mail from the SMTP/MR Message Router mailbox (normally SMTP) to the TCPware mailer. Examine this

command procedure before submitting it to ensure it runs in the batch queue and under the desired user name.

Configuring the SMTP-DECnet Mail Gateway

TCPware can be set up as a gateway to route mail between SMTP and DECnet-only hosts, with appropriate address translations to make the DECnet-style addresses easier for Internet hosts to interpret. To do this, you set the `DECNET-DOMAIN` mail parameter and add an appropriate MX record to the Domain Name Service. The addresses of DECnet mail sent out via SMTP will be rewritten such that the DECnet node name(s) appear under the `DECNET-DOMAIN` name in the host-part of the address. The addresses of incoming SMTP mail for hosts under the `DECNET-DOMAIN` are automatically converted into DECnet addresses and delivered to the DECnet-only hosts.

DECnet-to-SMTP Mail

In the DECnet-to-SMTP direction, an OpenVMS MAIL user on a DECnet-only host sends SMTP mail by specifying an address of the form: `node::SMTP%"user@host"` where *node* is the DECnet node name of the system running TCPware.

TCPware recognizes that the mail originated in DECnet and, if the `DECNET-DOMAIN` parameter is set, rewrites the originating address in the form *user@node.decnet-domain*.

For example, EXAMPLE.COM has set up node HQ as a DECnet-SMTP gateway. A user named JOHN on DECnet-only node WHARFIN at EXAMPLE.COM addresses mail to the Info-TCPware mailing list as: `HQ::SMTP%"Info-TCPWARE@LISTS.PROCESS.COM"`

JOHN's DECnet return address, WHARFIN::JOHN, is rewritten by the gateway as:

```
JOHN@WHARFIN.DNET.EXAMPLE.COM
```

instead of:

```
"WHARFIN::JOHN"@HQ.EXAMPLE.COM
```

which some Internet mailers would have trouble parsing.

SMTP-to-DECnet Mail

For the SMTP-DECnet gateway to work in the SMTP-to-DECnet direction, other hosts on your network must be told that mail for host names under the `DECNET-DOMAIN` should be sent to the gateway host. If you use Domain Name Service, the easiest way to do this is to set up a wildcard MX record for the `DECNET-DOMAIN`. In our example, the MX record looks like this:

```
*.DNET.EXAMPLE.COM.     IN         MX      0     HQ.EXAMPLE.COM.
```

This MX record causes other hosts on the Internet to send mail destined for any host under DNET.EXAMPLE.COM to node HQ. The gateway automatically recognizes the `DECNET-DOMAIN` in the host-name part of the address, rewrites the address to its DECnet form, and sends it through OpenVMS MAIL.

If you do not use DNS, you must add a fully qualified host name for each DECnet node to your host tables. In our example, a return message to user JOHN on node WHARFIN would be addressed to:

```
JOHN@WHARFIN.DNET.EXAMPLE.COM.
```

When HQ receives that message, it translates the address to its DECnet form:

```
WHARFIN::JOHN
```

and sends the message to that address using OpenVMS MAIL.

17. Managing TELNET

Introduction

This chapter describes the TELNET server from the system manager's point of view. Topics include:

- TELNET logicals
- TELNET control functions
- Setting up Virtual Terminals
- TELNET options

RFCs 854 through 858, RFC 885, and RFC 1091 describe the TELNET protocol. The *User's Guide*, Chapter 11, *TELNET: Connecting to Remote Terminals*, the *Command Reference* section describes the commands you can use with TELNET.

TELNET Logicals

The logicals in the below table appear in the `TCPWARE:TELNET_CONTROL.COM` file.

> **Caution:** Editing the `TELNET_CONTROL.COM` file is not recommended, since each TCPware installation replaces this file. If you want to define a special purpose logical, do so in another place, such as in your system startup file.

Logical	Description
`TCPWARE_TELNETD_INTRO_MSG`	Define this logical as a special message that appears whenever a user reaches the host through TELNET. If the system logical name table contains a value for this logical, the server sends the equivalence string to the peer before the standard login sequence

	starts. Use this logical to issue warnings such as `Authorized Use Only` for remote logins.
`TCPWARE_TELNETD_DEFCHAR`	Define this logical to set up the default terminal characteristics for TELNET sessions. You can thereby avoid having to change the `SYSGEN TTY_DEFCHAR` and `TTY_DEFCHAR2` fields system wide. This logical forces the hangup bit set. To prevent the forcing of the hangup bit set, use the `TCPWARE_TELNETD_NO_FORCED_HANGUP` logical.
`TCPWARE_TELNET_WINDOW`	If defined, specifies the window size that the TELNET server offers to the peer. By default, this value is 4096. If the `TCPWARE_TELNET_WINDOW` logical's value is less than 512, TELNET uses 4096.
`TCPWARE_TELNETD_FLAGS`	This logical has several purposes. The default value is 1. The below table describes the bit options. Setting either bit 0 or 1 can improve server performance and reduce system processing overhead. Note, however, that doing so means that you are not adhering to the TELNET protocol.

The logical bit descriptions for the `TCPWARE_TELNETD_FLAGS` logical are:

Bit...	**(Mask)**	**Set to...**	**Description**
0	1	0	Inserts a <NULL> character following a <CR> character that is not followed by an <LF> character. (Ignored if bit 1 is set to 1.)
		1 (default)	Inserts nothing after a <CR> character that is not followed by an <LF> character.
1	2	0 (default)	Server does not run in raw mode. It doubles <IAC> data characters as required by the TELNET protocol.
		1	Ignores bit 0: Sends all characters without special processing, doubling of characters, or inserting <NULL> after <CR>.

7	128	0 (default)	Incoming TELNET sessions are REMOTE.
		1	Incoming TELNET sessions are LOCAL (compatible with earlier TCPware versions).
8	256	0 (default)	Tries to return the peer's hostname in the NTA terminal's `TT_ACCPORNAM` field. Otherwise uses the IP address.
		1	Ignores bit 9: Adds the port number to the IP address, such as 192.168.95.1,1094 (compatible with an earlier TCPware).
9	512	0 (default)	Tries to return the peer's hostname in the NTA terminal's `TT_ACCPORNAM` field. Otherwise uses the IP address.
		1	Adds the port number to either the hostname or IP address, such as bart.example.com,1094 or 192.168.95.1,1094. (Ignored if bit 8 is set to 1.)

The combination of bit 8 and 9 settings has the following effect on `TT_ACCPORNAM`*:

Bit 8	Bit 9	**Returned Remote Port Info (`TT_ACCPORNAM` Field)**
0	0	*host* or *address* (default): `bart.example.com` or `192.168.95.1`
0	1	*host*,*port* or *address*,*port*: `bart.example.com,1094` or `192.168.95.1,1094`
1	0	*address*,*port*: `192.168.95.1,1094`
1	1	*address*,*port*: `192.168.95.1,1094`

*The `TT_ACCPORNAM` field is limited to 63 bytes; if the value is too long, the port number may be truncated or even dropped.

Virtual Terminals

Virtual terminals (VTAs) allow users to disconnect from a physical terminal without terminating a process - the process remains active on a virtual terminal. Virtual terminals are used to reconnect to a process when the connection is lost, and to maintain sessions on more than one disconnected terminal.

To set up the TELNET server to support VTAs:

1. Set up VTA devices as follows:

   VAX:

   ```
   $ RUN SYS$SYSTEM:SYSGEN
   SYSGEN>CONNECT VTA0: /NOADAPTER /DRIVER=TTDRIVER
   SYSGEN>EXIT
   ```

   Alpha/Integrity:

   ```
   $ RUN SYS$SYSTEM:SYSMAN
   SYSMAN>IO CONNECT VTA0 /NOADAPTER -
   _SYSMAN>/DRIVER=SYS$LOADABLE_IMAGES:SYS$TTDRIVER.EXE
   SYSMAN>EXIT
   ```

2. Edit the `TELNET_CONTROL.COM` file to define the `TCPWARE_TELNETD_DEFCHAR` logical (see *TELNET Logicals*):

   ```
   $ DEFINE/SYSTEM TCPWARE_TELNETD_DEFCHAR 402657952,135174
   ```

   `TCPWARE_TELNETD_DEFCHAR` permits TELNET server devices to override the values in the `SYSGEN TTY_DEFCHAR` and `TTY_DEFCHAR2` parameters if they are not set up correctly for TELNET sessions. These SYSGEN parameters apply to all terminals and only take effect after a reboot.

   In the above logical definition:

   - `402657952 (%X180012A0)` is the VMS default value for `TTY_DEFCHAR`.
   - An optional value `135174` (`%X0021006`) for `TTY_DEFCHAR2` is appended.
   - The default value for `TTY_DEFCHAR2` is normally `%X0001002`, which is `TT2$M_SECURE` combined with `TT2$M_AUTOBAUD`.
   - The value `135174` (`%X00021006`) reflects the addition of `TT2$M_DISCONNECT` (`131072`, or `%X00020000`) and `TT2$M_HANGUP` (`4`).
   - (See the `SYS$LIBRARY:TTDEF.H` file and `SYS$LIBRARY:TT2DEF.H` file for bit definitions for `TTY_DEFCHAR` and `TTY_DEFCHAR2`, respectively.)
   - `TT2$M_DISCONNECT` is needed to allow disconnected virtual terminals.

> **Note:** TT2$M_HANGUP is forced on even if it is not specified in the TCPWARE_TELNETD_DEFCHAR logical. If you are using this logical to prevent the TT2$M_HANGUP bit from being set, you need to define the TCPWARE_TELNETD_NO_FORCED_HANGUP logical.

3. Restart TELNET for the changes to take effect:

```
$ @TCPWARE:RESTART TELNET
```

When a client opens a session to the server set up with VTAs and the virtual terminal disconnects, the opening sequence might be as follows:

```
Username:
Password:
     You have the following disconnected process:
Terminal    Process name   Image name
VTA2:_      VTA2:          (none)
Connect to above listed process [YES]: Return
Connecting to terminal _VTA2:
```

Options

The TELNET client and server support the following options. The client and server negotiate options using the TELNET protocol WILL, WONT, DO, and DONT commands.

ECHO

The ECHO option enables or disables echoing data received over the network. You can configure each side of a connection independently. The initial default is not echoing.

The client supports enabling and disabling of echoing for characters it sends over the network. It does not support echoing for characters it receives over the network. The server supports enabling and disabling of echoing for characters it receives over the network. It refuses any attempts to have the client echo characters that the client receives over the network.

The user's or interactive process terminal sets the OpenVMS terminal device driver TT$M_NOECHO characteristics depending on the echo requirements. Both the client and server attempt to negotiate for the server to echo the characters it receives over the network from the client.

END-OF-RECORD

Use the END-OF-RECORD option during IBM 3270-class terminal emulation. This option affirms that the client and server both use the TELNET end-of-record character. Use this character to delimit TELNET screens.

REMOTE-FLOW-CONTROL

The Remote Flow Control Option (RFC 1372) is supported on both the client and server to enable or disable local flow control on the client, or the handling of **Ctrl+S** and **Ctrl+Q** keystrokes to stop and resume TELNET transmission, respectively. These controls are usually processed locally by the terminal driver and are not sent to the remote server.

The SET LOCAL_FLOW_CONTROL and SET NOLOCAL_FLOW_CONTROL commands are provided on the client if the remote server does not support the flow control option. By specifying SET NOLOCAL_FLOW_CONTROL, the flow control characters are passed to the remote server and are not processed locally.

The default flow control setting depends on the TT$V_TTSYNC value for the terminal. You can set "TTSync" mode (local flow control) outside of TELNET by using the DCL SET TERMINAL /TTSYNC command or set "No TTSync" mode (server flow control) by using the DCL SET TERMINAL /NOTTSYNC command; some full-screen editors also set these modes. However, if you are inside TELNET, SET NOLOCAL_FLOW_CONTROL can force the terminal into "No TTSync" mode for a particular connection.

SUPPRESS-GO-AHEAD

The SUPPRESS-GO-AHEAD option enables or disables sending the TELNET "go-ahead" (GA) control function. You can configure each side of a connection independently. The initial default is not to suppress go-aheads.

Both the client and server support negotiating this option. Regardless of the option's state, both ignore the GA control function if they receive it and never transmit it. The user can send this option by entering the SET GA or SEND commands in the client. Both the client and server attempt to negotiate for GA suppression.

TERMINAL-SPEED

The Terminal Speed Option (RFC 1079) is useful for applications that may want to adjust some actions based on the baud rate at which a user connects. The server does an equivalent of a DCL SET TERMINAL/SPEED command. This is supported on the client and server.

TERMINAL-TYPE

TELNET uses the TERMINAL-TYPE option to negotiate the type of terminal used. TELNET uses this option if both the client and server support the option and are willing to use it. This option tells the TELNET server what type of terminal the client has.

TRANSMIT-BINARY

The TRANSMIT-BINARY option enables or disables the full eight-bit ASCII character set. You can configure each side of a connection independently.

The initial TELNET protocol default is to use the seven-bit ASCII character set. However, the peer TELNET implementation can strip the eighth bit unless you use the TRANSMIT-BINARY option. For the client, use the SET BINARY command to enable full eight-bit mode.

The client and server always transmit the full eight-bit ASCII character set over the network, regardless of the option's status.

When you request the client to enable the TRANSMIT-BINARY option, it sets the OpenVMS terminal device driver TT$M_EIGHTBIT characteristics. When you request the server to enable this option, it does not change the TT$M_EIGHTBIT characteristic.

Depending on the state of the EIGHTBIT terminal option, OpenVMS may strip the eighth bit. OpenVMS strips the eighth bit on input if the terminal attribute is NOEIGHTBIT. However, OpenVMS does not strip this bit on output.

WINDOW-SIZE

The Window Size Option (RFC 1073) is also known as the Negotiate About Window Size (NAWS) option. The client and server negotiate sending the window size information using the standard TELNET WILL/DO/DONT/WONT mechanism. If the client and server agree, the client may send a sub-negotiation to convey the window size. If the client's window size later changes, the client may send a subsequent sub-negotiation. This is supported on the client and server.

Control Functions

The TELNET protocol defines several control functions. Some of these functions include interrupting a process, aborting output, and erasing a character or line.

The client sends these functions with the SEND command, or if the user types characters defined with the SET commands. The client ignores these functions if it receives them.

The server never sends these functions. The server replaces the received functions with an OpenVMS character sequence as shown in the below table. The exception is the "are you there" (AYT) function that returns the BELL character (ASCII 7).

The server recognizes the control functions listed in the below table; it ignores all other control function.

TELNET control function...	**Recognizes OpenVMS character sequence...**
Interrupt Process (IP)	`Ctrl+Y`
Abort Output (AO)	`Ctrl+O`
Erase Character (EC)	`DELETE`
Erase Line (EL)	`Ctrl+U`

Exiting Status

To exit TELNET use the `EXIT` command or type **`Ctrl+Z`**.

TELNET exits with the last error status, if any. DCL command procedures can use the `$STATUS` and `$SEVERITY` symbols to test for success or failure of the TELNET commands issued. A success status indicates that all commands succeeded. A warning, error, or severe status indicates that one or more commands failed to execute, either because of syntax errors or because of operational problems.

When possible, the status code is a System Service (defined in `$SSDEF`), RMS (defined in `$RMSDEF`), or shared (defined in `$SHRDEF`) status. In some cases, status codes are TCPware private codes with a facility number of 1577.

18. Managing TCPware Security

Introduction

This chapter presents the security features available with TCPware.

TCPware security features can be divided into two parts: independent, free-standing security features, and those inherent within products. The free-standing security features include:

- Incoming Access Restrictions
- Outgoing Access Restrictions
- Kerberos Services
- IP Security Option (IPSO)
- Packet Filtering

Security Tips

Here are some general tips to maintain system security on your TCPware system:

1. Implement an aggressive password strategy for users and especially for privileged accounts. This means long passwords and frequent password changes.
2. Disable all unused accounts, especially privileged ones that are predefined with the operating system (such as field test accounts).
3. If a break-in is in progress, use one of the `DEBUG` commands in the Network Control Utility (`NETCU`) to capture the packets. However, be advised that monitoring might be construed as an invasion of privacy.
4. Carefully consider the system announcement or welcome messages. Do not welcome people to the site. Instead, make it clear that the system is for authorized users only, and that all users may be monitored for security reasons.
5. Subscribe to CERT (Computer Emergency Response Team) announcements by e-mail. If you believe that your system was compromised, contact the CERT Coordination Center or your representative in Forum of Incident Response and Security Teams (FIRST). If you wish to send sensitive incident or vulnerability information to CERT staff by e-mail, we strongly advise that

the E-mail be encrypted. The CERT Coordination Center can support a shared DES key, PGP (public key available through anonymous FTP on `info.cert.org`), or PEM (contact CERT staff for details).

6. Disable services not required for your system. For example, if SSH is enabled, disable the services it replaces such as rlogin/rshell and possibly telnet.

Independent Security Features

This section describes the security features that you can use with more than one TCPware product. These include incoming and outgoing access restrictions, packet filtering, and the IP Security Option.

Incoming Access Restrictions

With incoming access restrictions, the system imposes restrictions on the remote hosts having access to local services. Manage incoming access restrictions using the following NETCU commands:

- `ADD ACCESS_LIST`
- `REMOVE ACCESS_LIST`
- `MODIFY SERVICE`
- `ADD SERVICE`
- `SHOW ACCESS_LISTS`

To manage incoming access restrictions, see Chapter 21, *Access Restrictions*.

Outgoing Access Restrictions

With outgoing access restrictions, you restrict access to outside services such as FTP or TELNET for local users. You can base restrictions on a user's ID (UIC or rights identifiers), or the destination address or port (service). Since the system checks only outgoing access restrictions with active open connections and not on each and every datagram, it involves relatively low system overhead.

Manage outgoing access restrictions using two commands in the TCPware Network Control Utility (NETCU):

- `SET OUTGOING_ACCESS_RESTRICTIONS`
- `SHOW OUTGOING_ACCESS_RESTRICTIONS`

To manage outgoing access restrictions, see Chapter 21, *Access Restrictions*.

Packet Filtering

Packet filtering restricts the datagrams that may be received on a network interface. The system drops all datagrams it denies entry. This software-based filtering allows you to filter datagrams by:

- Protocol (IP, TCP, UDP, or ICMP)
- Source or destination address
- Destination port (TCP or UDP) or ICMP message type

Packet filtering prevents a site from receiving datagrams from certain networks or hosts. For example, a site might wish to restrict incoming access from the rest of the Internet, but allow local users to have full access to Internet resources.

Use packet filtering only when absolutely necessary since the system must scan the packet filter for each datagram. If there is a question whether to use packet filtering or access restrictions on incoming datagrams, packet filtering is more complete, since it covers all connections. However, packet filtering requires more overhead than incoming access restrictions.

Manage packet filtering using the following NETCU commands:

- `SET FILTER / SET NOFILTER`
- `SHOW FILTER`

To manage packet filtering, see Chapter 20, *Packet Filtering*.

IP Security Option

The IP Security Option (IPSO) is a standard for preventing a system from receiving or transmitting unauthorized datagrams. It was developed for the U.S. Department of Defense and conforms to RFC 1108, *U.S. Department of Defense Security Options for the Internet Protocol*. IPSO incorporates both a Basic Security Option and an Extended Security Option. TCPware supports both options.

To manage IPSO, see Chapter 24, *IP Security Option (IPSO)*.

Component Security

This section describes the TCPware security features available with particular TCPware products.

R Commands

The R commands implement security at TCPware configuration and later through Service Access Lists and host equivalence files. The R services use standard TCPware and OpenVMS security facilities to

ensure that only authorized hosts and users have access to the TCPware host. TCPware implements R command security through:

Configuration	During TCPware configuration, you first select whether you want to enable the login, shell, and exec services. For login service, you also specify whether you want `NORMAL` or `SECURE` login authorization (the default is `SECURE`). The login service is used for RLOGIN, while shell or exec is used for RCP, RMT, and RSH (exec if you specify a username and password).
Service Access Lists	If desired, create a Service Access List to control which hosts, group of hosts, or network can have access to the service.
Host equivalence files	Host equivalence files allow a set of remote hosts access to the server. They contain single-line entries for each authorization. The system manager can set up a `TCPWARE:HOSTS.EQUIV` file on the system level. Each individual client user can also set up a `SYS$LOGIN:.RHOSTS` file. This allows equivalent access beyond that specified in the `TCPWARE:HOSTS.EQUIV` file. The format for `SYS$LOGIN:.RHOSTS` and `TCPWARE:HOSTS.EQUIV` file entries is identical.

See Chapter 15, *Managing R Commands*, for details on host equivalence files.

The typical R commands server security steps are that the server:

1. Checks that the Service Access List (if any) is configured to protect the desired service.
2. Checks that the client's port number is in a reserved range.
3. Checks the password file for an entry for the supplied username.
4. Searches the `/etc/hosts.equiv` (on UNIX systems) or `TCPWARE:HOSTS.EQUIV` (on OpenVMS systems) file for the hostname and username.
5. Searches the `.rhosts` (on UNIX systems) or `.RHOSTS` (on OpenVMS systems) file in the user's home directory for the hostname and username.
6. Prompts for a password if `SECURE` login is specified (for RLOGIN) and there is a match-up in the `.RHOSTS` or `HOSTS.EQUIV` file, or a username and password if `NORMAL` login is specified (for RLOGIN) without a match-up in the `.RHOSTS` or `HOSTS.EQUIV` file.
7. Grants or rejects access depending on the server configuration and authorization results:
    - Grants access for shell, exec, or `NORMAL` login service without a login prompt if there is a match-up in the `.RHOSTS` or `HOSTS.EQUIV` file, or for `SECURE` login service if the password entered is authorized.

- Rejects access if the server is configured for shell, exec, or SECURE login service and there is no matchup in the .RHOSTS or HOSTS.EQUIV files, or for NORMAL login service if the password entered fails authorization.

For details on Service Access Lists, see the ADD ACCESS_LIST in the *NETCU Command Reference*.

DECwindows

TCPware provides the following security for the DECwindows Transport Interface:

Target display host configuration	Configure the target display host to allow incoming X Window System applications from the OpenVMS system host.
Displaying on the local host	Bring up the Security option under the Session Manager's Options menu.
Displaying on the remote host	Configure "security" on the remote host to allow incoming connections on the currently active session.

To manage DECwindows security, see Chapter 29, *DECwindows Transport Interface*.

FTP

The FTP server provides the following security functions and options:

Functions and Options	**Description**
Passwords	Like DECnet, you cannot use FTP to log in to multiple-passworded accounts.
Directory access restrictions	The FTP server lets you define the TCPWARE_FTP_ROOT logical for directory access restrictions on a system-wide basis, the TCPWARE_FTP_ANONYMOUS_ROOT logical for the ANONYMOUS user directory access restrictions, and the TCPWARE_FTP_*username*_ROOT logical for directory access restrictions for specific user accounts.

Log file	The FTPSERVER_DTP.LOG file contains information about files transferred during the FTP session. Examining this file helps to isolate security problems.
Idle timeout	If the control connection (other than during a data transfer) is idle for more than 10 minutes, the system aborts the connection.
SYLOGIN.COM	This procedure and user account login command procedures can contain commands that cause the login to fail.
FTP ANONYMOUS Accounts	FTP supports ANONYMOUS accounts whereby the user can log in using the ANONYMOUS login and the GUEST password. You set up ANONYMOUS accounts using the AUTHORIZE utility. ANONYMOUS users only have read-only access unless you define the TCPWARE_FTP_ANONYMOUS_RIGHTS logical to allow write, rename, and delete access rights. In addition, the TCPWARE_FTP_ANONYMOUS_ROOT logical allows you to restrict ANONYMOUS users to files in the root directory and its subdirectories.
TCPWARE_FTP_LOGFILE	Define this system logical name to specify the log file name if you suspect break-in attempts through the server. Note that you must restart the FTP server after setting this logical by issuing the @TCPWARE:RESTART FTP command.
TCPWARE_FTP_421_REPLY	This logical defines a message to send when a user you want to prevent from logging in connects to the server. After sending the message, the connection closes.

To manage FTP server security, see Chapter 12, *Managing FTP*.

NFS Server

The NFS server provides several features that maintain the integrity of the OpenVMS file system:

PROXY database	The server requires that the local system must register any user attempting access to OpenVMS files. You do this through the PROXY database when you configure the server, and through later modifications as needed.
EXPORT database	You must export an OpenVMS directory for an NFS user to have access to it. The server does this through the EXPORT database when you configure the server, and through later modifications as needed.

You can take the following additional system security measures:

- Assign an NFS rights identifier to further restrict file access.
- Require all RPC requests to originate from privileged ports.
- Restrict all remote mounts to the NFS superuser only.
- Restrict mounts only to explicit directories and not their subdirectories.
- Require the PROXY database to define the mount requester's identification.

See Chapter 14, *Managing NFS Server*.

Remote Copy Program

Passwords can be a problem in RCP since the `/PASSWORD` qualifier requires entry of plain text that someone on the network can intercept.

You can prevent users from having to specify the `/USER`, `/PASSWORD`, or `/TRUNCATE` qualifier with the RCP command. Check that remote hosts include your hostname entry in their host equivalence files (such as the `/etc/hosts.equiv` file in UNIX systems). On OpenVMS hosts, the `TCPWARE:HOSTS.EQUIV` or `SYS$LOGIN:.RHOSTS` file serves as the host equivalence file to permit remote hosts to log in.

To manage RCP security, see Chapter 16, *Managing R Commands*.

Secure Server

The SSH server provides safe, encrypted access, replacing rlogin, rshell, and telnet. It provides the following security features:

X11 display forwarding	Encrypts the display output for X11 sessions for transmission to another system over an unsecure network.

Port forwarding	Provides transfer of encrypted data between ports on two systems over an unsecure network.
User Authentication	`.rhosts` - Provides simple access via the `rhosts` file. Inherently unsecure and not recommended. `.rhosts` with RSA authentication - the user and system must exist in the `rhosts` file, and the client system's public key must be known to the server system. RSA challenge-response - the client and server system must encode, exchange, and decode successfully a random 32-bit number using the private and public client and server keys. Password - the user is validated via the use of a password.
Multiple Encryption Algorithms	idea, blowfish, twofish, des, 3des, and arcfour.

TELNET

The TELNET Server provides the `TCPWARE_TELNETD_INTRO_MSG` option. With this logical, you can define a special message that appears whenever a user attempts access to the host through TELNET. You can use this logical to issue warnings such as "Authorized Use Only" for remote logins.

Kerberos password protection is available for the TELNET service.

To manage TELNET security, see Chapter 17, *Managing TELNET Server*.

19. Packet Filtering

Using Packet Filtering

Packet filtering is used today in almost all (from basic to sophisticated) security *firewalls*. Packet filtering firewalls apply filtering rules to each packet received to determine whether to accept or discard it. These filtering rules specify the protocol, source and destination IP addresses, and destination ports (for TCP and UDP) for accepted or discarded packets.

You use packet filtering on routers between an internal network and one or more external networks (such as a connection to the Internet). Packet filter rules restrict what may come in over the interface connected to the external network.

Packet filtering can also be useful on hosts. For example, you can restrict the hosts allowed access to services. In particular, these are UDP-based services and services that the TCPware master server does not activate, and thus cannot use incoming access restrictions.

TCPware's packet filtering support is similar to what many routers provide. Once you create a file containing the packet filter rules and load it for an interface, any IP datagrams received on that interface are filtered before being processed or forwarded. (Packets are only forwarded if you enable forwarding; see the `NETCU ENABLE FORWARDING` command for details.)

Note that when you use packet filtering, each and every datagram received on the interface is filtered. This increases processing overhead depending on the size of the filter list.

Packet filtering can be an effective and useful security mechanism. However, it cannot solve all of your security problems. To be effective, you must construct the filtering rules carefully.

Cautions

Consider the following cautions when setting up packet filtering on an interface:

- Packet filtering does not use state information. Each datagram is filtered without any knowledge of packets that preceded it.
- This means that for UDP-based applications, it is not possible to add a rule that says to accept replies to requests. This also affects connection-oriented protocols such as FTP that use two connections, one for commands and the other for data.

- Fragmented datagrams for UDP or TCP are difficult to filter, since only the first fragment has the necessary port information. TCPware solves this problem by applying the filter rules only to the first fragment of UDP and TCP datagrams. The other fragments are accepted and processed or forwarded, but are eventually discarded since they cannot be reassembled without the first fragment. For all other IP protocols, the filter rules apply to each fragment.
- To set up secure packet filtering lists, you need a detailed knowledge of IP, ICMP, TCP, UDP and applications protocols.

Suggested reading includes the protocol RFCs (listed elsewhere in the TCPware documentation) and books such as Cheswick, William R. & Steven M. Bellovin, *Firewalls and Internet Security: Repelling the Wily Hacker*.

Packet Filter File

Packet filtering uses a filter list to determine whether you can receive a datagram. Filter lists are in packet filter files having the `.DAT` extension by default. Create one of these files first and then edit it. Format each file entry in the manner described in the below table.

Field...	**With valid values...**	**Description**
Entry Format: *`action protocol saddr smask [sport] daddr dmask [dport [doption]]`*		

`action`	`permit`
	`drop`
	`deny`
		`Permit` permits the datagram; `deny` denies the datagram and sends the ICMP; `drop` denies the datagram without sending an ICMP destination unreachable message to the sender.

`protocol`	`ip ip-number`
	`tcp`
	`udp`
	`icmp`
		Protocol to check: the values indicated or the numeric IP protocol number. The value `ip` matches any IP protocol.

`saddr`	Example: `192.168.123.123`	Source IP address to check.

smask	Example: `255.255.255.255`	Source address mask to check, in standard bit mask format. To match a single address, use `255.255.255.255`. To match any address, use `0.0.0.0`

sport	operator operand
	
	`lt port`
	`le`
	`eq`
	`ge`
	`gt`
	`ne`
		Optional source port specification to check (for TCP and UDP entries only). Consists of an operator, space, and port name or number. If port name, must be defined in the `TCPWARE:SERVICES.` file. If omitted, any source port is valid.
		
		Example: `gt 1023`

daddr	Example: `192.168.123.123`	Destination IP address to check.
dmask	Example: `255.255.255.255`	Destination address mask to check, in standard bit mask format, as in `s-mask` above.

dport	operator operand
	
	`lt` *port*
	`le` *icmp-msg-type*
	`eq`
	`ge`
	`gt`
	`ne`
		Optional destination port (for TCP and UDP entries) or ICMP message type specification consisting of an operator, space, and operand. If a port name, must be in the `TCPWARE:SERVICES.` file. If *icmp-msg-type*, use:
		
		0 Echo Reply
		3 Destination Unreachable
		4 Source Quench
		5 Redirect
		8 Echo
		11 Time Exceeded
		12 Parameter Problem
		13 Timestamp
		14 Timestamp Reply
		15 Information Request
		16 Information Reply

doption	established	Matches only established connections (TCP segments with ACK or RST bits set).
start time		Absolute of delta VMS-format date/time. If specified, the filter takes effect starting at the time specified. By default, a filter takes effect when loaded by TCPware if the start time is not defined.
end time		Absolute or delta VMS-format date/time. If specified, the filter is ignored after the time specified is reached if an absolute time is specified, or after the time calculated by adding the end time to the start time if a delta time is specified. If no start time was specified, the delta time is added to the current time. If the end time for a non-repeating filter has already passed when the filter definition is parsed, the filter is not loaded and the user is informed of that fact.
repeat		If Y and start/end times are specified, this filter repeats until it is removed. Both a start and end time must be specified in order to specify a repeating filter.
log		Logs events from this filter.

Each entry specifies a packet filtering condition for a particular protocol type, source or destination address and mask, and destination port and mask specification, with certain additional options. The system looks at each condition in sequence, looks for a match, and takes a permit (accept) or deny (reject) action. The system stops testing conditions after the first match. This means that the order of the entries in the file is important; if the file lists a subsequent condition for an address, the system ignores it.

An implicit deny terminates the list of entries in the packet filter file. This means that if no condition matches, the system rejects the datagram. To use packet filtering:

1. Create address list entries in the packet filter file.
2. Apply the list to interfaces on your system by using packet filtering commands.

To create a packet filter file, edit a file and add address list entries in the format described.

Any number of spaces or tabs can separate each entity. Lines beginning with an exclamation point (`!`) are comment lines. You can use the dash (−) continuation character at the end of a line that needs to continue onto the next.

To apply the list to a particular network interface or interfaces on your system, use the `SET FILTER` command.

Filtering by Time

Filters may be set to be activated only during a specified time period. These filters may be specified as a one-time filter or as a filter that repeats. For example, a filter may be set up to filter all traffic from a specific address during the hours of 5am to 5pm each day, or a filter may be specified that filters traffic starting from the time the filter is loaded and for the next 3 hours.

Time-based filtering is done by specifying a start time, an end time, or both start and end times for a filter in the filter definition file. For repeating filters, both start and end times must be specified. Note that all time values for start and end times must be specified in VMS absolute or delta time format. For example, the following are all valid:

“29-DEC-2014”	would be interpreted as "29-DEC-2014 <current time>
“29-DEC-2014 18:03”	would be interpreted as "29-DEC-2014 18:03:00.00"
18:03:00	would be interpreted as "<current date> 18:03:00.00"
“1 15:00”	would be interpreted as a delta time of 1 day, 15 hours

Note that if an absolute time is specified that contains both a date and time (example 2 above), it MUST be enclosed by double quotes. For example:

```
deny icmp 0 0 0 0 eq 5 start 17:00:00 end "29-Sep-2014 6:00:00"
```

Given the following filter file (the dashes at the end of some lines are for readability only, and are not valid):

```
deny tcp 15.1.94.2 15.1.94.255 2.22.2.5 255.255.255.255 start 15:20 end 18:30 repeat
deny tcp 1.1.94.2 1.1.94.255 207.225.29.51 255.255.255.255 end
"1-JAN-2014 18:30" deny tcp 195.101.94.209 195.101.94.255 207.225.29.51 255.255.255.255 start
   18:00 end "1 00:30" repeat
deny tcp 195.101.94.209 195.101.94.255 207.225.29.51 255.255.255.255
deny tcp 195.101.94.209 195.101.94.255 207.225.29.51 255.255.255.255 start 17:00 end 18:30
deny tcp 15.1.94.2 15.1.94.255 2.22.2.5 255.255.255.255 start "2 00:00" end 3 00:00"
```

Line 1 will filter from 15:20 to 18:30 each day.
Line 2 will filter from the time the filter is loaded through 18:30 on January 1, 2014
with no logging. After that time, if the filters are reloaded, this filter will not be loaded.
Line 3 will filter from 18:00 to 19:30 each day.
Line 4 has no time limits on it.
Line 5 will log from 17:00 through 18:30 today.
Line 6 will filter starting 2 days from the time the filter is loaded, through 3 days after that.

Filter Logging

Filter "hits" may be logged, either to OPCOM or to a file defined by the user. Logging is enabled on a filter-by-filter basis, by using the `log` keyword on the end of a filter definition line. For example:

```
deny tcp 192.10.9.209 192.10.9.255 207.225.29.51 255.255.255.255 log
```

Logging for the interface is controlled via the `NETCU SET FILTER` command. The actual logging is performed by the `TCPware_FLOG` process, which is started the first time a `NETCU SET FILTER` *`interface filter_file`*`/LOG` command is issued (a single `TCPware_FLOG` process handles logging for all interfaces defined on the system).

The `NETCU SET FILTER` command switches used to support logging are

Qualifier	Values	Default	Description
`/[NO]LOG`	OPCOM or a valid filename	None	Used to turn logging on or off. Filter events may be logged to OPCOM or to a specified file. Only those events with the `LOG` qualifier in their definition are affected by this qualifier.

/FORMAT	COMMA or NORMAL	NORMAL	If NORMAL, then the normal formatting as seen by NETCU SHOW FILTER will be used. If COMMA, then a comma-delimited file will be created that can be, for example, loaded into a spreadsheet.
/INTERVAL	Number of seconds, between 5 and 2^{31}	5 seconds	Reporting interval. The minimum reporting interval is 5 seconds, so that a flood of filter events doesn't drag the system down. When reporting events, a count of missed events will be included for each event where the event couldn't be reported before the next event occurred.

When filter logging is enabled, the TCPware_FLOG process will be started. This process checks each interface at the interval defined by the /INTERVAL qualifier for the NETCU SET FILTER command. As unlogged filter hits are found, it will log them to OPCOM or to a file, according on the parameters set by the /LOG and /FORMAT qualifiers for the NETCU SET FILTER command.

When logging to OPCOM, only NORMAL formatting is allowed. An OPCOM message, formatted as the filter output from NETCU SHOW FILTER, will be displayed for each filter with unlogged hits on it.

When logging to a file, the output will be identical to that of the filter displays from NETCU SHOW FILTER command, if /FORMAT=NORMAL is specified. If /FORMAT=COMMA is specified, the data will be recorded as comma-delimited fields, one line per filter, to the file. The first line of this file will contain the field names (comma-delimited) to aid in interpreting the contents of the file.

Examples:

```
$ NETCU SET FILTER EWA-0 FILTERS.DAT /LOG=OPCOM/INTERVAL=10
```

enables logging to OPCOM, with a reporting interval of 10 seconds.

```
$ NETCU SET FILTER EWA-0 FILTERS.DAT/LOG=FOO.DAT/FORMAT=COMMA
```

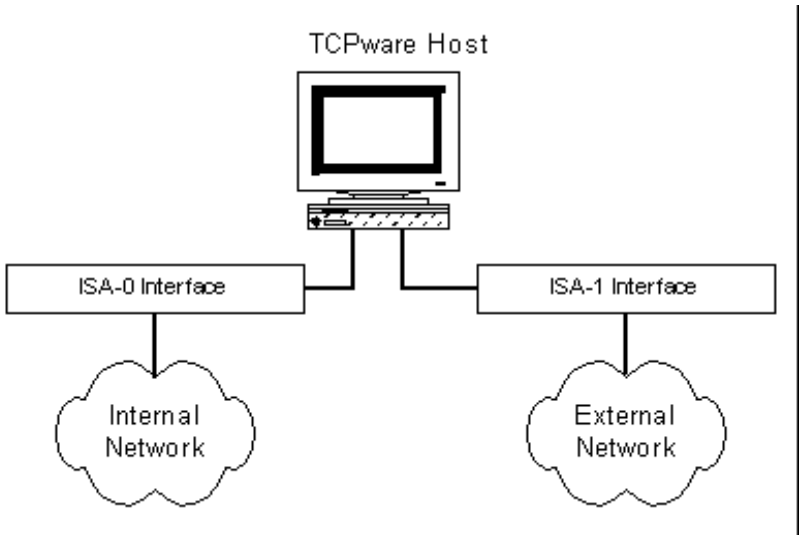
enables logging to the file FOO.DAT in comma-delimited format, and a reporting interval of 5 seconds (the default).

```
$ NETCU SET FILTER EWA-0 /NOLOG
```

This disables all logging for the interface, closing all open log files.

Configuration Recommendations

A packet filtering configuration might look something like the below figure. A packet filter is installed at the ISA-1 interface to prevent unwanted packets from the external networks from entering the internal network. TCPware's packet filters are applied to packets received on a specific interface (no filtering of transmitted packets is done).



Constructing the address filter list requires care in that you only want to let in the packets you need. Here are some recommendations in setting up an address filter list for an interface:

- Add an entry to prevent IP spoofing (that is, having an external host send a datagram as if it came from a local machine). For a router, this makes sense since no datagram received from an external network should ever have a local source address. Add the following entry to the filter list for the external interface (ISA-1 in the above figure):

  ```
  deny ip local-network local-network-mask
  ```

- Be careful with services that use "unprotected" port numbers (greater than 1024). Some examples are NFS (port 2049) and X Windows (port 6000 and higher). Explicitly denying these services is a good idea:

  ```
  deny udp 0 0 0 0 eq 2049
  deny tcp 0 0 0 0 eq 2049
  deny tcp 0 0 0 0 eq 6000
  deny tcp 0 0 0 0 eq 6001
  ```

- Prevent broadcast and loopback packets from entering your network. It is best to restrict the broadcast (the first two of the following entries) to an external interface; apply the loopback restriction (the last entry) to any interface:

```
drop ip 0.0.0.0 255.255.255.255
drop ip 255.255.255.255 255.255.255.255
drop ip 127.0.0.0 255.0.0.0
```

- Guard against datagrams from invalid source addresses when connected to the Internet (provided you are not using these network numbers for internal-only traffic purposes). Add the following to the filter list for the external interface (ISA-1 in the above figure):

```
drop ip 10.0.0.0 255.0.0.0
drop ip 172.16.0.0 255.240.0.0
drop ip 192.168.0.0 255.255.0.0
```

- You generally need to allow in domain name (DNS) requests using:

```
permit udp 0 0 0 0 eq 53
```

  Whether to allow TCP DNS traffic (usually used for zone transfers) is also something to consider. To disallow TCP DNS traffic, add:

```
deny tcp 0 0 0 0 eq 53
```

- You should not be concerned with what services local users use in the external world. You would want to add:

```
permit tcp 0 0 0 0 gt 1023 established
```

  This allows all TCP datagrams in for ports greater than 1023 that have either the ACK or RST bits set in the TCP flags. Connection establishment requests have just the SYN bit set, so they are not allowed in by this entry.

  You may want to drop the `established` option if you want to allow incoming connections to unprotected ports. This would allow use of the FTP PASV capability.

- You may offer services to the external world such as a World Wide Web or anonymous FTP server. Add the following entries:

```
permit tcp 0 0 (web-server) 255.255.255.255 eq 80
permit tcp 0 0 (ftp-server) 255.255.255.255 eq 21
```

  If you have several hosts for each service, add an entry for each. Note that for the FTP Server, the data connections are normally outgoing and thus the earlier `permit tcp 0 0 0 0 gt 1023 established` configuration works to allow these. However, if users switch to PASV mode, the connections will be incoming (to unprotected port numbers) and therefore the `permit tcp 0 0 0 0 gt 1023` configuration (without the `established` option) may be more effective.

- Allow all ICMPs except ICMP redirects in:

```
deny icmp 0 0 0 0 eq 5
permit icmp
```

This is useful for informing hosts about problems. But it can open up denial of service attacks, especially if hosts are not careful about the ICMP redirects that they accept. That is why discarding them is recommended.

- Watch the order of the entries in the table carefully.

```
permit tcp 0 0 0 0 gt 1023
deny tcp 0 0 0 0 eq 2049
```

This entry would not work since the permit entry allows the datagram in and processing stops as soon as a match is found. TCPware processes the entries in the order in which you specify them.

- Remember that an implicit "deny everything" is added to the end of the filtering list. This means that to permit a datagram in, you need to have a permit entry in the list.
- Once you applied your filter list, test it first. Get an account on a host on an outside network that you can use to connect to your local hosts. Check that you are not allowing any access you do not want, and that you are allowing access that you do want. If something is not right, modify the filter list, reload it, and retest.

While packet filtering is very useful, it is by no means the only step you should take to secure your network. You must take special care to secure the system to assure that it cannot be compromised. One way to do this is to greatly limit the services it offers.

Setting and Showing

The commands to set and show packet filters are:

`SET FILTER`	Associates a packet filter list with particular network lines
`SET NOFILTER`	Removes a previously associated packet filter list from network line(s)
`SHOW FILTER`	Displays the current packet filter list for network lines

See the *NETCU Command Reference* manual for details on these commands.

Setting at Startup

When you start TCPware, the STARTNET procedure looks for a `TCPWARE:FILTER-`*`line-id`*`.DAT` file for each interface it starts. If the file exists, STARTNET issues the following NETCU command to set the filter list for the interface:

```
SET FILTER line-id TCPWARE:FILTER-line-id.DAT
```

You can also add the necessary NETCU commands to the `TCPWARE:ROUTING.COM` file.

20. Access Restrictions

Introduction

This chapter describes how to manage incoming access restrictions:

- Incoming access restrictions are used to restrict the hosts (or groups of hosts) allowed access to TCP-based services that the TCPware master server initiates. You can use incoming access restrictions to help secure your host by restricting which hosts have access to its services.
- Outgoing access restrictions apply to outgoing TCP connections made from the TCPware host. You can use them to limit the TCP-based services that TCPware host users can use on other systems. You can base these restrictions on user rights identifiers, or destination IP addresses or ports.

You can also use outgoing access restrictions to log or alarm outgoing connections. Log means that a record is written to the `TCPWARE:NETCP.LOG` file, which identifies the user and the destination of the connection. Alarm means that this same message is sent to OPCOM. Even if you do not restrict the connections users can make, you might want to log or alarm all attempts.

If the process attempting to open a connection has SYSPRV or BYPASS privilege or is running under a system UIC, outgoing access restrictions are not applied. However, logging and alarming entries are still honored.

Incoming Access Restrictions

Permit or deny incoming access using access lists controlled by commands in the Network Control Utility (NETCU), as follows:

```
$ NETCU [command]
```

The commands are:

- `ADD ACCESS_LIST`
- `ADD SERVICE`
- `MODIFY SERVICE`
- `REMOVE ACCESS_LIST`
- `SHOW ACCESS_LISTS`

See the *NETCU Command Reference* for details on each command.

You first set up an access list using the ADD ACCESS_LIST command to specify the hosts or groups of hosts you want to have explicit use of the service. Access lists should be defined in SERVERS.COM. You then define (or modify) a service using the ADD SERVICE (or MODIFY SERVICE) command and use the /ACCESS_LIST qualifier to indicate the access list number. If a host's internet address is not in the permitted list, it cannot use the service. Any connections the host makes to that service are immediately closed. The special /ACCESS_LIST=0 entry means that all users are allowed access to that service.

An implicit DENY is added to the end of each access list, meaning that you need an explicit permit to allow access. Also, an empty access list denies all users access to the service.

Subnet Masks

Consider the ADD ACCESS_LIST command syntax:

```
$ NETCU ADD ACCESS_LIST list condition ia mask
```

The *list* is the access list number, *condition* is either DENY or PERMIT, *ia* is the internet address, and *mask* is the subnet mask. The *mask* is the most significant field used to determine the order of entries in the file, followed by DENY before PERMIT conditions, followed by the address.

Often in entries without masks, all DENY entries are listed before all PERMIT entries. If you do use masks, this order can change based on your intent. Consider the following sequence of commands where you want to deny access to a class C network yet permit access for a particular host on that network:

```
$ NETCU ADD ACCESS_LIST 1 DENY 192.168.142.0 255.255.255.0 -
_$ /MESSAGE="Sorry, access denied."
$ NETCU ADD ACCESS_LIST 1 PERMIT 192.168.142.25
$ SHOW ACCESS_LISTS
List Condition Internet Address Address Mask    Access Denied Message
---- --------- ---------------- ------------    ---------------------
1    PERMIT    192.168.142.25   255.255.255.255 "Sorry, access denied."
     DENY      192.168.142.0    255.255.255.0
```

The longer mask (255.255.255.255) is more specific about the entry and has priority in the file. That is why the PERMIT entry appears before the DENY entry, so that host 192.168.142.25 has access while other hosts on its class C network do not.

Note that the Access Denied Message you can add using the /MESSAGE qualifier applies to DENY entries for a particular list. The message appears next to the first list entry in a SHOW ACCESS_LISTS display, even if it is a PERMIT entry as in the previous example.

Examples

The following example restricts SMTP server access to hosts only on the local network.

```
$ NETCU ADD ACCESS_LIST 21 PERMIT local-network local-network-mask
$ NETCU MODIFY SERVICE 21 TCP/ACCESS_LIST=21
```

The following example uses an empty access list to make a service temporarily unavailable:

```
$ NETCU REMOVE ACCESS_LIST 1
$ NETCU MODIFY SERVICE SMTP TCP/ACCESS_LIST=1
```

To make the service available again later, use:

```
$ NETCU MODIFY SERVICE SMTP TCP/ACCESS_LIST=0
```

The 0 access list is special and means all users are allowed access.

Outgoing Access Restrictions

To use outgoing access restrictions, you create a file that contains the list of restrictions. Once you create this file, load it using the `SET OUTGOING_ACCESS_RESTRICTIONS` command in the Network Control Utility (NETCU). Once loaded, TCPware checks the outgoing access restrictions list whenever an active, outgoing connection attempt is made. If the outgoing access restrictions list permits the access, the connection attempt is allowed. If not, the connection attempt fails with a privilege violation status.

When building an outgoing access restrictions list, be sure to add permit entries for all permitted activity, since an implicit "deny everything else" is always at the end of the list. Therefore, you need an explicit permit entry to allow access.

The format of outgoing access restrictions list entries is as follows:

```
action ID [destination [mask] [operator port]]
```

The below table describes the fields in each entry.

Field...	With valid values...	Description
action	permit deny	Keyword or keyword list (separated by commas; no spaces in between). Do not use `PERMIT` together with `DENY`.

	`log`
	
	`alarm`
		Permits access.
		
		Denies access.
		
		Logs the access attempt in the `NETCP.LOG` file.
		
		Generates an OPCOM message on the access attempt.

`ID`	username or UIC rights-ID *	Local user's username or UIC, such as `SMITH`, `[10,100]`, or `[SMITH]`. Local user's rights identifier, such as `ACCOUNTING`. Wildcard used by itself (to mean "anyone") or as in `[*,100]` or `[10,*]`.
`destination`	Example: `192.168.95.126`	Destination IP address to check, in dotted-decimal format. If omitted, indicates all destination addresses.
`mask`	Example: `255.255.255.0` (This example sets only the first three bytes in the destination address as significant. This restricts access to a class C network.)	Destination address mask to check, in standard bit mask format. Each bit set indicates that the corresponding bit in the destination field is significant. If omitted, TCPware determines the mask based on whether the host portion is 0; if 0, TCPware uses the network or subnet mask; if not 0, TCPware uses the mask 255.255.255.255 (meaning all bits are significant).

`operator`	`lt`
	`le`
	`eq`
	`ge`
	`gt`
		Use with the port field:
		
		Less than
		Less than or equal to

		Equal to Greater than or equal to Greater than
port	port-number port-name	Destination port specification of either the port number (such as 2049 for the NFS port) or the port name (such as SMTP). If a name, must be defined it in the `TCPWARE:SERVICES.` file.

The order of the entries in the file is important. TCPware looks at each condition in sequence and stops testing after the first match that involves a permit or deny, ignoring any subsequent condition for an address. To use outgoing access restrictions:

1. Create outgoing access restrictions list entries in a file.
2. Apply the file entries to your system using commands in NETCU.

To create an outgoing access restriction file, edit a file and add entries in the format described. For example, you can set the following outgoing access criteria:

```
! Log all connections
log * 0.0.0.0 0.0.0.0
! Deny access to SMTP applications over the network
deny * 0.0.0.0 0.0.0.0 eq 25
! Permit all connections to the INTERNET_USER rights identifier only
permit internet_user 0.0.0.0 0.0.0.0
```

Any number of spaces or tabs can separate each entity. Lines beginning with an exclamation point (`!`) are comment lines. Use the dash (`-`) continuation character at the end of a line continuing onto the next.

An implicit `deny *` (deny all) is the last entry in the file. This means that an explicit `permit` entry must exist to allow any outgoing access. However, `permit` access always exists for privileged users (with SYSPRV or BYPASS privilege or running from a system UIC).

In addition, facilities that use a TCPware internal interface are also considered privileged users; for example, any application that uses the PWIPDRIVER interface (such as the PATHWORKS v5 server and the DECnet/OSI product from HP).

Once you load the file using `SET OUTGOING_ACCESS_RESTRICTIONS`, the outgoing access restrictions indicated in the file go into effect. You can use a subsequent `SHOW OUTGOING_ACCESS_RESTRICTIONS` command to display the results:

```
$ NETCU SET OUTGOING_ACCESS_RESTRICTIONS HOMERRESTRICT.DAT
$ NETCU SHOW OUTGOING_ACCESS_RESTRICTIONS
TCPware(R) for OpenVMS Outgoing Access Restrictions List
Actions  Userid        Destination Address Destination Mask  Port
-------  ------        ------------------- ----------------  ----
LOG      *             0.0.0.0             0.0.0.0
DENY     *             0.0.0.0             0.0.0.0           EQ 25
PERMIT   INTERNET_USER 0.0.0.0             0.0.0.0
```

Setting and Showing

Use the outgoing access restrictions commands on the NETCU level. The commands are:

- `SET OUTGOING_ACCESS_RESTRICTIONS`
- `SET NOOUTGOING_ACCESS_RESTRICTIONS`
- `SHOW OUTGOING_ACCESS_RESTRICTIONS`

All commands require OPER privilege. The `SET OUTGOING_ACCESS_RESTRICTIONS` command only affects the running system. To load the outgoing access restrictions each time you start TCPware, you can:

1. Place outgoing access restrictions in the `TCPWARE:TCPWARE_OUTGOING_RESTRICTIONS.DAT` file (STARTNET loads this file if it exists).
2. Add a `SET OUTGOING_ACCESS_RESTRICTIONS` command to the `TCPWARE:ROUTING.COM` file.

See the *NETCU Command Reference* for details about these commands.

> **Caution!** Avoid using the `SET OUTGOING_ACCESS_RESTRICTIONS NLA0:` command. You can deny all outgoing access.

Examples

Examples of how you might use outgoing access restrictions include:

- For service bureaus or sites that want to limit which users are allowed access to TCP-based services, you might consider creating an `INTERNET_USER` rights identifier, and granting it to those users that purchase or need TCP/IP access. Then, you would create a very simple outgoing access restriction list such as:

```
permit INTERNET_USER
```

This list would permit all users with the `INTERNET_USER` rights identifier to establish TCP connections. Anyone without that rights identifier would not be allowed to establish TCP connections.

Note that users can still send SMTP mail using TCPware's SMTP mailer. To disable use of this service, you would need to add ACLs to the `TCPWARE:SMTP_MAILSHR.EXE` file to allow only users with the `INTERNET_USER` rights identifier access.

- If you have problems with TELNET users accessing the SMTP port and creating fake mail messages, prevent this by using an outgoing access restriction list:

```
deny * 0 0 eq smtp
permit *
```

Note that users can still send and receive SMTP mail using SMTP-OpenVMS (which runs with privileges that allow it to bypass the outgoing access restrictions list).

- If you want to log all outgoing connections:

```
permit,log *
```

To alarm all connections, use:

```
permit,alarm *
```

You can also log or alarm selected outgoing connections. For example, to log all outgoing SMTP connections:

```
permit,log * 0 0 eq smtp
permit *
```

Note that the second entry (`permit *`) permits all other outgoing connections.

21. Managing Kerberos

CAUTION! This chapter used to document TCPware's Kerberos V4 server, which has been deprecated and is no longer available.

22. IP Security Option

IPSO Security

IPSO is an option included in the header of a datagram that specifies the datagram's level of security. This option is a *label*. A system can check for the existence of an IPSO label in an incoming datagram and determine whether the datagram matches its specific security requirements. Likewise, a system can screen outbound datagrams based on their labels.

For an IPSO-supported system, if a datagram does not pass a label test, the system considers it out of range and rejects (discards) it. The system then sends an ICMP error message (if enabled) describing the condition to the datagram's originator. The system manager must decide:

The consequences of using IPSO security	IPSO has a far reaching effect on all network applications running over IP.
The IPSO security option the system requires	The system manager can decide to implement the Basic Security Option or Extended Security Option.
The security level (or levels) for incoming or outgoing datagrams	Top Secret, Secret, Confidential, or Unclassified.
The protection authority (or combination of authorities) for incoming or outgoing datagrams	GENSER, SIOP-ESI, SCI, NSA, or DOE.
How to set IPSO on a system-wide and individual port (line) basis.	
What to do with unlabeled datagrams	The system manager can either reject such datagrams or implicitly label them so that they pass IPSO screening tests.
How to set security on ICMP error messages generated by out-of-range datagrams	In some cases, you must set IPSO on these messages so that the system does not reject them; in other cases, the system manager may want these messages to be "invisible."

Consequences

IPSO is for physically secure TCP/IP networks run by trusted system managers with the highest security clearance. You should segregate IPSO networks from non-IPSO networks. IPSO-protected datagrams should never enter a non-IPSO network.

IPSO affects everything in a TCP/IP-based network. For example, TELNET connections are impossible if receiving systems reject IPSO-labeled datagrams. This also affects FTP, NFS, FINGER, Line Printer Services, RSH, RCP, RLOGIN, among others.

IPSO also affects other protocols on which these applications may depend, such as the DNS, Kerberos, NTP, TIMED, RIP, and RAP. If a host cannot communicate with a DNS server, for example, all host information required to run an application becomes unavailable. In such cases, system managers must set all possible DNS servers with the same IPSO security.

Basic and Extended Security Options

IPSO incorporates a Basic Security Option and an Extended Security Option. Each IPSO-labeled datagram has in its IP header authorization information related to these options.

The Basic Security Option is well-documented, standardized, and includes the security levels and protection authorities described in the next subsection.

RFC 1108 minimally mentions the Extended Security Options, but they need more definition. Security managers can use extended options as custom, site-specific protocols they do not want to reveal. These options occupy extra space in the IP header after the basic option. You can only use them together with an existing basic option. TCPware supports the extended options in allowing or disallowing extended options in datagrams.

Security Levels and Protection Authorities

An IPSO security label consists of a classification level and a protection authority. The classification levels appear in the below table with their hexadecimal values. The protection authorities appear in the following table with their hex values and points of contact.

Security level	Has hexadecimal value
`Top_Secret`	`%X3D`
`Secret`	`%X5A`

Confidential	%X96
Unclassified	%XAB

Protection authority	Has hex value	With point of contact
GENSER	%X80	Designated Approving Authority per DOD 5200.28
SIOP-ESI	%X40	DoD Joint Chiefs of Staff
SCI	%X20	Director of Central Intelligence
NSA	%X10	National Security Agency
DOE	%X08	Department of Energy

Labeling as Opposed to Screening a Datagram

There are two distinctions in IPSO:

- Labeling an outgoing datagram
- Screening datagrams

Some datagrams already have IPSO labels when they reach your system, for example, by socket calls or other labeling methods. You can label "raw" datagrams by adding an implicit transmit label onto the datagram using a TCPware command. (You can also redefine already labeled datagrams in the same way.)

Screening a datagram involves checking whether to allow the labeled datagram in or out. You use TCPware IPSO commands for this purpose. You can screen on a system-wide and line-by-line basis, as discussed in the next section.

System and Line Basis Protection

You can set IP Security Options on a system-wide and line-by-line basis.

- The system requirements determine whether a datagram can enter or leave the system.
- The line requirements determine whether a datagram can enter or leave a specific port (line) on the system.

Datagrams destined for your system or the ones your system generates undergo system level IPSO checks. Datagrams just passing through (being forwarded) undergo only line checks on the port they use. All datagrams undergo line checks.

You can restrict types of security to certain lines or make security requirements for forwarded datagrams different than for datagrams destined for the system. You can also use system requirements to cover lines not specifically set. To be effective, system requirements should be supersets of all individual line requirements.

Both system and line settings consist of incoming and outgoing requirements. The below example shows how to use a `SET IPSO /SYSTEM` command so that both incoming and outgoing requirements are for a security level of `UNCLASSIFIED` and a protection authority of `NONE`. These are the default settings if you do not specify a level or authority. (Note that the levels show up as ranges of values with the `SHOW IPSO` command.)

```
NETCU>SET IPSO /SYSTEM
NETCU>SHOW IPSO
Tcpware(R) for OpenVMS IPSO Configuration for System:
Label       Level                          Authorities
-----       -----                          -----------
In:         UNCLASSIFIED to UNCLASSIFIED   None
Out:        UNCLASSIFIED to UNCLASSIFIED   None

Implied
  Receive:  UNCLASSIFIED                   None
  Transmit: UNCLASSIFIED                   None
ICMP Error: UNCLASSIFIED                   None
```

To set the incoming and outgoing requirements separately, you could specify a command such as in the below example. The incoming label now requires an `UNCLASSIFIED` to `SECRET` level range on a system basis. All the other values remain the same.

```
NETCU>SET IPSO /SYSTEM /IN_LABEL=(LEVEL=(UNCLASSIFIED,SECRET))
NETCU>SHOW IPSO
Tcpware(R) for OpenVMS IPSO Configuration for System:
Label       Level                          Authorities
-----       -----                          -----------
In:         UNCLASSIFIED to SECRET         None
Out:        UNCLASSIFIED to UNCLASSIFIED   None

Implied
  Receive:  UNCLASSIFIED                   None
  Transmit: UNCLASSIFIED                   None
ICMP Error: UNCLASSIFIED                   None
```

If you want the line to accept just SECRET incoming datagrams, you could use a command such as in the below example. Again, only the incoming line requirement changes.

```
NETCU>SET IPSO /LINE=SVA-0 /IN_LABEL=(LEVEL=SECRET
NETCU>SET IPSO /LINE=SVA-0 /RECEIVE_IMPLICIT_LABEL=(LEVEL=SECRET)
NETCU>SHOW IPSO
Tcpware(R) for OpenVMS IPSO Configuration for line SVA-0:
Label       Level                          Authorities
-----       -----                          -----------
In:         SECRET to SECRET               None
Out:        UNCLASSIFIED to UNCLASSIFIED   None

Implied
  Receive:  SECRET                         None
  Transmit: UNCLASSIFIED                   None
ICMP Error: UNCLASSIFIED                   None
```

You can also use `SET NOIPSO` to disable IPSO.

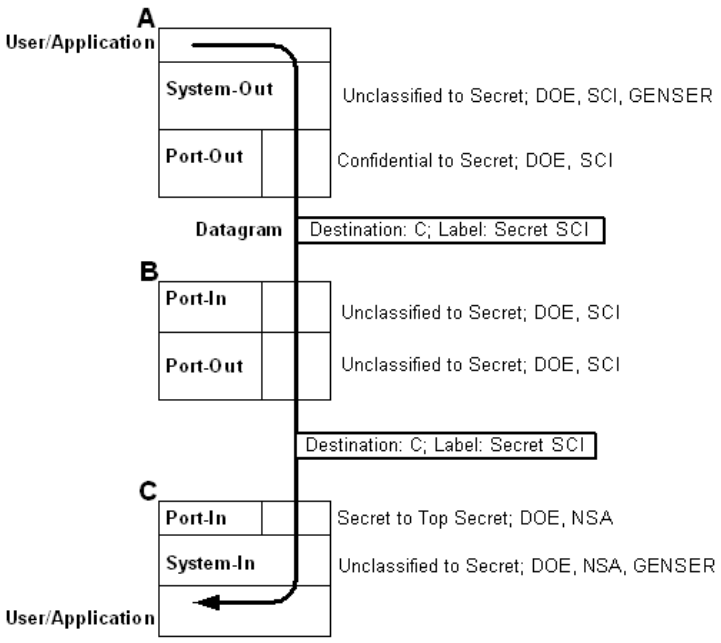
Unlabeled Datagrams

The example in the previous section also shows what to do about unlabeled datagrams. The `/RECEIVE_IMPLICIT_LABEL` qualifier used on the second line ensures that the system accepts unlabeled received datagrams as if they were SECRET (see the `Implied... Receive:` output). If the received implicit label had been left at `UNCLASSIFIED`, unlabeled datagrams would not pass the incoming test.

Frequently, systems do not allow implicit labeling, to prevent unlabeled datagrams from entering (or passing through) a system.

Sample Implementation

The below example shows a sample IPSO implementation.



Host A routes a datagram through host B to get to its destination, host C. The datagram has the label "SECRET SCI." This label passes A's outgoing system and port checks, as indicated. Since B only routes the datagram, it performs only port checks on the datagram. B then forwards the datagram to C. The datagram passes C's incoming port and system checks and successfully reaches the user or application.

The implementation assumes a cooperative network scenario. In this scenario, system managers on hosts A, B, and C communicate with each other on setting security options. Even intermediate hosts (like B) need to be aware of the nature of the secure traffic routed through. Otherwise, such a system could lose datagrams.

Other configurations may not set up system checks, and therefore, not perform them. If an IPSO check fails, the datagram is out of range and the system sends an ICMP error message back to the originator. However, the ICMP message itself must have the appropriate label so that it can pass the system and port checks for an outgoing datagram.

Commands

You add IPSO labels to datagrams, and set system and line requirements for IPSO screening using commands in TCPware's Network Control Utility (NETCU). These commands are:

- `SET IPSO`
- `SHOW IPSO`

To invoke NETCU, enter:

```
$ NETCU
```

and enter the series of IPSO commands you need to configure the system or lines at the NETCU> prompts.

See Chapter 2, *NETCU Commands*, in the *NETCU Command Reference* for detailed descriptions of these commands.

You can also include each command in your TCPWARE:IPSO_SETUP.COM file.

The basic format of the NETCU command that sets IPSO screening is:

```
SET IPSO {/SYSTEM | /LINE} /LABEL-TYPE=(LEVEL=level, AUTHORITY=auth)
```

/SYSTEM or /LINE=*line*	is the system-wide setting or the line number (such as SVA-0).
/LABEL-TYPE	is one of the following: /LABEL - label for both received (incoming) and transmitted (outgoing) datagrams /IN_LABEL - label for incoming datagrams only /OUT_LABEL - label for outgoing datagrams only
level	is either a single security level (*level*), or a range (minimum and maximum) security level in the form (*min-level, max-level*). You can use either the name of the security level or its hexadecimal value as shown in *IPSO Security Levels*; for example, Top_Secret or %X3D. In specifying the range of levels, the *min-level* must be lower on the list in *IPSO Security Levels* than the *max-level*.
auth	is either a single protection authority or a list of authorities in the form (*auth1, auth2, ..., authn*). You can use either the name of the authority or its hexadecimal value as shown in *IPSO Protection Authorities*; for example, GENSER or 0x80. Decimal and octal values are also acceptable.

	Each *authn* can also be in the form "*auth1+auth2+...+authn*" (a combination of authorities strung together with plus signs and enclosed in quotes). An example is "GENSER+SCI" - this means that you can process only datagrams with the *combination* GENSER+SCI. Alternatively, you can use the site-specific name set up in the IPSO_AUTHORITIES. file; for example, ALPHA for "SCI+NSA". (See *Site-Specific Authority Names*.)

Suppose you want to set up line SVA-0 to accept all incoming or outgoing datagrams with a security level ranging from UNCLASSIFIED to SECRET. You also want the line to accept a protection authority of either GENSER or a combination of SCI+NSA. Perform the following command at the NETCU prompt:

```
NETCU>SET IPSO /LINE=SVA-0 -
_NETCU>/LABEL=(LEVEL=(UNCLASSIFIED,SECRET), -
_NETCU>AUTHORITY=(GENSER,"SCI+NSA"))
```

Adding an IPSO Label

Add explicit IPSO labels to transmitted datagrams using the /TRANSMIT_IMPLICIT_LABEL qualifier together with the /ADD qualifier, as in:

```
NETCU>SET IPSO /LINE=SVA-0 -
_NETCU>/TRANSMIT_IMPLICIT_LABEL=(LEVEL=SECRET, AUTHORITY=DOE, ADD)
NETCU>SHOW IPSO
%TCPWARE_NETCU-I-NOIPSODATA, no IPSO being processed on System
TCPware(R) for OpenVMS IPSO Configuration for line SVA-0:
Label       Level                          Authorities
-----       -----                          -----------
In:         UNCLASSIFIED to UNCLASSIFIED   None
Out:        SECRET to SECRET               DOE
Implied
  Receive:  UNCLASSIFIED                   None
  Transmit: SECRET                         DOE
ICMP Error: SECRET                         DOE
Adding implied label to transmitted datagrams
```

This imposes the SECRET level and DOE authority on all unlabeled datagrams going out on line SVA-0. The ADD keyword actually adds the IPSO header information to the datagrams.

Note from the SHOW IPSO output that the outgoing label and ICMP error requirements must already have been set to match the explicitly added label:

```
NETCU>SET IPSO /LINE=SVA-0 /OUT=(LEVEL=SECRET,AUTH=DOE)
NETCU>SET IPSO /LINE=SVA-0 /ERROR=(LEVEL=SECRET,AUTH=DOE)
Otherwise, the following error messages would appear:
%TCPWARE_NETCU-W-OUTOFRANGE, implied outgoing authority out-of-range for
SVA-0
%TCPWARE_NETCU-W-OUTOFRANGE, implied outgoing level out-of-range for SVA-0
%TCPWARE_NETCU-W-OUTOFRANGE, error authority out-of-range for SVA-0
%TCPWARE_NETCU-W-OUTOFRANGE, error level out-of-range for SVA-0
```

Accepting Datagrams Regardless of Authority

You can have the system accept datagrams regardless of their authority label by using the `AUTHORITY=ANY` value. In the following example, the host processes any datagram with a `SECRET` level and ignores the authority setting (note that the implicit labels and ICMP error were already set to match the line requirements):

```
NETCU>SET IPSO /LINE=SVA-0 /LABEL=(LEVEL=SECRET, AUTHORITY=ANY)
NETCU>SHOW IPSO
%TCPWARE_NETCU-I-NOIPSODATA, no IPSO being processed on System
TCPware(R) for OpenVMS IPSO Configuration for line SVA-0:
Label       Level               Authorities
-----       -----               -----------
In:         SECRET to SECRET    None
Out:        SECRET to SECRET    None

Implied
  Receive:  SECRET              None
  Transmit: SECRET              None
ICMP Error: SECRET              None
Ignoring authority field on received datagrams
Ignoring authority field on transmitted datagrams
```

Applying Implicit Labels

You can have the system associate labels with incoming or outgoing datagrams not having labels. Use the `/RECEIVE_IMPLICIT_LABEL` or `/TRANSMIT_IMPLICIT_LABEL` qualifier (without the `ADD` keyword), as in:

```
NETCU>SET IPSO /LINE=SVA-0 -
_NETCU>/RECEIVE_IMPLICIT_LABEL=(LEVEL=SECRET, AUTHORITY=GENSER)
%TCPWARE_NETCU-W-OUTOFRANGE, implied incoming authority out-of-range for
SVA-0
%TCPWARE_NETCU-W-OUTOFRANGE, implied incoming level out-of-range for SVA-0
NETCU>SHOW IPSO
%TCPWARE_NETCU-I-NOIPSODATA, no IPSO being processed on System
TCPware(R) for OpenVMS IPSO Configuration for line SVA-0:
```

```
Label         Level                            Authorities
-----         -----                            -----------
In:           UNCLASSIFIED to UNCLASSIFIED     None
Out:          UNCLASSIFIED to UNCLASSIFIED     None

Implied
  Receive:    SECRET                           GENSER
  Transmit:   UNCLASSIFIED                     None
ICMP Error:   UNCLASSIFIED                     None

%TCPWARE_NETCU-W-OUTOFRANGE, implied incoming authority out-of-range for
SVA-0
%TCPWARE_NETCU-W-OUTOFRANGE, implied incoming level out-of-range for SVA-0
```

In the example, the system manager associated the `SECRET GENSER` option with any unlabeled incoming datagrams on line SVA-0. The system does not actually add the label to the datagram header. (The example also shows the error messages resulting from not setting up matching requirements for the incoming and outgoing labels.) Additional keywords can accomplish the following:

`REQUIRED`	Require an IPSO label and do not use an implicit one
`NOREQUIRED`	Do not require an IPSO label and use an implicit one
`NONE`	Set an implicit label with an `UNCLASSIFIED` level and null authority

Datagrams with Extended Security Options

You can have the system accept or transmit datagrams containing extended security options. TCPware enables this by default. To explicitly enable extended options, use the `/EXTENDED_ALLOWED` qualifier for incoming or outgoing datagrams, or both, as in:

```
NETCU>SET IPSO /LINE=SVA-0 /EXTENDED_ALLOWED
```

You can selectively disallow security options using the `NOIN` and `NOOUT` keywords (as in `/EXTENDED_ALLOWED=NOIN`), or disallow them more generally using `/NOEXTENDED_ALLOWED`. The latter produces the following lines at the bottom of a `SHOW IPSO` output:

```
Extended IPSO options on received datagrams disallowed
Extended IPSO options on transmitted datagrams disallowed
```

Stripping Datagrams of Options

You can have the system strip any Basic Security Options present in an outgoing datagram. This is useful for routers and forwarding datagrams on which you do not want to impose security restrictions. Use the `/STRIP` qualifier, as in:

```
NETCU>SET IPSO /LINE=SVA-0 /STRIP
```

This produces `SHOW IPSO` output that includes the line `Stripping IPSO option on transmitted datagrams`. `/NOSTRIP` is the default.

Be careful stripping datagrams of basic options. The datagrams could get lost as a result of further checks that require these options.

Setting IP Security Options First in the Datagram Header

Some security systems require that the security options appear first in datagram headers. If this is the case, use the `/FIRST` qualifier, as in:

```
NETCU>SET IPSO /LINE=SVA-0 /FIRST
```

This produces `SHOW IPSO` output that includes the line `Placing IPSO option first on transmitted datagrams`. `/NOFIRST` is the default.

Enabling ICMP Errors

If a datagram arrives out of range, the originator should receive a returned ICMP message to indicate the error. For this to happen, the returned message itself must have the proper security label. Enable this using the `/ERROR_LABEL` qualifier, as in:

```
NETCU>SET IPSO /LINE=SVA-0 -
_NETCU>/ERROR_LABEL=(LEVEL=SECRET, AUTHORITY=GENSER)
```

Define only a single level and authority for an error label. The level and authority should always be a subset of the level and authority defined for the outgoing label. For example, the following `SHOW IPSO` output shows a valid port setting for the requirement above:

```
TCPware(R) for OpenVMS IPSO Configuration for line SVA-0:
Label       Level                          Authorities
-----       -----                          -----------
In:         UNCLASSIFIED to UNCLASSIFIED   None
Out:        UNCLASSIFIED to SECRET         GENSER

Implied
```

```
  Receive:  UNCLASSIFIED                    None
  Transmit: SECRET                          GENSER
ICMP Error: SECRET                          GENSER
```

To disallow ICMP message handling, use `/ERROR_LABEL=NONE`. This resets the ICMP error setting to `UNCLASSIFIED NONE`. However, disallowing ICMP message handling can be risky.

> **Note:** Always set the system requirements first. This sets each line identically. Then make isolated changes to the line requirements.

Automatic Startup

The STARTNET procedure uses the `TCPWARE:ROUTING.COM` file to automatically set security options at TCPware startup.

The file can contain the NETCU commands used to set IPSO options, as discussed earlier. Any commands in this file can refer to the `IPSO_AUTHORITIES.` file for authority definitions. (See *Site-Specific Authority Names*.) The below example shows sample content of the `ROUTING.COM` file.

Sample `ROUTING.COM` file:

```
$ NETCU
SET IPSO /SYSTEM /LABEL=(LEVEL=(UNCLASSIFIED,SECRET),AUTH=(DOE,SCI))
SET IPSO /SYSTEM /RECEIVE_IMPLICIT=(LEVEL=UNCLASSIFIED,AUTH=DOE)
SET IPSO /SYSTEM /TRANSMIT_IMPLICIT=(LEVEL=SECRET,AUTH=DOE)
SET IPSO /SYSTEM /ERROR_LABEL=(LEVEL=SECRET,AUTH=DOE)
SET IPSO /LINE=SVA-0 /OUT_LABEL=(LEVEL=SECRET,AUTH=DOE)-
/IN_LABEL=(LEVEL=(UNCLASSIFIED,SECRET),AUTH=(DOE,SCI,"DOE+SCI"))
EXIT
```

The commands produce the following result:

```
NETCU>SHOW IPSO
TCPware(R) for OpenVMS IPSO Configuration for System:
Label       Level                     Authorities
-----       -----                     -----------
In:         UNCLASSIFIED to SECRET    DOE
                                      SCI
Out:        UNCLASSIFIED to SECRET    DOE
                                      SCI

Implied
```

```
  Receive:  UNCLASSIFIED              DOE
  Transmit: SECRET                    DOE
ICMP Error: SECRET                    DOE


TCPware(R) for OpenVMS IPSO Configuration for line SVA-0:

Label       Level                     Authorities
-----       -----                     -----------
In:         UNCLASSIFIED to SECRET    DOE
                                      SCI
                                      (DOE+SCI)
Out:        SECRET to SECRET          DOE

Implied
  Receive:  UNCLASSIFIED              DOE
  Transmit: SECRET                    DOE
ICMP Error: SECRET                    DOE
```

Site-Specific Authority Names

The `TCPWARE:IPSO_AUTHORITIES.` file (include the dot at the end) contains a table translating bit masks to protection authorities. Edit this file to create site-specific authorities (aliases) or authority combinations you can associate with an alias name. You can create an alias authority name that is the name of the system a certain combination of authorities frequently protect. For example, you can associate the name ALPHA, the system that frequently needs to have SCI and NSA (inclusive) authority protection, with the authority combination `SCI+NSA`.

To do this, enter the value ALPHA in the file along with the logically OR'd bit mask values of the combined pre-defined authorities, as indicated in the below example. In the case of ALPHA, the hexadecimal value `0x30` comes from `0x20 + 0x10`. In the case of DELTA, the hexadecimal value `0xC0` comes from `0x80 + 0x40` (`GENSER+SIOP-ESI`).

```
!GENSER    0x80
!SIOP-ESI  0x40
!SCI       0x20
!NSA       0x10
!DOE       0x08
!
!SITE-SPECIFIC:
ALPHA      0x30     !SCI+NSA
BETA       0x50     !SIOP-ESI+NSA
DELTA      0xC0     !GENSER+SIOP-ESI
```

As indicated in the above example, the predefined authorities (GENSER to DOE) are in the IPSO_AUTHORITIES. file for reference purposes only and are commented out using an exclamation point (!). You must also comment out the description after each site-specific entry.

> **Caution!** Do not delete the IPSO_AUTHORITIES. file.

This setup allows you to specify a site-specific authority in an IPSO command such as:

```
NETCU>SET IPSO /LINE=SVA-0 /LABEL=(LEVEL=(UNCLASSIFIED,SECRET),-
_NETCU>AUTHORITY=(GENSER,ALPHA))
```

which, if combined with other appropriate settings, produces the SHOW IPSO output:

```
NETCU>SHOW IPSO
%TCPWARE_NETCU-I-NOIPSODATA, no IPSO being processed on System
TCPware(R) for OpenVMS IPSO Configuration for line SVA-0:
Label       Level                    Authorities
-----       -----                    -----------
In:         UNCLASSIFIED to SECRET   GENSER
                                     ALPHA(NSA+SCI)
Out:        UNCLASSIFIED to SECRET   GENSER
                                     ALPHA(NSA+SCI)

Implied
  Receive:  UNCLASSIFIED             GENSER
  Transmit: UNCLASSIFIED             GENSER
ICMP Error: UNCLASSIFIED             GENSER
```

Full SHOW IPSO Output

With the /FULL qualifier, the SHOW IPSO command produces the usual output along with additional counter information for incoming and outgoing datagrams. This information includes the number of datagrams that:

- Contain Basic Security Options
- Were delivered or transmitted
- Contain Extended Security Options
- Use implicit labeling
- Were rejected as out-of-range

- Were rejected due to containing extended options
- Lack a required basic option.

Typical output is:

```
NETCU>SHOW IPSO /FULL
TCPware(R) for OpenVMS IPSO Configuration for System:
Label       Level                          Authorities
-----       -----                          -----------
In:         UNCLASSIFIED to UNCLASSIFIED  None
Out:        UNCLASSIFIED to UNCLASSIFIED  None

Implied
  Receive:  UNCLASSIFIED                  None
  Transmit: UNCLASSIFIED                  None
ICMP Error: UNCLASSIFIED                  None

          Incoming datagrams screened by IPSO
              0 contained a BSO
            559 were delivered to receivers
              0 contained extended options
            559 used implicit labelling
              0 were rejected as out-of-range
              0 were rejected due to containing ESO
              0 lacked a required BSO

          Outgoing datagrams screened by IPSO
              0 contained a BSO
            100 were successfully transmitted
              0 contained extended options
              0 used implicit labelling
              0 were rejected as out-of-range
              0 were rejected due to containing ESO
              0 lacked a required BSO
```

Troubleshooting

Out-of-range datagram error messages appear both as warnings after `SET IPSO` and in `SHOW IPSO` output. Here are a few conditions that could produce out-of-range error messages:

- A label was set without an associated implicit label:

```
NETCU>SET IPSO /SYS /IN=(LEVEL=(CONFIDENTIAL,SECRET),AUTH=DOE)
%TCPWARE_NETCU-W-OUTOFRANGE, implied incoming authority out-of-range
for System
%TCPWARE_NETCU-W-OUTOFRANGE, implied incoming level out-of-range for
System
NETCU>SHOW IPSO
```

```
TCPware(R) for OpenVMS IPSO Configuration for System:
Label       Level                          Authorities
-----       -----                          -----------
In:         CONFIDENTIAL to SECRET         DOE
Out:        UNCLASSIFIED to UNCLASSIFIED   None

Implied
  Receive:  UNCLASSIFIED                   None
  Transmit: UNCLASSIFIED                   None
ICMP Error: UNCLASSIFIED                   None

%TCPWARE_NETCU-W-OUTOFRANGE, implied incoming authority out-of-range
for System
%TCPWARE_NETCU-W-OUTOFRANGE, implied incoming level out-of-range for
System
```

- An implicit label is not a subset of the default label setting:

```
NETCU>SET IPSO /SYS /REC=(LEVEL=CONFIDENTIAL,AUTH=DOE)
%TCPWARE_NETCU-W-OUTOFRANGE, implied incoming authority out-of-range
for System
%TCPWARE_NETCU-W-OUTOFRANGE, implied incoming level out-of-range for
System
NETCU>SHOW IPSO
TCPware(R) for OpenVMS IPSO Configuration for System:

Label       Level                          Authorities
-----       -----                          -----------
In:         UNCLASSIFIED to UNCLASSIFIED   None
Out:        UNCLASSIFIED to UNCLASSIFIED   None
Implied
  Receive:  CONFIDENTIAL                   DOE
  Transmit: UNCLASSIFIED                   None
ICMP Error: UNCLASSIFIED                   None

%TCPWARE_NETCU-W-OUTOFRANGE, implied incoming authority out-of-range
for System
%TCPWARE_NETCU-W-OUTOFRANGE, implied incoming level out-of-range for
System
```

- An ICMP error label does not match the outgoing label setting:

```
NETCU>SET IPSO /SYS /ERROR=(LEVEL=CONFIDENTIAL,AUTH=DOE)
%TCPWARE_NETCU-W-OUTOFRANGE, error authority out-of-range for System
%TCPWARE_NETCU-W-OUTOFRANGE, error level out-of-range for System
NETCU>SHOW IPSO
TCPware(R) for OpenVMS IPSO Configuration for System:

Label       Level                          Authorities
-----       -----                          -----------
In:         UNCLASSIFIED to CONFIDENTIAL   SCI
```

```
Out:         SECRET to TOP_SECRET          SCI

Implied
  Receive:  CONFIDENTIAL                   SCI
  Transmit: SECRET                         SCI
ICMP Error: CONFIDENTIAL                   DOE

%TCPWARE_NETCU-W-OUTOFRANGE, error authority out-of-range for System
%TCPWARE_NETCU-W-OUTOFRANGE, error level out-of-range for System
```

23. Configuring the Secure Shell (SSH) V1 Server

This chapter describes how to configure and maintain the TCPware Secure Shell (SSH) server v1.

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.

SSH1 and SSH2 Differences

SSH1 and SSH2 are different, and incompatible, protocols. The SSH1 implementation is based on the V1.5 protocol, and the SSH2 implementation is based on the V2. While SSH2 is generally regarded to be more secure than SSH1, both protocols are offered by TCPware, and although they are incompatible, they may exist simultaneously on a system. The server front-end identifies what protocol a client desires to use and will create an appropriate server for that client.

Restrictions:

When using SSH1 to connect to a VMS server, if the VMS account is set up with a secondary password, SSH1 does not prompt the user for the secondary password. If the VMS 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.

Understanding the TCPware Secure Shell Server

Secure Shell daemon (SSHD) is the daemon program for SSH1 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.

Servers and Clients

A TCPware 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 TCPware, but any SSH client that uses SSH version 1 protocol may be used to access the server. Examples of such programs are MultiNet SSH, TCPware SSH, and FISSH on OpenVMS; SecureCRT and TTSSH on Windows-based systems; and other SSH programs on UNIX-based systems.

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 public keys to the client.
- The client compares the host key 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 host key 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:
    - .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.

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.

Options

SSHD can be configured using command-line options or a configuration file. Command-line options override values specified in the configuration file. The following configuration file template is in the TCPware kit: `SSHD_CONFIG.TEMPLATE`.

> **Note:** The recommended method to start the SSHD Master process is to use the `@TCPWARE:STARTNET SSH` command. All of these options are set using `@TCPWARE:CNFNET SSH`.

Configuration File

SSHD reads configuration data from `TCPWARE:SSHD_CONFIG` (or the file specified with `/CONFIG_FILE` on the command line). 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
`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_x`' to specify greater than, less than, or inclusive port range. By default, all port forwardings are allowed.
`AllowForwardingTo`	Host/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. 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_x' to specify greater than, less than, or inclusive port range. By default, all port forwardings are allowed.
AllowGroups	List		Can be followed by any number of OpenVMS rights identifier patterns, separated by spaces. Login is allowed only if the user's list of rights identifiers contains an identifier that matches one of the patterns. You can use * and ? as wildcards in the patterns. By default, logins as all users are allowed. All other login authentication steps must be completed successfully. DenyGroups is an additional restriction.
AllowHosts	Host list		Can be followed by any number of host name patterns, separated by spaces. Login is allowed only from hosts whose name matches one of the patterns. You can use * and ? as wildcards in the patterns. 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. By default, all hosts are allowed to connect.
AllowSHosts	Host list		Can be followed by any number of host name patterns, separated by spaces. .SHOSTS (and

			`.RHOSTS` and `SSH_DIR:HOSTS.EQUIV`) entries are honored for hosts whose name matches one of the patterns. 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 host name. By default, all hosts are allowed to connect.
`AllowTcpForwarding`	Y/N	Y	Specifies whether TCP forwarding is permitted. The default is `yes`. Note that disabling TCP forwarding does not improve security in any way, as users can always install their own forwarders.
`AllowUsers`	User list		Can be followed by any number of username patterns or *user@host* patterns, separated by spaces. Host name may be either the DNS name or the IP address. Login is allowed only for users whose name matches one of the patterns. You can use `*` and `?` as wildcards in the patterns. By default, logins as all users are allowed. Note that all other login authentication steps must be completed successfully. `DenyUsers` is an additional restriction.
`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	Host/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. 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.
DenyGroups	Rights list		Can be followed by any number of OpenVMS rights identifier patterns, separated by spaces. Login is disallowed only if the user's list of rights identifiers contains an identifier that matches one of the patterns.

DenyHosts	Host list		Can be followed by any number of host name patterns, separated by spaces. Login is disallowed from the host whose name matches any of the patterns.
DenySHosts	Host list		Can be followed by any number of host name patterns, separated by spaces. .SHOSTS (and .RHOSTS and SSH_DIR:HOSTS.EQUIV) entries whose name matches any of the patterns are ignored.
DenyUsers	User list		Can be followed by any number of username patterns or *user@host* patterns, separated by spaces. A host name may be either the DNS name or the IP address. Login is disallowed a user whose name matches any of the patterns.
HostKey	Filename	SSH_HOST_KEY	Specifies the file containing the private key. The default is TCPWARE:SSH_HOST_KEY.
IdleTimeout	Time	0	Sets the idle timeout limit in: • seconds (s or nothing after the number) • minutes (m) • hours (h) • days (d) • weeks (w) If the connection has been idle (all channels) for the time specified, the process is terminated and the connection is closed. The default is zero; it never times out. An idle process is one that has done no I/O to stdin or stdout in the timeout value.
IgnoreRhosts	Y/N	N	Specifies that the SYS$LOGIN:RHOSTS file will not be used in authentication.

			`SSH_DIR:HOSTS.EQUIV` is still used. The default is `no`.
`KeepAlive`	Y/N	Y	Specifies whether the system should send keepalive messages to the other side. If sent, termination of the connection or crash of one of the machines will be noticed. This means that connections will terminate if the route is down temporarily. If keepalives are not ended, sessions may hang indefinitely on the server, leaving “ghost” users and consuming server resources. The default is `yes` (to send keepalives), and the server will notice if the network goes down or the client host reboots. This avoids infinitely hanging sessions. To disable keepalives, set the value to `no` in both the server and the client configuration files.
`KeyRegenerationInterval`	Time	3600	Specifies how long to wait before the server key is regenerated automatically. Regeneration prevents decrypting captured sessions by later breaking into the machine and stealing the keys. The key is never stored on disk. If the value is 0, the key is never regenerated. The default is 3600 seconds.
`ListenAddressee`			Specifies the IP address of the interface where the SSHD server socket is bound.

`LoginGraceTime`	Time	600	Specifies the time the server should disconnect if the user has not logged in successfully. If the value is 0, there is no time limit. The default is 600 seconds.
`PasswordAuthentication`	Y/N	Y	Specifies whether password authentication is allowed. The default is `yes`.
`PermitEmptyPasswords`	Y/N	N	Specifies whether the server allows login to accounts with empty password strings when password authentication is allowed. The default is no. **Note:** Use of this keyword may contribute to your system becoming unsecure. Process Software encourages you to NOT enable the use of empty passwords.
`PermitRootLogin`	Y/N	N	Specifies whether the user can log in as `SYSTEM` using SSH. This keyword may be set to: `yes` - allows SYSTEM logins through any of the authentication types allowed for other users. `no` (default) - disables SYSTEM logins through any of the authentication methods (`nopwd` and `no` are equivalent), unless you have a rhosts or `SYS$DISK:[`*`login_dir`*`.SSH]AUTHORIZED_KEYS` file in the `SYS$MANAGER` directory. `nopwd` - disables password-authenticated SYSTEM logins.

			System login with RSA authentication when the “command” option has been specified is allowed regardless of the value of this keyword (which may be useful for taking remote backups even if SYSTEM login is not allowed).
QuietMode	Y/N	N	Specifies whether the system runs in quiet mode. In quiet mode, nothing is logged in the system log, except fatal errors. The default is no.
RandomSeed	Filename	Random_seed	Specifies the file containing the random seed for the server. This file is created automatically and updated regularly. The default is SSH_DIR:SSH_RANDOM_SEED.
RhostsAuthentication	Y/N	N	Specifies whether authentication using SYS$LOGIN:RHOSTS or SSH_DIR:HOSTS.EQUIV files is sufficient. Normally, this method should not be permitted because it is insecure. Use RhostsRSAAuthentication because it performs RSA-based host authentication in addition to normal rhosts or SSH_DIR:HOSTS.EQUIV authentication. The default is no.
RhostsRSAAuthentication	Y/N	Y	Specifies whether SYS$LOGIN:RHOSTS or SSH_DIR:HOSTS.EQUIV authentication together with successful RSA host authentication is allowed. The default is yes.
RSAAuthentication	Y/N	Y	Specifies whether pure RSA authentication is allowed. The default is yes.
SilentDeny			Specifies whether denied (or not allowed) connections are denied silently (just close the

			connection, no logging, etc.) or if they are closed cleanly (send an error message and log the connection attempt). Defaults to silent mode, `yes`.
`StrictIntrusionLogging`	Y/N	N	Determine how intrusion records are created by failed authentication attempts
`StrictModes`	Y/N	N	Specifies whether SSH should check file protection and ownership of the user's home directory and rhosts files before accepting login. The default is yes.
`SyslogFacility`	Syslog level	`"AUTH"`	Gives the facility code that is used when logging messages from SSHD. The values are: `DAEMON` (default), `AUTH`, `USER`, `LOCAL0`, `LOCAL1`, `LOCAL2`, `LOCAL3`, `LOCAL4`, `LOCAL5`, `LOCAL6`, `LOCAL7`.
`X11DisplayOffset`	#offset	10	Specifies the first display number available for SSHD's X11 forwarding. This prevents SSHD from interfering with real X11 servers.

Starting the SSH Server for the First Time

Follow these instructions for using SSH for the first time.

1. Use the TCPware `CNFNET` utility to configure the SSH server.

> **Note:** TCPware must be running before issuing the `SSHKEYGEN` command.

2. Use SSHKEYGEN to create the file SSH_HOST_KEY in the TCPWARE: directory.

```
$ NETCU 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 TCPWARE_ROOT:[TCPWARE]SSH_HOST_KEY
Your identification has been saved in TCPWARE:SSH_HOST_KEY.
Your public key is:
1024 37
12103183655766986978653678692919694763882284449699056118642763 08
90727769044627444159668210201094636176442023972946422779467185 49
44044425775948682970871710133597438531824425799238013020208440 11
53437549098475139731602493247359131463302324104249367510159536 11
18716872491123857940537322891584850459319961275605927
SYSTEM@gg1.prr.com
Your public key has been saved in
TCPWARE_ROOT:[TCPWARE]SSH_HOST_KEY.pub
```

3. Review the SSH1 server configuration file, SSH_DIR:SSHD_CONFIG, making any appropriate modifications. As delivered, the configuration file provides a reasonably secure SSH environment.
4. Restart SSH. This creates the SSH server process and defines the SSH logical names.

```
$ @TCPWARE:STARTNET SSH
$ SHOW PROCESS "SSHD Master"
7-JUL-2014 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*

  "SSH2_DIR" ="TCPWARE_SPECIFIC:[TCPWARE.SSH2]"
  "SSH_DIR" = "TCPWARE_SPECIFIC:[TCPWARE]"
  "SSH_EXE" = "TCPWARE_COMMON:[TCPWARE]"
  "SSH_LOG" = "TCPWARE_SPECIFIC:[TCPWARE.SSH]"
  "SSH_MAX_SESSIONS"= "100"
  "SSH_TERM_MBX" = "MBA286:"
```

```
  "TCPWARE_SSH2_HOSTKEY_DIR" =
"TCPWARE_SPECIFIC:[TCPWARE.SSH2.HOSTKEYS]"
  "TCPWARE_SSH2_KNOWNHOSTS_DIR" =
"TCPWARE_SPECIFIC:[TCPWARE.SSH2.KNOWNHOSTS]"
  "TCPWARE_SSH_ALLOW_EXPIRED_PW" = "1"
  "TCPWARE_SSH_ALLOW_PREEXPIRED_PW" = "1"
  "TCPWARE_SSH_DISPLAY_SYS$ANNOUNCE" = "1"
  "TCPWARE_SSH_ENABLE_SSH1_CONNECTIONS" = "1"
  "TCPWARE_SSH_ENABLE_SSH2_CONNECTIONS" = "1"
  "TCPWARE_SSH_LOG_MBX" = "MBA287"
  "TCPWARE_SSH_PARAMETERS_0" = "/BITS=768/QUIET/PORT=22"
  "TCPWARE_SSH_PARAMETERS_1" = "/KEY_GEN_TIME=3600"
  "TCPWARE_SSH_PARAMETERS_2" = "/DEBUG=4"
  "TCPWARE_SSH_PARAMETERS_3" = ""
```

Changing SSH Configuration File After Enabling SSH

If you make a change to the SSH configuration file after you have enable SSH, you have to restart SSH. To have the changes take effect, use the command:

```
$ @TCPWARE:RESTART SSH
```

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 necessary information to it, such as the server key and operating parameters.
2. SSHD performs validation for the user.
3. Assuming the login is successful, SSHD creates a pseudo terminal for the user (an `_FTA`*`nn`*`:` device). This device is owned by the user attempting to log in.
4. 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.
5. SSH generates the file `SSH_LOG: 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 SSH_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.

AUTHORIZED_KEYS File Format

The `SYS$DISK:[`*`login_dir`*`.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`	The length of the key in bits.
`comment`	Not used for anything (but may be convenient for the user to identify the key).
`exponent`	A component used to identify and make up the key.
`modulus`	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) consist of comma-separated option specifications. No spaces are permitted, except within double quotes. Option names are case insensitive.

The following option specifications are supported:

Option Specification	**Description**
`Allowforwardingport="`*`ports`*`"`	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_x`' to specify greater than, less than, or inclusive port range. By default, all port forwardings are allowed. The quotes (" ") are required. For example: `allowforwardingport "2,52,2043"`
`Allowforwardingto="`*`hosts and ports`*`"`	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_x' 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="*ports*"	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="*hosts and ports*"	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.

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.tazm.com
allowforwardingport="localhost:80"
1024 35 23...2334 www@localhost
```

SSH_KNOWN_HOSTS File Format

The `TCPWARE:SSH_KNOWN_HOSTS` and `SYS$DISK:[`*`login_dir`*`.SSH]KNOWN_HOSTS` files contain host public keys 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 to 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 host key. They can be obtained from `TCPWARE: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 host keys 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 host keys, generate them by a script or by copying `TCPWARE:SSH_HOST_KEY.PUB` and adding the host names at the front.

Example

```
closenet,closenet.hut.fi,...,130.233.208.41
1024  37  159...93 closenet.example.com
```

FILES

File Name	Description
`TCPWARE: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 '−'. 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.

TCPWARE:SHOSTS.EQUIV	Processed as TCPWARE:HOSTS.EQUIV. May be useful in environments that want to run both rshell/rlogin and SSH.
TCPWARE:SSH_HOST_KEY	Contains the private part of the host key. This file does not exist when TCPware is installed. The SSH server starts only with this file. This file must be created manually using the command: $ NETCU SSHKEYGEN/SSH1/HOST. This file should be owned by SYSTEM, readable only by SYSTEM, and not accessible to others. To create a host key with a name that is different than what SSHKEYGEN creates, do one of the following: • Generate with NETCU SSHKEYGEN/SSH1/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 SSH_DIR points to the same directory as TCPWARE:. Refer to the *TCPware User's Guide*, Chapter 16, "Accessing Remote Systems with the Secure Shell (SSH) Utilities" for more details about SSHKEYGEN.
TCPWARE:SSH_HOST_KEY.PUB	Contains the public part of the host key. 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.
TCPWARE:SSH_KNOWN_HOSTS SYS$DISK:[*login_dir*.SSH]KNOWN_HOSTS	Checks the public key 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 writable only by SYSTEM (the owner).

	TCPWARE:SSH_KNOWN_HOSTS should be world-readable, and [.SSH]KNOWN_HOSTS can, but need not be, world-readable.

TCPWARE:SSH_RANDOM_SEED	Contains a seed for the random number generator. This file should only be accessible by system. The file SSH_RANDOM_SEED. has the potential for increasing its number of versions. SSH_RANDOM_SEED. is created in the SSH_DIR:(TCPWARE:) directory as well as in individual user accounts in the SYS$LOGIN:[.SSH] directory.
	
	This DCL command limits the number of versions of this file in the TCPWARE directory:
	
	`SET FILE VERSION_LIMIT=x TCPWARE:SSH_RANDOM_SEED.`
	
	This DCL command limits the number of versions of this file in the user's SYS$LOGIN:[.SSH]directory.
	
	`SET FILE VERSION_LIMIT=x SYS$LOGIN:[.SSH]SSH_RANDOM_SEED.`
	
	Or
	
	`CREATE /DIRECTORY /VERSION_LIMIT=x SYS$LOGIN:[.SSH]SSH_RANDOM_SEED.`

TCPWARE:SSHD_CONFIG	Contains configuration data for SSHD. This file should be writable by system only, but it is recommended (though not necessary) that it be world-readable.
SYS$DISK:[<login_dir>.SSH] AUTHORIZED_KEYS	Lists the RSA keys that can be used to log into the user's account. This file must be readable by system (which may on some machines imply it being world-readable). It is recommended that

	it not be accessible by others. The format of this file is described above.
SYS$LOGIN:.SHOSTS	Permits access using SSH only. For SSH, this file is the same as for `.rhosts`. However, this file is not used by rlogin and rshell daemon.
SYS$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 (unless compiled with the `-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.

SSH Logicals

These logicals are used with the SSH server in the system logical name table.

Logical	**Description**
SSH_DIR	Points to the directory where the master server log file is kept. Normally, this is TCPWARE_SPECIFIC:[TCPWARE].
SSH_EXE	Points to the directory where SSH executables are kept. Normally, this is TCPWARE_COMMON:[TCPWARE].
SSH_LOG	Points to the directory where the log files are kept. Normally, this is TCPWARE_SPECIFIC:[TCPWARE.SSH].

SSH_MAX_SESSIONS	Set this to the maximum number of concurrent SSH sessions you want to allow on the server system. If SSH_MAX_SESSIONS is not defined, the default is 9999. Setting SSH_MAX_SESSIONS to zero (0) will cause an error. The value must be between 1 and 9999. It is defined through @TCPWARE:CNFNET SSH. The configuration procedure should write these to the common configuration file and check the values at start up and delete them at shutdown.
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.
TCPWARE_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 VMS 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.
TCPWARE_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 VMS 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.
TCPWARE_SSH_KEYGEN_MIN_PW_LEN	Defines the minimum passphrase length when one is to be set in SSHKEYGEN. If not defined, defaults to zero.
TCPWARE_SSH_PARAMETERS_*n*	These parameters are used to start SSHD_MASTER. They are parameters set by @TCPWARE:CNFNET SSH.
TCPWARE_SSH_USE_SYSGEN_LGI	If defined, causes SSHD to use the VMS SYSGEN value of LGI_PWD_TMO to set the login grace time, overriding

	anything specified in the command line or the configuration file.

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.

File Name	Description
`SSHD_MASTER.LOG`	This log file is created by SSHD_MASTER.
`SSHD.LOG`	This log file is created by each SSHD daemon.

24. Configuring the Secure Shell (SSH) V2 Server

This chapter describes how to configure and maintain the TCPware Secure Shell (SSH) server v2.

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.

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 TCPware, and although they are incompatible, they may exist simultaneously on a TCPware system. The TCPware 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 TCPware SSH2 (***this does not 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 v6.0 and earlier systems prior to using SSH. This is the `AACRT060.A` file. You can find the ECO on the TCPware CD in the following directory: `VAX55_DECC_RTL.DIR`.

Restrictions:

When using SSH2 to connect to a VMS server, if the VMS account is set up with a secondary password, SSH2 does not prompt the user for the secondary password. If the VMS primary password entered is valid, the user is logged in, bypassing the secondary password.

When using SSH2 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 SSH2. If this behavior poses a problem, log into the remote system using SSH2 in interactive mode to execute the program.

Understanding the TCPware Secure Shell Server

Secure Shell daemon (SSHD) is the daemon program for SSH2 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.

Servers and Clients

A TCPware 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 TCPware, but any SSH client that uses SSH version 2 protocol may be used to access the server. Examples of such programs are MultiNet SSH2 for OpenVMS; SecureCRT, puTTY, and F-Secure SSH Client for Windows, MacSSH for Apple systems, 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 VMScluster 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.

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"
$!
```

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.

Configuring SSHD Master

SSHD Master is configured using the TCPware CNFNET command procedure (`@TCPWARE:CNFNET SSH`). When CNFNET is used to make changes to the SSHD Master configuration, SSH must be restarted via `@TCPWARE:RESTART SSH`.

SSH2 Configuration File

SSHD reads configuration data from its configuration file. By default, this file is `SSH2_DIR:SSHD2_CONFIG`; however, it may be modified by executing `@TCPWARE:CNFNET SSH` and setting the alternate SSH2 configuration file. A default configuration is supplied in the file `SSH2_DIR:SSHD_CONFIG.TEMPLATE`. Lines starting with # and empty lines are interpreted as comments. The following keywords are possible. Keywords are case insensitive.

Keyword	Value	Default	Description

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 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 public key authentication.
`AuthPublicKey.MaxSize`	Number	0	Specifies the maximum size of a public key that can be used to log in. Value of 0 disables the check.
`AuthPublicKey.MinSize`	Number	0	Specifies the minimum size of a public key 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 rights list 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 rights list 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 the *ForwardACL Notes* below.
`ForwardAgent`	Y/N	Y	Enable agent forwarding
`HostCA`	Certificate	None	Specifies the CA certificate (in binary or PEM (base64) format) to be used when 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 public keys. 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 `HostKeyFile` option. If multiple certificates with the same public key 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	Host key 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)
`IgnoreRhosts`	Y/N	N	Ignore local rhosts
`IgnoreRootRhosts`	Y/N	Y	Ignore system rhosts
`KeepAlive`	Y/N	Y	Send keepalives
`LdapServers`	Server URL	None	Specified as ldap://*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 `LdapServers` is 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 `Pki` keyword. Multiple mapping files are permitted per one `Pki` keyword. 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
`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 `Pki` keywords 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 `Pki` keyword if argument is `Y`.
`PrintMotd`	Y/N	Y	Display `SYS$WELCOME` when logging in
`PublicHostKeyFile`	Filename	Hostkey.pub	Host key 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 a 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 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 “no”, you should not set `RequireReverseMapping` to `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`	Facility	None	Defines what log facility the sftp-server will use. By default, no logging is performed.
`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.Allow-Users` but matches groups instead of usernames.
`Terminal.DenyGroups`	pattern	None	Similar to `Terminal.Deny-Users` but matches groups instead of usernames
`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 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@ssh.com, rc2-cbc@ssh.com, 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@ssh.com, rc2-cbc@ssh.com, none`

	group `ANYCIPHER` consists of: `aes128-ctr, aes128-cbc, aes192-ctr, aes192-cbc, aes256-ctr, aes256-cbc, 3des-ctr, 3des-cbc, blowfish-ctr, blowfish-cbc, des-cbc@ssh.com, rc2-cbc@ssh.com`
	group `ANYSTD` consists of: `aes128-ctr, aes128-cbc, aes192-ctr, aes192-cbc, aes256-ctr, aes256-cbc, 3des-ctr, 3des-cbc, blowfish-ctr, blowfish-cbc, 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`

HostSpecificConfig Notes:

The global server file (SSH_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        foo.bar.com         dka0:[sshconfigs]fooconfig.dat
HostSpecificConfig        lima.beans.com      limaconfig.dat
```

In the first instance, an incoming connection from "foo.bar.com" will use the subconfig file dka0:[sshconfigs]fooconfig.dat. In the second example, an incoming connection from "lima.beans.com" will use ssh2_dir:limaconfig.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.

UserSpecificConfig Notes:

The global server file (SSH2_DIR:SSHD2_CONFIG) now 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
```

user 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     pointhair.dat
```

In the first instance, an incoming connection for user dilbert will use the subconfig file dka0:[sshconfigs]dilbert.dat. In the second example, an incoming connection from user boss at system lima.beans.com will use ssh2:dir:pointyhair.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.

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, Process Software may implement additional features using this authentication method.

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 forwads, *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 forwads, 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 forwards 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.

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 (following `egrep`-style syntax). If it matches an alternname (of type `email-address`) in the certificate, the certificate is good lor 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

```
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 `EmailRegex` permits 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 `SubjectRegex` rule.

Starting the SSH Server for the First Time

Follow these instructions for using SSH for the first time.

1. Use the TCPWARE `CNFNET` utility to configure the SSH server. Note that TCPware must be running before proceeding on to the next step.
2. Use `SSHKEYGEN /SSH2` to generate an SSH2 key and to create the server keys in the `SSH2_HOSTKEY_DIR` directory:

```
$ DEFINE TCPWARE_SSH2_HOSTKEY_DIR -
_$ TCPWARE_SPECIFIC:[TCPWARE.SSH2.HOSTKEYS]
$ NETCUSSHKEYGEN /SSH2 /HOST
Generating 1024-bit dsa key pair
  8 .oOo.oOoo.oO
Key generated.
1024-bit dsa, lillies@flower.example.com, Mon Aug 09 2020 09:19:47
Private key saved to tcpware_ssh2_hostkey_dir:hostkey.
Public key saved to tcpware_ssh2_hostkey_dir:hostkey.pub
```

3. Copy the template configuration file to the `ssh2_dir:` directory renaming it `SSHD2_CONFIG.`:

```
$ COPY TCPWARE_COMMON:[TCPWARE]SSHD2_CONFIG.TEMPLATE -
_$ TCPWARE_SPECIFIC:[TCPWARE.SSH2]SSHD2_CONFIG.
```

> **Note:** As delivered, the template file provides a reasonably secure SSH environment. However, Process Software recommends this file be examined and modified appropriately to reflect the security policies of your organization.

4. Restart the SSH component. This creates the SSH server process and defines the SSH logical names.

```
$ @TCPWARE:RESTART SSH
$ SHOW PROCESS "SSHD Master"
7-JUL-2020 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*

  "TCPWARE_SSH2_HOSTKEY_DIR" =
            "TCPWARE_SPECIFIC:[TCPWARE.SSH2.HOSTKEYS]"
  "TCPWARE_SSH2_KNOWNHOSTS_DIR" =
            "TCPWARE_SPECIFIC:[TCPWARE.SSH2.KNOWNHOSTS]"
  "TCPWARE_SSH_ENABLE_SSH2_CONNECTIONS" = "1"
  "SSH2_DIR" = "TCPWARE_SPECIFIC:[TCPWARE.SSH2]"
  "SSH_DIR" = "TCPWARE_SPECIFIC:[TCPWARE]"
  "SSH_EXE" = "TCPWARE_COMMON:[TCPWARE]"
  "SSH_LOG" = "TCPWARE_SPECIFIC:[TCPWARE.SSH]"
  "SSH_TERM_MBX" = "MBA36:"
```

Configuring the SSH2 Server on a VMScluster with a Common System Disk

When configuring the SSH2 server on a VMScluster with a common system disk, you must create the appropriate directories on all cluster nodes other than the one on which TCPware 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 TCPware was originally installed.

- Create the following directories:

```
$ CREATE/DIR TCPWARE_SPECIFIC[TCPWARE.SSH2]/PROT=(WO:RE,GR:RE)
$ CREATE/DIR -
_$ TCPWARE_SPECIFIC[TCPWARE.SSH2.KNOWNHOSTS]/PROT=(WO:RE,GR:RE)
$ CREATE/DIR -
_$ TCPWARE_SPECIFIC[TCPWARE.SSH2.HOSTKEYS]/PROT=(WO:RE,GR:RE)
$ CREATE/DIR TCPWARE_SPECIFIC[TCPWARE.SSH]/PROT=(WO:RE,GR:RE)
```

- Edit the `TCPWARE_SPECIFIC:[TCPWARE_SSH2]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 SSH2 server using `@TCPWARE:CNFNET SSH`

- Generate the SSH2 host keys using `NETCU SSHKEYGEN/SSH2/HOST`
- (Re)start SSHD Master using `@TCPWARE:RESTART SSH`

Changing SSH2 Configuration File After Enabling SSH2

If you make a change to the SSH configuration file after you have enabled SSH, you must restart SSH for these changes to take effect.

```
$ @TCPWARE:RESTART SSH
```

> **Note:** When restarting SSH, all active SSH server sessions are terminated. Active client sessions are not affected.

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 pseudoterminal for the user (an `_FTA`*`nn`*`:` device). This device is owned by the user attempting to log in.
3. SSHD creates an interactive process on the pseudoterminal, 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 TCPWARE_ROOT:[TCPWARE.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.

FILES

TCPWARE: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 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.

TCPWARE:SHOSTS.EQUIV

Processed as `TCPWARE:HOSTS.EQUIV`. May be useful in environments that want to run both rshell/rlogin and ssh.

TCPWARE_SSH2_HOSTKEY_DIR:HOSTKEY

Contains the private part of the host key. This file does not exist when TCPware is installed. The SSH server starts only with this file. This file must be created manually using the command:

```
$ NETCU SSHKEYGEN /SSH2 /HOST.
```

This file should be owned by SYSTEM, readable only by SYSTEM, and not accessible to others.

To create a host key with a name that is different than what `SSHKEYGEN` creates, do one of the following:

- Generate with `NETCU 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 TCPWARE_SPECIFIC:[TCPWARE.SSH2] directory.

Refer to the *TCPware User's Guide*, Chapter 16, for more details about SSHKEYGEN.

TCPWARE_SSH2_HOSTKEY_DIR:HOSTKEY.PUB

Contains the public part of the host key. This file should be world-readable but writeable 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 writeable 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. 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.

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 host keys of hosts that a user wants to log in from using "host-based" 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 host key 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 public key algorithm of the key (`ssh-dss` or `ssh-rsa`).

For example flower.example.com's host key algorithm is `ssh-dss`. The host key would then be `flower_plants_com_ssh-dss.pub` in the `[.SSH2.KNOWNHOSTS]` directory.

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 public key 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 public keys to perform certain operations.

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 `TCPWARE_SPECIFIC:[TCPWARE]`.

SSH_EXE
Points to the directory where SSH executables are kept. Normally, this is `TCPWARE_COMMON:[TCPWARE]`.

SSH_LOG
Points to the directory where the log files are kept. Normally, this is `TCPWARE_SPECIFIC:[TCPWARE.SSH]`.

TCPWARE_LOG_SSH_MBX
Points to the OpenVMS mailbox used to log connection accept and reject messages. This must not be modified by the user.

TCPWARE_SSH_ACCESS_AUTHORIZATION
This logical makes user authentication checking will take place separately from action authorization checking. The value of the logical will be used to determine whether or not the desire action is allowed at this time. The value of the logical should be a string of the format:

`SHELL=`*`METHOD`*`,EXEC=`*`METHOD`*`,SUBSYSTEM=`*`METHOD`*

where *`METHOD`* is one of `NETWORK`, `LOCAL`, `REMOTE`. If one of `SHELL`, `EXEC`, or `SUBSYSTEM` is omitted, then that type of access will not be allowed at all.

TCPWARE_SSH_ACCESS_CHECK_OLD_STYLE
Use the `NETWORK` access mask instead of the `REMOTE` access mask to determine access during login.

TCPWARE_SSH_ACCESS_USE_LOCAL
Use the `LOCAL` login mask to allow logins instead of the `NETWORK` or `REMOTE` masks in SYSUAF.

TCPWARE_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 `@TCPWARE:CNFNET SSH`, and should not be modified directly by the user.

TCPWARE_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. This logical is set by `@TCPWARE:CNFNET SSH`. A successful connection request doesn't equate to a successful authentication request. This logical should not be modified directly by the user.

TCPWARE_SSH_LOG_FILE
Define the default file specification for the SSH server log file. The default is SSH_LOG:SSHD.LOG. The string %D will insert the date in the file name string and the string %N will insert the SCS node name in the file name string.

TCPWARE_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. This logical is set by @TCPWARE:CNFNET SSH. This logical should not be modified directly by the user.

TCPWARE_SSH_MAX_SESSIONS
Defines the maximum number of concurrent SSH sessions allowed on the server system. This is defined with the CNFNET procedure. If not defined, this defaults to 1000.

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.

TCPWARE_SSH_KEYGEN_MIN_PW_LEN
Defines the minimum passphrase length when one is to be set in SSHKEYGEN. If not defined, defaults to zero.

TCPWARE_SSH_PARAMETERS_n
These values are set by TCPware and must not be modified by the user.

TCPWARE_SSH_USE_SYSGEN_LGI
If defined, causes SSHD to use the VMS SYSGEN value of LGI_PWD_TMO to set the login grace time, overriding anything specified in the command line or the configuration file.

TCPWARE_SSH_ENABLE_SSH2_CONNECTIONS
Enables SSHD Master to accept SSH V2 sessions.

TCPWARE_SSH2_HOSTKEY_DIR
Directory containing the host keys for the SSH V2 server. Normally set to
`TCPWARE_SPECIFIC:[TCPWARE.SSH2.HOSTKEYS]`

TCPWARE_SSH2_KNOWNHOSTS_DIR
Directory containing the public keys for known systems. Normally set to
`TCPWARE_SPECIFIC:[TCPWARE.SSH2.KNOWNHOSTS].`

SSH2_DIR
Contains all SSH V2-specific files, such as configuration files. Normally set to
`TCPWARE_SPECIFIC:[TCPWARE.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.

25. Intrusion Prevention System (IPS)

This chapter describes the TCPware 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 TCPware 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 TCPware customers can incorporate the IPS functionality into user-written applications.

IPS is implemented by instrumenting components (e.g, TCPware SSH or FTP, or user-supplied components) with a Process Software-supplied API that allows them to report events, such as invalid login attempts, to the filter server process. The filter server, started when TCPware 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 TCPware kernel to filter that traffic.

IPS Operation

All the operating parameters such as the definition of the rules, 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.

Configuring IPS

IPS is configured in two steps:

1. Enabling IPS and configuring the main process-specific parameters of the filter server (for example, the size of the mailbox used by applications to communicate with the filter server).
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.

Configuring Process-Specific Parameters

The TCPware CNFNET procedure is used to enable and disable IPS, as well as to configure the process-specific parameters for the filter server. The process-specific parameters that are set/modified are the size of the filter server's mailbox and some of the process-specific quotas for the filter server process. These parameters set the following logical names:

- `TCPWARE_FILTER_SERVER_TQELM`
- `TCPWARE_FILTER_SERVER_ASTLM`
- `TCPWARE_FILTER_SERVER_MBX_MSGS`

Determining the Correct Filter Server Process Quotas

It is important to correctly determine the correct process quotas for the filter server. 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 the 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 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 ASTLM and TQELM values default to 500.

Determining the Correct Filter Server 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 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 (`NETCU SET IPS /STOP`).
2. Stop all applications using IPS (e.g., telnet sessions, ftp session, etc.).
3. Delete the old mailbox by running `TCPWARE:DELMBX.EXE`.
4. Start IPS (`NETCU SET IPS/START`).
5. Start any other applications previously stopped.

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 TCPware 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 `TCPWARE:` directory when TCPware is installed. Once these have been copied and modified, the filter server configuration may be reloaded via the `NETCU 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
- RLOGIN_FILTER_CONFIG.TEMPLATE
- RSHELL_FILTER_CONFIG.TEMPLATE

The following table lists the main configuration file keywords. These are found in the file TCPWARE:FILTER_SERVER_CONFIG.TXT:

Keyword	**Default**	**Description**
BLOCK_AT_DESTINATION_PORT	NO	If set to YES, indicates that all filters generated by the filter server will be for a specific destination port (the equivalent of a filter line of "EQ *port*"). 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 numbers mean more debug. This value should ordinarily never be set above 4 without direction from Process Software.
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 which are valid for the VMS REPLY/ENABLE command.
SPECIFIC_TRAP_ID		An integer representing the specific trap value when filter logging events are sent via SNMP.

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 has the following format. The definition must begin with a COMPONENT keyword. Comment 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:

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 TCPware interface address) in CIDR format to check. 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 doesn't 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.
`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.log`*

`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 as used in a filter file, with one exception as noted below. **NOTE**: The source address & subnet mask and destination address & subnet mask for the prototype filter must be specified in CIDR format. This is to maintain consistency with the other address/subnet mask combinations in, for example, the `EXCLUDE_ADDRESS` keywords.
`RULE` *`rulename`*	component	The user-defined name for a rule.

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 of Process Software, as higher debug levels will
#          negatively impact the filter server (and possibly, system)
#          performance.  The debug messages will be found in the file
#          TCPware: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    tcpware: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 chapter 23 of the TCPware
#         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 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 defihe the individual configuration files
#          for each configured component that uses the filter server
#
#==============================================================================
#
include tcpware:ftp_filter_config.txt
```

```
include tcpware:imap_filter_config.txt
include tcpware:pop3_filter_config.txt
include tcpware:smtp_filter_config.txt
include tcpware:snmp_filter_config.txt
include tcpware:ssh_filter_config.txt
#
```

For this configuration:

- Debug will be reported at level 4 (this produces fairly detailed information, normally useful only when debugging a problem).
- Log messages will be logged to `TCPWARE: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 TCPware FTP, IMAP, POP3, SMTP, SNMP and SSH servers will be loaded.

Sample Component Configuration File

The following is a configuration file for the SSH component:

```
component ssh
    proto_filter "deny tcp 192.168.0.100/32 192.168.0.11/24 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  192.168.10.2/32
           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
```

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 255.255.255.0, respectively.

The third rule is also `ssh_authfailed`, but applies to events received as 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 than 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 255.255.255.0.

> **Note:** If a rule specifies a destination address for an interface which does not currently exist, events for that interface will be dropped until the interface becomes available.

The `component` keyword may occur exactly once in the configuration file. The following example shows a configuration file for component ftp for 5 interfaces (EIA-0, EIA-1, PSD-0, PSD-1, PSD-2):

```
=====================================================================
#
#       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 EIA-0 and EIA-1:
#
        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
        rule    ftp_userauth
                max_count       21
                delta_time      180
                filter_durations  300,600,1800,3600,-1
        rule    ftp_authfailed
                max_count       21
                delta_time      90
                filter_durations  300,600,1800,3600,-1
        rule    ftp_timeout
                max_count       21
                delta_time      90
                filter_durations  300,600,1800,3600,-1
#
#  For PSD-0:
#
        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 PSD-1:
#
```

```
        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 PSD-2:
#
        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      90
                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 4 interfaces. Specifically:

- Interfaces EIA-0 and EIA-1 will use identical rules because they didn't specify destination addresses within their rulesets, and the destination addresses for EIA-0 and EIA-1 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 has been specified for all interfaces. However, interface PSD-0 has specified a port of 1521, so that port will be used for PSD-0 only. All other interfaces will use the default port of 21.
- Interface PSD-1 has an `exclude_address` specified for net 192.168.0.0/24. All events for PSD-1 that originated from that source net will be excluded from being counted by IPS. All other interfaces will count events from that network.

> **Note:** Filters for TCPware pseudo device (PSD-*n*) interfaces will be set on the actual physical interface to which the pseudo device maps. However, the `destination_address` field in the rule definition must specify the address for the pseudo device.

Configuring IPS for Paired Network Interfaces

To configure IPS for a paired network 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's 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 (PSD-*nnn*) are not true interfaces, they cannot have filters set on them.
- When a filter is created because of events coming in via a pseudo device, the destination address shown in the filter (using the `NETCU SHOW FILTER` command) will show the destination address for the pseudo device.

When NETCU is used to perform any of the following tasks:

- Create a paired network interface set via `SET INTERFACE /COMMON_LINK`
- Stop an interface via `STOP/IP`
- Start an interface via `START/IP`

IPS is notified of the change being made. This allows the filter server 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 EWA-0, EWA-1, and PSD-0 where PSD-0's physical interface is EWA-1:

```
component ssh
    proto_filter "deny tcp 192.168.0.100/32 192.168.0.11/24 log"
    #
    # EWA-0's address
    #
    destination_address 192.168.0.70/32
    #
    # EWA-1's address
    #
    destination_address 192.168.0.71/32
    #
    # PSD-0'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 PSD-0:

```
RAPTOR_$
%%%%%%%%%%  OPCOM  29-OCT-2014 13:00:55.77  %%%%%%%%%%    (from node VOODOO at 29-
OCT-2014 13:00:59.12)
```

```
Message from user JOHNDOE on VOODOO
FILTER_SERVER: Filter queued on EWA-0 (192.168.0.70/32) at 29-OCT-2014 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-OCT-2020 13:00:59.12  END: 29-OCT-2020 14:00:59.12


RAPTOR_$
%%%%%%%%%%  OPCOM  29-OCT-2020 13:00:55.80  %%%%%%%%%%    (from node VOODOO at
29-OCT-2020 13:00:59.15)
Message from user JOHNDOE on VOODOO
FILTER_SERVER: Filter queued on EWA-1 (192.168.0.71/32) at 29-OCT-2020 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-OCT-2020 13:00:59.15  END: 29-OCT-2020 14:00:59.15


RAPTOR_$
```

Note some things illustrated above:

- Each physical address (EWA-0 and EWA-1) had a filter set on it.
- No filter was set on interface PSD-0 because it is a pseudo interface.
- The destination address for each event is that of interface PSD-0, since that was the source of the events that caused the filters to be set.

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-2020 10:33:19.74  %%%%%%%%%%
Message from user SYSTEM on TWEET
FILTER_SERVER: Filter queued on EIA-0 (192.168.0.16) at 16-MAY-2020
10:33:19.74
               Component: ssh, Rule: ssh_bogus_id
deny      tcp    192.168.0.16/32
                 192.168.0.0/24      eq 22
                 FLTSVR,LOG
                 START: 16-MAY-2020 10:33:19  END: 16-MAY-2020 10:38:19
TWEET_$
```

This message is in essentially the same format as that when a NETCU SHOW FILTER command is performed:

```
$ netcu show filter eia-0
TCPware(R) for OpenVMS Packet Filter List for EIA-0:

Logging to OPCOM at 5 second intervals (normal format)

             Source          Source   Destination     Destination
Action Proto Address/Mask    Port     Address/Mask    Port     Option       Hits
------ ----- --------------  ----     --------------- ----     ------       ----
deny     tcp 192.168.0.16             192.168.0.22
             192.168.0.255            255.255.255.255          FLTSVR,LOG     0

permit  ip   0.0.0.0                   0.0.0.0
             0.0.0.0                   0.0.0.0                  FLTSVR         0

$
```

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 TCPware to act as if a "deny everything" filter terminates the list. The permit ip filter will essentially prevent that problem from happening.

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 TCPware system as specified in Chapter 7 of the TCPware Management Guide.

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
```

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 `NETCU SHOW FILTER` *`interface`*`/EXTRACT_FILTER` command. For example:

```
$ netcu show filter eia-0
TCPware(R) for OpenVMS Packet Filter List for EIA-0:

Logging to OPCOM at 5 second intervals (normal format)

             Source          Source   Destination     Destination
Action Proto Address/Mask    Port     Address/Mask    Port     Option       Hits
------ ----- --------------  ----     --------------- ----     ------       ----
deny     tcp 192.168.0.16             192.168.0.22
             192.168.0.255            255.255.255.255          FLTSVR,LOG     0

permit  ip   0.0.0.0                   0.0.0.0
             0.0.0.0                   0.0.0.0                  FLTSVR         0

$
```

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 TCPware to act as if a `deny everything` filter terminates the list. The `permit ip` filter will essentially prevent that problem from happening.

```
$ netcu show filter eia-0/extract_filter=filter.txt
$ netcu show filter eia-0
TCPware(R) for OpenVMS Packet Filter List for EIA-0:

Logging to OPCOM at 5 second intervals (normal format)

             Source          Source   Destination     Destination
Action Proto Address/Mask    Port     Address/Mask    Port     Option       Hits
------ ----- --------------  ----     --------------- ----     ------       ----
deny     tcp 192.168.0.14             192.168.0.22      22
             255.255.255.255           255.255.255.0            FLTSVR,LOG     0

deny     tcp 192.168.0.45             192.168.0.22
             255.255.255.255           255.255.255.0                           0
permit   ip  0.0.0.0                   0.0.0.0
             0.0.0.0                   0.0.0.0                  FLTSVR         0

$ type filter.txt
#
#   FILTER.TXT
#
#   Generated 16-MAY-2020 10:51:31
```

```
#
#=================================================================
deny tcp 192.168.0.14 255.255.255.255 192.168.0.22 225.255.255.0 eq 22 start "16-
MAY-2014 10:33:19" end "16-MAY-2014 10:38:19"LOG
deny tcp 192.168.0.45 255.255.255.255.192.168.0.22 255.255.255.0
permit ip 0.0.0.0 0.0.0.0 0.0.0.0 0.0.0.0
$ <edit to remove the first (filter) line>
$ netcu set filter eia-0 filter.dat


$ netcu show filter eia-0
TCPware(R) for OpenVMS Packet Filter List for EIA-0:

Logging to OPCOM at 5 second intervals (normal format)

             Source          Source   Destination     Destination
Action Proto Address/Mask    Port     Address/Mask    Port     Option       Hits
------ ----- --------------  ----     --------------- ----     ------       ----
deny     tcp 192.168.0.45             192.168.0.22
             255.255.255.255          255.255.255.0                          0
permit   ip  0.0.0.0                  0.0.0.0
             0.0.0.0                  0.0.0.0                 FLTSVR         0

$
```

Configuring PMDF to use IPS on TCPware

The IMAP, POP3 and SMTP servers referred to in the configuration template files above refer to the TCPware 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 TCPWARE:*.TXT and modified as appropriate for your installation. Edit TCPWARE:FILTER_SERVER_CONFIG.TXT to add INCLUDE lines for these component files, and comment out the INCLUDE lines for the standard TCPware 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 MessageStore 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 TCPware `NETCU SET IPS/RESTART` command.

Controlling the Filter Server

The filter server is started at system startup time. However, it can be controlled using the `NETCU SET IPS` command. The valid commands and their uses are:

Command	Description
`/DEBUG_LEVEL=`*`level`*	Change the debug level for the server. See the description for the `debug` main configuration keyword.
`/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's 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 `NETCU SHOW IPS /CONFIG=`*`filename`* command. For example:

```
$ netcu 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-2014 10:00:12.41
                expires:    29-APR-2014 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
```

Filter Server Files

The following files are associated with the filter server:

TCPWARE: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 05/18/2021

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-2021 05:55:45.35

        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-2021 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's no more activity for it.

TCPWARE: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.

TCPWARE: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.

IPS Logicals

The following logical names are used by the `FILTER_SERVER` process within IPS: They are generally created and set by `CNFNET`, which is used to configure IPS running parameters, and each of these logical names must have the `/SYSTEM` attribute.

FILTER_SERVER_CONFIG
Defined by the user to be the name of the FILTER_SERVER configuration file.

TCPWARE_FILTER_SERVER_ASTLM
Defined by CNFNET, this defines the size of the ASTLM quota for the FILTER_SERVER process. This must not be set by hand; CNFNET must be used to modify the value of this logical name.

TCPWARE_FILTER_SERVER_MBX_MSGS
Defined by CNFNET, this defines the size of the FILTER_SERVER mailbox. If this value is set too low, event messages may be lost. This must not be set by hand; CNFNET must be used to modify the value of this logical name.

TCPWARE_FILTER_SERVER_QUOTA_CHECK_TIME
Defined and set by the system administrator, this defines the value, in seconds, of how often the FILTER_SERVER process checks its TQELM and ASTLM parameters. If more than 90% of the quote value for either of these parameters is exceeded, a warning message is output to OPCOM. This is useful for determining the proper settings for the TQELM and ASTLM quotas for the FILTER_SERVER process.

TCPWARE_FILTER_SERVER_TQELM
Defined by IPS_CONTROL, this defines the size of the TQELM quota for the FILTER_SERVER process. This must not be set by hand; CNFNET must be used to modify the value of this logical name.

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, except for 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's no limit on the number of rules the filter server can maintain; the limit depends 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 TCPware `NETCU SET IPS/RELOAD` command.

> **Note:** The filter server configuration may be reloaded multiple times without causing problems for the filter server.

Filter Server API

There are two calls available in the filter server API. The function prototypes are defined in `TCPWARE_COMMON:[TCPWARE.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 as zero.

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's registering MUST be an installed image, using `/OPEN` or `/SHARED`. It doesn't 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 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 `TCPWARE_COMMON:[TCPWARE]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);
}
```

26. PATHWORKS Support

Introduction

You can use TCPware as a transport for HP's PATHWORKS product running between the OpenVMS system and a personal computer (PC).

You do not need to perform special configuration steps on either the OpenVMS system or the PC to use TCPware with PATHWORKS. However, the following information about PATHWORKS and TCPware may be helpful.

PATHWORKS Version 4 Server

The OpenVMS system runs the PATHWORKS server software. It resides in the `SYS$COMMON:[PCSA]` directory. When you use TCPware and PATHWORKS, note the following:

- TCPware writes error logs to the `SYS$SYSROOT:[PCSA]` directory.
- Make sure that you start TCPware before starting PATHWORKS.
- Make sure that you use the following command when you start PATHWORKS:

```
$ @SYS$STARTUP:PCFS_STARTUP TCP
```

After you start PATHWORKS on your OpenVMS system:

- The OpenVMS DCL command `SHOW SYSTEM` lists two processes:
  - `NBNS`
  - `PCFS_SERVER`
- The `SHOW CONNECTIONS` command in TCPware Network Control Utility (NETCU) shows the following two PATHWORKS-related services:

ID [1]	RecvQ	SendQ	Local address...	Foreign address...	State...
BG*nn*	0	0	*.netbios-ssn	*.*	LISTEN
BG*nn*	0	0	*.netbios-ns	*.*	

[1] The actual values may be different

PATHWORKS Version 5 Server

The OpenVMS system runs the PATHWORKS server software. When you use TCPware and PATHWORKS, note the following:

- Make sure that you start TCPware before starting PATHWORKS
- Configure the PATHWORKS Server to use TCP/IP as a transport

After you start PATHWORKS on the OpenVMS system:

- The OpenVMS DCL command `SHOW SYSTEM` lists the PATHWORKS master process `PWRK$MASTER` and several other PATHWORKS processes.
- The TCPware `SHOW CONNECTIONS` command in NETCU shows the following PATHWORKS-related services:

ID [1]	RecvQ	SendQ	Local address...	Foreign address...	State...
`INET`*`nn`*	0	0	*.netbios-ssn	*.*	LISTEN
`INET`*`nn`*	0	0	*.netbios-ns	*.*	
`INET`*`nn`*	0	0	*.netbios-dgm	*.*	

[1] The actual values may be different

Troubleshooting

Version 4

If you cannot make a complete connection between the workstation and the OpenVMS system using PATHWORKS Version 4, do the following:

- Check the TCPWARE:HOSTS. file on the OpenVMS system. Make sure that the workstation's name and internet address are in the file.
- Do a SHOW SYSTEM on the OpenVMS system. Make sure that the NBNS and PCFS_SERVER processes are running; if not, start PATHWORKS on the OpenVMS system.
- Issue the NETCU SHOW CONNECTIONS command on the OpenVMS system. Verify that the *.netbios-ssn and *.netbios-ns services are listed.
- If these services are not listed, make sure that you start TCPware before PATHWORKS.
- Look in the SYS$SYSROOT:[PCSA] directory on the OpenVMS system, to see if any of the error log files contain useful information.
- On the workstation, check that the PROTOCOL.INI or LANMAN.INI file has the correct configuration information for your workstation.
- On the workstation, verify that the syntax of your USE or NET USE command is correct. (Note the use of uppercase characters.)
- Make sure that the service specified in the USE or NET USE command is listed in the SYS$SYSROOT:[PCSA]PCFS$SERVICE_DATABASE.DAT file on the OpenVMS system.

Version 5

If you cannot make a complete connection between the workstation and the OpenVMS system using PATHWORKS Version 5, do the following:

- Make sure TCPware knows how to map the workstation's host name to its internet address. Check this using NSLOOKUP *hostname*. NSLOOKUP should report the internet address for the host. If it does not, update your primary DNS server so it can resolve the workstation name (or add an entry for the workstation in the OpenVMS system's TCPWARE:HOSTS. file).
- Do a SHOW SYSTEM on the OpenVMS system. Make sure that the PWRK$MASTER, NETBIOS, and other PWRK$ processes are running. If they are not, start PATHWORKS on the OpenVMS system.
- Issue the NETCU SHOW CONNECTIONS command on the OpenVMS system. Verify that the *.netbios-ssn, *.netbios-ns, and *.netbios-dgm services are listed. If these services are not listed, make sure that you start TCPware before PATHWORKS.
- Look in the PWRK$ROOT:[LOGS] directory on the OpenVMS system to see if any of the error log files contain useful information. See specifically the file PWRK$KNBDAEMON_*node*.LOG, where *node* is the particular PATHWORKS node. This contains the TCP/IP binding information.
- On the workstation, check that the PROTOCOL.INI or LANMAN.INI file has the correct configuration information for your workstation.
- On the workstation, verify that the syntax of your USE or NET USE command is correct. (Note the use of uppercase characters.)

- Make sure that the service specified in the `USE` or `NET USE` command is available on the OpenVMS system.

27. Tunneling DECnet over IP

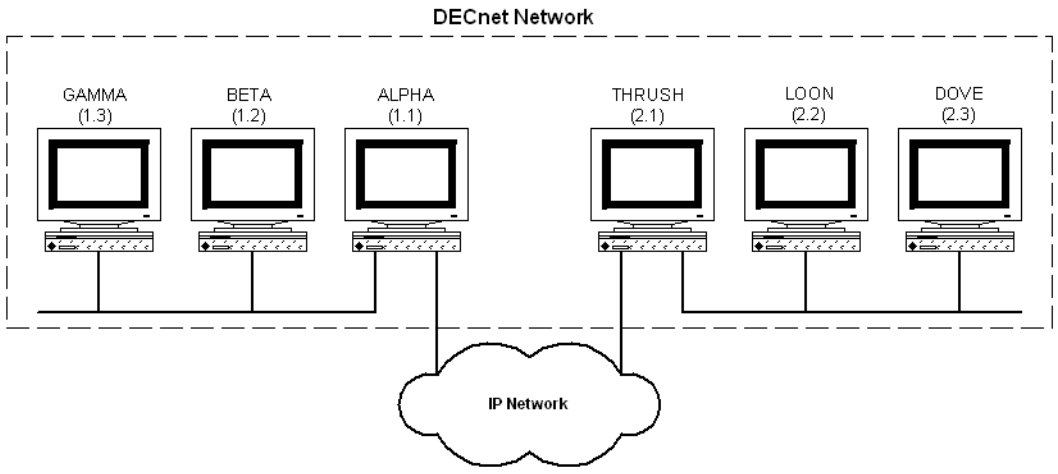
Introduction

This chapter describes how to set up and manage the TCPware for OpenVMS feature that supports tunneling of DECnet over IP-based networks.

DECnet over IP support provides tunneling of DECnet data link layer packets over a TCP/IP connection between two OpenVMS systems. Both systems must run TCPware for OpenVMS. This feature lets you establish a DECnet line and circuit between two OpenVMS systems connected to each other over a TCP/IP network.

> **Note:** This feature is intended for DECnet Phase IV. There is no need to use it with DECnet/OSI (DECnet Phase V). Instead, configure DECnet/OSI to run over TCP/IP as described in the DECnet/OSI documentation. No special TCPware configuration is required.

The below diagram shows an example of one of the many ways that you can use tunneling DECnet over IP. The example shows two DECnet areas connected together through a DECnet over IP tunnel. The shaded nodes, ALPHA and THRUSH, run both DECnet and TCPware for OpenVMS. The other nodes may run only DECnet.



From the system manager's point of view, DECnet over IP support consists of:

- CNFNET prompts that configure and start up DECnet over IP tunnels
- Options to `STARTNET` and `SHUTNET` that start and shut any DECnet over IP tunnels
- `NETCU SHOW DNIP` command to show information about any DECnet over IP tunnels
- Device driver `DNIPDRIVER`

DECnet over IP Lines

To use DNIP lines, you must install a DECnet routing license on the system, unless the DNIP line is the only DECnet line on the system.

DECnet over IP Tunnels

You invoke DECnet over IP configuration in one of two ways:

- During TCPware for OpenVMS configuration with CNFNET, by answering `YES` at the `Do you want to configure DECnet over IP tunnels?` prompt

  At any time after configuration, by invoking just the DECnet over IP portion of configuration by entering this command: `@TCPWARE:CNFNET DNIP`

  CNFNET prompts for the following information for each DECnet over IP tunnel:

- DECnet line name (for example, `DNIP-0-0`)
- Remote host name or IP address
- Port number of the tunnel on the local host
- Port number of the tunnel on the remote host (this is optional)

The port number is the TCP port number at which the system establishes DECnet over IP connections. You can reuse this same port number for all tunnels on a single system, and for all systems in your network. Note that the default value (64215) has no special significance, it is simply a "random" TCP port that any other service on your system is unlikely to use.

See the *Installation & Configuration Guide*, Chapter 4, *Configuring the TCP/IP Services*, the *Configure DECnet over IP Tunnels* section.

When you start TCPware, it automatically:

- Starts up the configured DECnet over IP tunnels.
- Performs DECnet configuration of the associated DECnet lines and circuits.

You need to perform all other DECnet network management through NCP, for example: defining DECnet node numbers and names.

Starting and Stopping

Normally, all configured DECnet over IP tunnels start when you start TCPware. If for any reason you want to shut down and later restart, only the DNIP tunnels (and not all of TCPware), use the following commands:

- To stop all DECnet over IP tunnels configured on your host: `@TCPWARE:SHUTNET DNIP`
- To start all DECnet over IP tunnels configured on your host: `@TCPWARE:STARTNET DNIP`

Status

Use the `SHOW DNIP` command in the Network Control Utility (NETCU) to show information about the currently configured DECnet over IP tunnels.

See the `SHOW DNIP` command of the *NETCU Command Reference*.

Troubleshooting

If you get the following message when configuring the DECnet over IP tunnel, you may have entered a host domain name instead of an IP address. It is possible that your `HOSTS.` file or DNS could not resolve the domain name. Try re-specifying the host address as an IP address:

```
%DNIP-E-INVIA, invalid internet address
```

28. X Display Manager Server

Introduction

TCPware's X Display Manager (XDM) server manages remote X displays (X terminals). When starting up, remote X displays communicate with XDM through the UDP-based X Display Manager Control Protocol (XDMCP). The XDM server creates a DECwindows login process that prompts remote X display users to log in and create a DECwindows session.

TCPware's XDM server supports broadcast, direct, and indirect requests. It also supports display authentication. A direct request is to a particular display or displays, and a broadcast request polls any number of displays for a response. An indirect request forwards the request to another XDM server to determine the X display. Display authentication requires that the XDM server and the X display share a private key.

The server is supported on all versions of the Alpha platform, all versions of the I64 platform and on VAX/VMS 6.1 or higher. VAX/VMS 5.5-2 is supported if DEC C or the DEC C Run-Time Library (RTL) is installed.

The XDM server is based on the X11R6.1 release from X Consortium.

This chapter describes the XDM server, its configuration, and how to impose access restrictions from other servers.

Installation and Initial Setup

Enable the XDM server through CNFNET configuration by responding with `YES` at the following prompts (both default to `NO`):

- `Do you want to use the TCPware for OpenVMS XDM server:`
- `Do you want to restart XDM:`

When enabled and started, the XDM server runs as a detached process, `TCPware_XDM`. A log file, `TCPWARE:XDMSERVER.LOG`, is created that captures log information enabled through the `DisplayManager.debugLevel` X Resource (see the next section).

Server Configuration

Configure the XDM server using an X Resource style configuration file, `TCPWARE:XDM_CONFIG.DAT`. You must create this file. (A template is provided as `TCPWARE:XDM_CONFIG.TEMPLATE`, which you can rename `TCPWARE:XDM_CONFIG.DAT`.)

The below example shows a sample configuration file.

```
# TCPWARE:XDM_CONFIG.DAT
#
DisplayManager.debugLevel:       1
DisplayManager.accessFile:       TCPWARE:XDM_ACCESS.DAT
DisplayManager.removeDomainname: false
DisplayManager.keyFile:          TCPWARE:XDM_KEYS.DAT
#
```

Any characters after the pound sign (`#`) are ignored. The XDM server supports the following X Resources:

- `DisplayManager.debugLevel`
- `DisplayManager.opcomLevel`

  If these resource values are set to non-zero, this enables logging. The default is `0` (no logging). If set to non-zero, the XDM server logs debugging information to the `TCPWARE:XDMSERVER.LOG` file or sends a message to OPCOM. The debug levels are as follows:

0	None	2	Critical	4	Warning	6	Informational
1	Alert	3	Error	5	Notice	7	Debug

  The defaults are:

  - `DisplayManager.debugLevel:4`
  - `DisplayManager.opcomLevel:5`

- `DisplayManager.accessFile`

  Specifies the access control file that defines access limitations of servers to the XDM server. By default no file is specified, which allows any X display to connect to the XDM server. If this resource is specified with a non-existent or empty file, no X display can connect to the XDM server. The usual file is `TCPWARE:XDM_ACCESS.DAT`. See *Server Access Control*.

- `DisplayManager.removeDomainname`

When computing the display name for XDMCP clients, the name resolver typically creates a fully qualified hostname for the terminal. As this may be confusing, you can use this resource to remove the domain name portion of the hostname, if it is the same as the domain name of the local host when this variable is set. The value is **true** by default.

- DisplayManager.keyFile

  XDM-AUTHENTICATION-1 style XDMCP authentication requires that a private key be shared between XDM and the terminal. This resource specifies the file containing those values. Each entry in the file consists of a display name and the shared key.

  The display name is either a unique name associated with the display hardware (such as -Ethernet-8:0:2b:a:f:d2 in the form *manufacturer-model-serial*) or any unique identifier. The shared key is a 56-bit integer represented by a 14-digit hexadecimal integer prefixed with **0x**.

  The below example shows a sample key file. (A template file, XDM_KEYS.TEMPLATE, is provided so that you can rename XDM_KEYS.DAT and use as the default.)

```
# TCPWARE:XDM_KEYS.DAT
#
# display name              shared key (14 digit hex)
# ------------------------  -------------------------
  -Ethernet-8:0:2b:a:f:d2   0x4f098af322dd98
#
```

Server Access Control

You can control access to the XDM server in the TCPWARE:XDM_ACCESS.DAT file by enabling the DisplayManager.accessFile resource in the XDM_CONFIG.DAT file (see *Server Configuration*). (A template file, XDM_ACCESS.TEMPLATE, is provided that you can rename XDM_ACCESS.DAT and use as the default.)

Note that if you specify the DisplayManager.accessFileresource without a filename value, any X display can connect to the XDM server. If the file specified is empty or nonexistent, no X display can connect to the XDM server.

The XDM_ACCESS.DAT file contains entries that control the response to direct and broadcast queries and contains separate entries for indirect queries. Each entry is a hostname, pattern, or macro.

- For a hostname, all comparisons involve network addresses, so that you can use any name that converts to the correct network address. An exclamation point (!) preceding a hostname excludes that host.

- A pattern involves wildcards, where only canonical hostnames are compared – do not attempt to match aliases. An exclamation point (`!`) preceding a pattern excludes hosts that match the pattern. A pattern includes one or more of the following wildcard meta-characters:

`*`	Matches any sequence of characters compared against the hostname
`?`	Matches any single character compared against the hostname

- A macro definition contains a macro name preceded by `%` and a list of hostnames or other macros, which can be nested.

An indirect entry also contains a hostname or pattern but follows it with a list of hostnames (such as defined by a macro) to which indirect queries should be sent. When checking access for a particular display host, each entry is scanned in turn and the first matching entry determines the response. Direct and broadcast entries are ignored when scanning for an indirect entry, and indirect entries are ignored when scanning for direct or broadcast entries. Other access control file formats are as follows:

Blank lines are ignored	
`#`	Comment delimiter, causing the rest of that line to be ignored
`\`	Line continuation character, allowing indirect host lists to span multiple lines

The below example shows a sample `XDM_ACCESS.DAT` file. Note that the positioning of entries is important. For example, if the `!bogus.example.com` and `*.example.com` lines had been transposed, direct or broadcast access from `bogus` would have been allowed, which is not desired. In the entry `!bogus.example.com dummy`, the `dummy` is a dummy host.

```
# TCPWARE:XDM_ACCESS.DAT
#
# Direct/broadcast query entries
#
bambi.example.edu     # allow access from bambi at Example U
!bogus.example.com    # disallow direct/broadcast service from bogus
*.example.com         # allow access from any display in example (x bogus)
#
# Indirect query entries
#
%HOSTS  expo.example.com  xenon.example.com  excess.example.com
kanga.example.com
extract.example.com  xenon.example.com  # force extract to contact xenon
```

```
!bogus.example.com   dummy            # disallow indirect access to bogus
*.example.com        %HOSTS           # all others get to choose from
#                                    hosts in the %HOSTS macro list
#
```

29. Network Testing Tools

Introduction

This chapter describes the following network testing tools included with TCPware:

ARPSNMP	Keeps track of ARP information and notifies of changes
CHARGEND	Sends strings of characters, ignoring data received
DAYTIMED	Sends the current date and time
DISCARD	Discards any data it receives
ECHOD	Returns any data it receives
FINGER	Extracts user information from a remote user information program
IDENT	Other servers use this to determine the user associated with a connection
NETCU DEBUG	TCPware's NETCU provides debugging commands
NSLOOKUP	Verifies DNS name server and host domain information
PEERNAME	Shows IP address and port information for a connection.
PING	Verifies that a host on the network is up and reachable
QUOTED	Checks for TCP connectivity between hosts
TCPDUMP	Tracks TCP packets over the network
TIME	Time service

TRACEROUTE	Tracks TCP routes over the network

ARPSNMP

ARPSNMP is a program that reads TCPware's ARP information and keeps track of information that it has seen in ARP.DAT. TCPWARE:ETHERCODES.DAT is used to display information about the manufacturer of the Ethernet address.

To use, create an empty ARP.DAT in your local directory and define the following symbol:

```
$ arpsnmp :== $tcpware:arpsnmp
```

Command syntax is as follows:

```
arpsnmp [-d] [-f file_specification]
```

-d	debug mode: information is displayed to the terminal and not mailed to SYSTEM
-f *file_specification*	specifies location for ARP.DAT if it isn't in the current working directory

The first time that ARPSNMP is run with an empty ARP.DAT, it reports all ARP entries as new as it adds them to the file, for example:

```
new station 192.168.0.69 aa:1:2:3:4:5
```

Each subsequent time that ARPSNMP is run, it compares the ARP table to the contents of ARP.DAT and reports any differences by sending email to the SYSTEM account (unless -d is specified on the command line).

CHARGEND

CHARGEND is the Character Generator Protocol server, defined in RFC 864. It sends strings of characters, ignoring data received, and is a useful debugging tool. CHARGEND can be TCP- or UDP-based. STARTNET starts the service automatically using the TCPWARE:MISC_CONTROL.COM file. The TCPWARE_COMMON:[TCPWARE.EXAMPLES] directory provides the source code. You can test this service through a TELNET connection to it.

DAYTIMED

DAYTIMED is the Daytime Protocol server, defined in RFC 867. It sends the current date and time as a character string without regard to input. It is a useful debugging and measurement tool. DAYTIMED can be TCP- or UDP-based. STARTNET starts the service automatically using the TCPWARE:MISC_CONTROL.COM file. The TCPWARE_COMMON:[TCPWARE.EXAMPLES] directory provides the source code. You can test this service through a TELNET connection to it.

DISCARD and DISCARDD

DISCARD is the Discard Protocol client and DISCARDD is the Discard Protocol server, defined in RFC 863. It simply throws away any data it receives, and is a useful debugging and measurement tool. It can be TCP- or UDP-based. STARTNET starts the service automatically using the TCPWARE:MISC_CONTROL.COM file. The TCPWARE_COMMON:[TCPWARE.EXAMPLES] directory provides the source code for DISCARD and DISCARDD. You can test the service using a TELNET connection to it.

ECHOD

ECHOD is the Echo Protocol server, defined in RFC 862. It returns any data it receives and is a useful debugging and measurement tool. It can be TCP- or UDP-based. STARTNET starts the service automatically using the TCPWARE:MISC_CONTROL.COM file. You can test this service using a TELNET connection to it.

FINGER and FINGERD

FINGER is the Finger User Information Protocol, defined in RFC 1288. It is used to extract user information from a remote user information program (RUIP). FINGER is TCP-based. STARTNET starts the service automatically using the TCPWARE:MISC_CONTROL.COM file. The TCPWARE_COMMON:[TCPWARE.EXAMPLES] directory provides the source code for FINGERD.

Use FINGER at the DCL level as follows:

```
$ FINGER user@host
```

The *user* is the user to extract information about on the specified *host*. You must use both parameters.

FINGERD reports the day of the week, hours, and minutes of a login time only if the user has been logged in for LESS than seven days. If the user has been logged on for MORE than seven days, the date in the format *dd-mm-yy* is reported.

IDENT

Other servers use the Identification server (formerly the Authentication Server) to determine the user associated with a connection. The MISC_CONTROL command procedure starts the IDENT server automatically.

NETCU DEBUG

TCPware's Network Control Utility (NETCU) includes DEBUG commands for the IP, TCP, and UDP layers. They display information about IP datagrams, TCP segments, and UDP datagrams sent and received over the network. You can use them to debug network problems.

See the *NETCU Command Reference* for the DEBUG IP, DEBUG TCP, and DEBUG UDP command descriptions.

NSLOOKUP

The NSLOOKUP utility sends test queries to a DNS name server to test the DNS configuration. You would use NSLOOKUP to find information such as the following for a selected host or domain:

- Default name server
- DNS database records, such as all, A, CNAME, and MX
- All the hosts in a domain
- User processes

The NSLOOKUP utility has two modes, noninteractive and interactive:

Noninteractive mode	Query a host and exit NSLOOKUP
Interactive mode	Query a host and enter further NSLOOKUP commands

Setting the mode depends on the syntax you use in entering the NSLOOKUP command. You access noninteractive mode if you use the following syntax at the DCL prompt:

```
$ nslookup host-to-find [server-to-query]
$
```

You access interactive mode if you use the following syntax at the DCL prompt:

```
$ nslookup [- server-to-query]
>
```

The difference is that noninteractive mode requires at least the *host-to-find* parameter, while interactive mode is signaled by either the lack of a parameter altogether or a dash preceding the *server-to-query* parameter. Also, after a noninteractive query, you are back at the DCL prompt, while the interactive query puts you at the > prompt so that you can enter further NSLOOKUP commands.

Both modes allow you to enter options, as described in *Setting Options*.

Noninteractive Mode

The full noninteractive mode syntax is as follows:

nslookup[*-option*[*-option...*]]*host-to-find* [*server-to-query*]

-option	Sets the state information for a group of lookups. (See *Setting Options* for details on state information.) You can have multiple options, each prefixed by a dash and separated by a space.
host-to-find	Hostname or IP address of the host about which to seek information.
server-to-query	Hostname or IP address of the DNS server to query. You can specify a server only if you also specify a *host-to-find*. If omitted, TCPware uses the current default server.

Enter the `nslookup` command and options in lowercase but enter the hostname and server name values in their proper case. To interrupt noninteractive mode, use **Ctrl+C**.

Interactive Mode

The full interactive mode syntax is as follows:

```
$ nslookup [-option[-option...]] [- server-to-query]
>{ host-to-find | command }
```

-option	Sets the state information for a group of lookups. (See *Setting Options* for details on state information.) You can have multiple options, each prefixed by a dash and separated by a space.
- server-to-query	Hostname or IP address of the DNS server to query. Precede the parameter with a dash and a space.
host-to-find	Hostname or IP address of the host about which to seek information, as entered at the `NSLOOKUP` prompt (>).
command	One of the query commands listed in the section. The command must be less than 256 characters; its case does not matter.

Enter the `nslookup` command and options in lowercase but enter the hostname or server name in its proper case. The `nslookup` command alone always returns the default server information before

presenting the > prompt, at which you can enter either the *host-to-find* or a *command*. To interrupt interactive mode, use **Ctrl+C**. To exit, enter **exit** or **Ctrl+Z**.

Entering Host-to-Find Names

You can enter the *host-to-find* as a hostname or an IP address. If you enter an IP address, the hostname is returned by default.

To look up a host not in the current domain, add a trailing period, such as:

```
$ nslookup iris.example.com.
```

If the trailing period is missing, the default domain name (as determined by the `set domain`, `set srchlist`, `set defname`, or `set search` commands) is appended.

See the **set** command description in the *Query Command Reference*.

Setting Options

`NSLOOKUP` options set the state information for the query. You can set options using the *-option* parameter, or in interactive mode using the `set` command. Each option corresponds to a `set` command keyword.

See the `set` command description for the keywords that correspond to the options.

For example, the following command uses *-option* parameters to set the `querytype` and `timeout` parameters for the next series of interactive lookups to server AMOS.

```
$ nslookup -querytype=hinfo -timeout=10 - amos
```

The alternative would be:

```
$ nslookup
> set querytype=hinfo
> set timeout=10
> amos
```

The following command tries to get information about node SIRIUS in domain EXAMPLE.COM by querying server ANDY. If unsuccessful, `NSLOOKUP` tries ten more times and returns to the DCL prompt.

```
$ nslookup -domain=example.com -retry=10 sirius andy
```

Query Command Reference

This section describes the NSLOOKUP query commands accessible at the > prompt. Enter the commands in lowercase or they are considered to be uppercase *host-to-find* entries.

---

exit

Exits NSLOOKUP.

Format

```
> exit
```

Enter the command in lowercase or it is considered to be hostname "EXIT".

Synonym

Ctrl+Z

Example

This example displays information about the server and exits NSLOOKUP.

```
$ nslookup
Default Server:  sirius.example.com
Address:  192.168.95.1
>exit
$
```

---

finger

Connects with the FINGER server on the (previously defined) host to display FINGER information. With the optional *username* parameter, you can focus the lookup on one user.

You can redirect the output from the finger command to a file using the greater-than symbol, as follows:

```
> finger> filename.ext
> finger>> filename.ext
```

The > places the output into *filename.ext*. The >> appends the output onto file *filename.ext*. When you direct output to a file, NSLOOKUP prints hash marks after every 50 records received from the server.

Format

```
> finger [username]
```

Enter the command in lowercase or it is considered to be hostname "FINGER".

Parameter

username
Focuses the lookup to one username. The *username* can be case-sensitive.

Example

This example defines the current host as SIRIUS and connects with the FINGER server on SIRIUS to display FINGER information on user GANDALF.

```
$ nslookup
>sirius
Default Server:  sirius.example.com
Address:  192.168.95.1

Name:    sirius.example.com
Address:  192.168.95.1

>finger gandalf
[sirius.example.com]
Status of VMScluster on node NENE at 23-JUN-2020 16:52:04.62
```

```
Username    Node      Name          New Mail    When         Terminal
GANDALF     SIRIUS    Gandalf                   3 Mon 08:34 FTA1211:
GANDALF     SIRIUS    Gandalf                   3 Mon 08:34 FTA1212:
```

---

help

Displays a brief summary of `NSLOOKUP` commands.

Format

```
> help
```

Enter the command in lowercase or it is considered to be hostname "HELP".

Synonym

`>?` (question mark)

Example

```
> help
                    NSLOOKUP COMMAND SUMMARY
      (identifiers are shown in uppercase, [] means optional)
Command         Meaning
------------------------------------------------------------------------
NAME          - print info about the host/domain NAME using default server
NAME1 NAME2   - as above, but use NAME2 as server
ls [opt] DOMAIN [> FILE] - list entities in DOMAIN (optional: output to
FILE)
  -a          - list canonical names and aliases
  -h          - list HINFO (CPU type and operating system)
  -s          - list well-known services
  -d          - list all records
  -t TYPE     - list records of the given type (e.g., A,CNAME,MX, etc.)
view FILE     - sort an 'ls' output file and view it
help or ?     - display this screen
spawn or ! [command] - create child process and execute command
finger [USER] - finger the optional NAME at the current default host
root          - set current default server to the root
server NAME   - set default server to NAME, using current default server
lserver NAME  - set default server to NAME, using initial server
set OPTION     - set an option
  all         - print options, current server and host
  [no]defname - append domain name to each query
  [no]recurse - ask for recursive answer to query
  [no]vc      - always use a virtual circuit
  domain=NAME - set default domain name to NAME
  srchlist=N1[/N2/.../N6] - set domain to N1 and search list to N1,N2, etc.
  root=NAME   - set root server to NAME
  retry=X     - set number of retries to X
  timeout=X   - set initial time-out interval to X seconds
```

```
  querytype=X - set query type, e.g., A,ANY,CNAME,HINFO,MX,NS,PTR,SOA,WKS
  type=X      - synonym for querytype
  class=X      - set query class to one of IN (Internet), CHAOS, HESIOD or
ANY
  [no]debug   - print debugging information
  [no]d2      - print exhaustive debugging information
exit          - exit the program, ^Z also exits
```

---

ls

Lists the information for a domain. The default output contains host names and their Internet addresses.

You can redirect the output from the `ls` command to a file as follows:

```
> ls...> filename.ext
> ls ...>> filename.ext
```

The > places the output into *filename.ext*. The >> appends the output onto file *filename.ext*. When you direct output to a file, NSLOOKUP prints hash marks after every 50 records received from the server.

Format

```
> ls [-option [-option...]] domain
```

Enter the command in lowercase or it is considered to be hostname "LS".

Parameters

-option
See the *Options* that follow for a list of available options. Always precede an option with a hyphen. Separate multiple options with spaces.

domain
Domain for which to list information.

Options

-a
Lists aliases of hosts in the domain. Synonym for `ls -t CNAME`.

-d
Lists all records for the domain. Synonym for `ls -t ANY`.

-h
Lists CPU and operating system information for the domain. Synonym for `ls -t HINFO`.

-s
Lists well-known services of hosts in the domain. Synonym for `ls -t WKS`.

-t [*type*]
Lists all records of the *type* specified by the `set type` command, as in the below table.

Value...	**Means...**
`any`	All records for the domain
`a`	Host internet address
`afsdb`	AFS database
`axfr`	Zone transfer
`cname`	Canonical name for an alias
`hinfo`	Host CPU and operating system type
`isdn`	ISDN information
`mb`	Mailbox
`mg`	Mail group
`minfo`	Mail information
`mr`	Mailbox rename
`mx`	Mail exchanger
`ns`	Name server for the named zone

ptr	Host name if query is Internet address, otherwise pointer to other information
rp	Responsible person
rt	Route through binding
soa	Domain "start-of-authority" information
txt	Text information
uinfo	User information
wks	Supported well-known services
x25	X.25 information

Examples

1. This example lists information for domain SIRIUS:

```
>ls example.com
[sirius.example.com]
 example.com.                  server = sirius.example.com

 sirius                        192.168.95.1
 example.com.                  server = nic.example.net
 nic.near.net.                 192.52.71.4
```

2. This example lists all records for domain SIRIUS:

```
>ls -d example.com
[sirius.example.com]
 example.com.    SOA   sirius.example.com leary.sirius.example.com
.(76 86400 1800 3600000 86400)
 example.com.    NS    sirius.example.com
```

3. This example appends the output in Example 2 onto an existing `FOOBAR.TXT` file:

```
>ls -d example.com >> foobar.txt
[sirius.example.com]
###
```

```
Received 165 records.
>
```

root

Changes the default server to the domain namespace root server.

The default host is `a.root-servers.net`.

You can change the name of the root server with the `set root` command.

Format

```
>root
```

Enter the command in lowercase or it is considered to be hostname "ROOT".

Synonym

```
>lserver a.root-servers.net
```

Example

This example lists the root server name and address.

```
>root
Default Server:  a.root-servers.net
Address:  198.41.0.4
```

---

server
lserver

Changes the default server to the specified host:

- The `server` command uses the current default server to look up information about the host.
- The `lserver` command uses the initial server to look up information about the host.

If `NSLOOKUP` cannot find an authoritative answer, it returns the names of servers that might have the answer.

Format

```
> server host
> lserver host
```

Enter the command in lowercase or it is considered to be hostname "SERVER" or "LSERVER".

Parameter

host
Domain name of the host.

Examples

1. This example changes the default server to GEMMA.

```
> server gemma
Default Server:  gemma.example.com
Served by:
- NIC.NEAR.NET
          192.168.71.4
          EXAMPLE.COM
- BU.EDU
          128.197.27.7
          EXAMPLE.COM
```

2. This example changes the default server back to SIRIUS.

```
> lserver sirius
Default Server:  sirius.example.com
Address:  192.168.95.1
```

---

set

Changes state information that affects the lookups.

Format

```
> set option
```

Enter the command in lowercase or it is considered to be hostname "SET".

The *options* are listed as under the Options heading. The abbreviated form of the option is underlined.

Synonym

```
$ nslookup -option [-option...]
```

Options

all

Prints the current value of all keyword options and information about the current default server and host.

class=*value*

Changes the query class to one of the values in the below table. The class specifies the protocol group of the information.

Value	Description
in	Internet class (default value)
chaos	Chaos class
hesiod	MIT Athena Hesiod class
any	Wildcard for any of the above

debug
Turns debug mode on. Debug provides information about the packet sent to the server and the resulting answer.

d2
Turns exhaustive debug mode on, which essentially displays all fields of every packet.

defname
nodefname
Appends (`defname`) the default domain name to every single-component lookup, or disables this function (`nodefname`). You should probably leave the append function on in most cases.

domain=*name*
Changes the default domain name to *name* and appends the default domain name to a lookup request, if `defname` and `searchare` set. The domain search list contains the parents of the default domain if it has at least two components in its name. The default is the local domain.

ignoretc
noignoretc
Ignores packet truncation errors (`ignoretc`) on output or disables this function (`noignoretc`). In most cases, you would want to display these errors. The default is `noignoretc`.

port=*number*
Changes the default TCP/UDP name server port to *`number`*.

querytype=*value*
Changes the type of information returned by a query to one of the values in the below table. The default *`value`* is `a`. The synonym is `type`.

Value	Description
`any`	All records for the domain

a	Host internet address
afsdb	AFS database
axfr	Zone transfer
cname	Canonical name for an alias
hinfo	Host CPU and operating system type
isdn	ISDN information
mb	Mailbox
mg	Mail group
minfo	Mail information
mr	Mailbox rename
mx	Mail exchanger
ns	Name server for the named zone
ptr	Host name if query is Internet address, otherwise pointer to other information
rp	Responsible person
rt	Route through binding
soa	Domain "start-of-authority" information
txt	Text information
uinfo	User information
wks	Supported well-known services

x25	X.25 information

retry=*number*
Sets the number of retries to *number*. If NSLOOKUP does not receive a reply to a request within the amount of time specified by set timeout, it resends the request. The retry value controls how many times NSLOOKUP resends a request before giving up. The default is 2 retries.

root=*host*
Changes the name of the root server to *host*. This changes the default root server when using the root command. The default is a.root.servers.net.

recurse
norecurse
The name server recursively queries other servers if it does not have the information (recurse), or does not do this (norecurse). You would normally want recursive queries, and this is the default.

search
nosearch
Searches for each name in parent domains of the current domain (search), or disables this function (nosearch). If the lookup request contains at least one period but does not end with a trailing period, search appends the domain names in the domain search list to the request until the server returns an answer. The default is search.

srchlist=*name1 [/name2 /... /name6]*
Sets up a search list by changing the default domain name to *name1* and the domain search list to *name1*, *name2*, and so on, up to six names, each separated by a slash (/). This command overrides the default domain name and search list of the set domain command.

timeout=*interval*
Changes the timeout interval NSLOOKUP uses while waiting for a reply to the *interval* number of seconds. The default is 10. (See also retry.)

type=*value*
See querytype.

vc
novc
Uses a TCP virtual circuit when sending requests to the server (vc), or uses UDP or allows NSLOOKUP to determine whether to use a virtual circuit based on the size of the request (novc). You would normally use the default of novc.

Examples

1. This example sets the search list to EXAMPLE.COM and COM.

```
>set srchl=example.com/com
```

2. This example sets the query type to ANY. When listing the host, all records appear.

```
>set q=any
>sirius
Server:  sirius.example.com
Address:  192.168.1.1

sirius.example.com   internet address = 192.168.1.1
sirius.example.com   CPU = VAXstation 4000-90   OS = VMS V5.5-2
SET SRCHL=EXAMPLE.COM/COMP.PROTOCOLS.SMTP
```

spawn

Spawns a subprocess to execute a DCL command.

> **Note:** You cannot `SPAWN` with `CAPTIVE` accounts.

Format

`> spawn [command]`

Enter the command in lowercase or it is considered to be hostname "SPAWN".

Synonym

`> ! [command]`

Parameter

command

Command to execute. If omitted, TCPware starts a DCL subprocess.

view

Sorts and lists previous `ls` command output redirected to a filename.

Format

`> view` *`filename`*

Enter the command in lowercase or it is considered to be hostname "VIEW".

Parameter

`filename`

Filename to which you redirected output during an `ls` command.

Example

This example redirects a listing output to the `FOOBAR.TXT` file, and views the contents of the file.

```
>ls sirius > foobar.txt
[sirius.example.com]
###
Received 165 records.
>view foobar.txt
[sirius.example.com]
 example.com.           server = sirius.example.com
 sirius                 192.168.95.1
 example.com.           server = nic.near.net
 nic.near.net.          192.52.71.4
```

NSLOOKUP Utility Error Messages

If the lookup request was not successful, NSLOOKUP prints an error message. Valid error messages are:

`Timed-out`	The server did not respond to a request after the amount of time specified in `timeout` and the number of retries specified in `retry`.
`Non-existent domain`	The host or domain name does not exist.
`No response from server`	The server machine does not run a name server.
`No records`	The server does not have resource records of the currently specified query type for the host, although the host name is valid.
`Connection Refused or Network Is Unreachable`	`NSLOOKUP` could not make a connection to the name or finger server. Common error with `finger` and `ls` requests.
`Server failure`	The name server found an internal inconsistency in its database and could not return a valid answer.
`Refused`	The name server refused to service the request.
`Format error`	The name server found that the request packet was not in the proper format.

PEERNAME

PEERNAME is a utility that displays the local and remote IP addresses and ports for a given connection. PEERNAME can handle all kinds of network devices: INET, TCP, or BG.

To use, define the following symbol:

```
$ peername :== $tcpware:peername
```

Command syntax is as follows:

```
$ peername network-device
```

For example, if you have used TELNET to log into a system, you can use PEERNAME to obtain information about that connection:

```
$ peername TT:
        Local address: 192.168.0.69, port: 23
        Remote address: 192.168.0.119, port: 21895
```

PING

The PING utility tells you whether a host is up and whether you can reach it. The PING utility uses the ICMP echo and echo reply messages. To use the PING utility, you need BYPASS or SYSPRV privilege. Also, always run the PING utility from an account that has NETMBX privileges.

Using PING

Before using the `PING` utility, enter the following:

```
$ PING :== $TCPWARE:PING
```

To "ping" a host, enter:

```
$ PING [-rv] host [data-size [npackets]]
```

- **`-r`** - Do not route: do not use another gateway to reach the destination; the destination must be on a local network
- **`-v`** - Verbose mode, which displays information on an invalid response
- ***`host`*** - Hostname or internet address of the host to "ping"
- ***`data-size`*** - Size, in bytes, of the ICMP echo data
- ***`npackets`*** - Number of packets to send; if omitted, `PING` sends an infinite number of packets

To terminate or interrupt `PING`, enter `Ctrl+C`.

The below example shows a `PING` command and the resulting output. The `Ctrl+C` appears because the user interrupted the output with `Ctrl+C`.

```
NETCU> PING process-gw
PING process-gw.example.com (192.168.95.126): 56 data bytes
64 bytes from 192.168.95.126: icmp_seq=0. time=10.ms
64 bytes from 192.168.95.126: icmp_seq=1. time=0.ms
64 bytes from 192.168.95.126: icmp_seq=2. time=0.ms
64 bytes from 192.168.95.126: icmp_seq=3. time=0.ms

Ctrl+C
----process-gw.example.com PING Statistics----
4 packets transmitted, 4 packets received, 0% packet loss
round-trip (ms)  min/avg/max = 0/2/10
$
```

QUOTED

The Quote-of-the-Day service (QUOTED) is a useful tool to check for connectivity. QUOTED is a TCP-based character generator service. As implemented, the QUOTED server listens for TCP connections on TCP port 17. Once you establish a connection, the service sends a short message. The service then throws away any data it receives and closes the connection.

It is your responsibility to provide the quote and define the TCPWARE_QUOTE logical.

There is no specific syntax for the quote. The quote can have a total of 512 characters and is limited to the following characters:

- ASCII printable
- Space
- Carriage return
- Line feed

You need to define a system logical, TCPWARE_QUOTE to specify the quote for the server. The TCPWARE_QUOTE logical name can be either a string or a filename that includes the quote text. Prefix a filename with the @ sign and enclose the definition or filename in quotation marks.

You need SYSNAM or SYSPRV privileges to define the system-wide logical. The following examples show three different ways to define the TCPWARE_QUOTE logical:

```
$ DEFINE/SYSTEM/EXEC TCPWARE_QUOTE "Quote-of-the-day"
$ DEFINE/SYSTEM/EXEC TCPWARE_QUOTE "@SYS$MANAGER:QUOTE.TXT"
$ DEFINE/SYSTEM/EXEC TCPWARE_QUOTE "Today's quote is",-
_$ "@SYS$MANAGER:QUOTE.TXT"
```

To test QUOTED, TELNET to the host that has a defined quote-of-the-day, as follows:

```
$ TELNET host QUOTE
$ TELNET host 17
```

(Specifying QUOTE or 17 is identical since the QUOTE server port number is 17.)

For example, the following command does a TELNET operation to BARTLETT's quote-of-the-day server:

```
$ TELNET BARTLETT QUOTE
```

The system displays:

```
%TCPWARE_TELNET-I-TRYING, trying BARTLETT,quote (192.168.5.75,17) ...
%TCPWARE_TELNET-I-ESCCHR, escape (attention) character is "^
"Quote-of-the-day"
```

If the system logical is not defined, the system displays:

```
%TCPWARE_TELNET-I-TRYING, trying BARTLETT,quote (192.168.5.75,17) ...
%TCPWARE_TELNET-I-ESCCHR, escape (attention) character is "^
No quote-of-the-day is currently defined.
```

TCPDUMP

TCPDUMP is a useful mechanism for tracking TCP packets by displaying information in the packet headers. You can specify the type of packet information to extract by including the relevant options and expressions.

Use of TCPDUMP assumes a thorough understanding of the TCP protocol.

Before using TCPDUMP, enter the following foreign command definition:

```
$ TCPDUMP :== $TCPWARE:TCPDUMP
```

The TCPDUMP command syntax is as follows:

```
$ TCPDUMP [options] [expressions]
```

You can also use the TCPDUMP command on the Network Control Utility (NETCU) level. This allows you to use OpenVMS qualifiers in place of (or in addition to) UNIX-style options. To use TCPDUMP on the NETCU level, enter the following:

```
$ NETCU TCPDUMP [qualifiers | options] [expressions]
```

For complete details on the available options and expressions, see *TCPDUMP Command Reference* later in this chapter.

Interpreting TCPDUMP Output

The output of TCPDUMP is protocol-dependent. The following gives a brief description and examples of most of the various output formats. The description assumes familiarity with RFC 793, *Transmission Control Protocol*.

Monitoring TCP Packets

Consider the command and its output in the below example:

```
$ tcpdump host bart
Getting stats.
tcpdump: listening on ESA0:
18:21:59.000400 bart.example.com.1023 > marge.login: S
560913442:560913442(0) win 24576 <mss 1460,wscale 0,eol> (DF)
18:21:59.004000 bart.example.com.1023 > marge.login: . ack 513152385
win 24576 (DF)
18:21:59.004100 bart.example.com.1023 > marge.login: P 0:1(1) ack 1 win 24576 (DF)
```

The general format of a TCP protocol line is as follows, with references to the above example in the explanations given in parentheses:

```
timestamp  src>dst:  flags [ first-byte:last-byte  (total-bytes)  ack seqno
win bytes  urg  <options>]
```

timestamp (18:21:59.000400)	By default, all output lines are preceded by a time stamp, the current clock time in the form *hours*:*minutes*:*seconds*.*decimal*. The time stamp reflects the time the driver delivers the packet. No attempt is made to account for the time lag between when the Ethernet interface removed the packet from the wire and when the driver services the I/O request. The additional `-tt` option you can use with the TCPDUMP command displays an unformatted time stamp, while the `-t` option removes the time stamp altogether.
src>*dst*: (bart.example.com.1023 > marge.login:)	Source and destination IP address and port.
flags - `S` (SYN), `F` (FIN), `P` (PUSH), `R` (RST), or `.` (no flag)	Transition flags that can appear in combination.
first-byte:*last-byte* (*total-bytes*) (560913442:560913442(0))	Packet data sequence number, the first byte followed by the last byte of data, and the total bytes.
ack *seqno* (ack 513152385)	Sequence number of the next data expected in the other direction on this connection.
win *bytes* (win 24576)	Number of bytes of receive buffer space available in the other direction on this connection.
`urg` (not shown)	The packet contains "Urgent" data.
<*options*> (not shown)	TCP options, enclosed in angle brackets (<>).
(`DF`**)** (various lines)	The IP do-not-fragment flag is included with the packet.

The *src* and *dst* values and the flags (S, F, P, R, or .) are always present. The other fields depend on the contents of the packet's TCP protocol header and appear only if appropriate.

The below example shows the opening portion of an RLOGIN operation from BART to MARGE.

```
bart.1023 > marge.login: S 768512:768512(0) win 4096 <mss 1024>
marge.login > bart.1023: S 947648:947648(0) ack 768513 win 4096 <mss 1024>
bart.1023 > marge.login: .ack 1 win 4096
bart.1023 > marge.login: P 1:2(1) ack 1 win 4096
marge.login > bart.1023: . ack 2 win 4096
bart.1023 > marge.login: P 2:21(19) ack 1 win 4096
marge.login > bart.1023: P 1:2(1) ack 21 win 4077
marge.login > bart.1023: P 2:3(1) ack 21 win 4077 urg 1
marge.login > bart.1023: P 3:4(1) ack 21 win 4077 urg 1
```

Here is the explanation for each line in the example output:

- `bart.1023 > marge.login: S 768512:768512(0) win 4096 <mss 1024>`

  TCP port 1023 on BART sends a packet to port LOGIN on MARGE. The `S` indicates that the SYN flag was set. The packet sequence number is 768512 and it contains no data. There is no piggybacked ACK, the available receive window is 4096 bytes, and there is a maximum segment size (MSS) option requesting an MSS of 1024 bytes.

- `marge.login > bart.1023: S 947648:947648(0) ack 768513 win 4096 <mss 1024>`

  MARGE replies with a similar packet except that it includes a piggy-backed ACK in response to BART's SYN.

- `bart.1023 > marge.login: . ack 1 win 4096`

  BART then sends an ACK in response to MARGE's SYN. The `.` means that no flags were set. The packet contains no data, so there is no data sequence number. Note that the ACK sequence number is a small integer (1). The first time TCPDUMP sees a TCP "conversation," it displays the sequence number from the packet.

- `bart.1023 > marge.login: P 1:2(1) ack 1 win 4096`
  `marge.login > bart.1023: .ack 2 win 4096`

  On subsequent packets of the conversation, the difference between the current packet's sequence number and this initial sequence number is displayed. This means that sequence numbers after the initial one can be interpreted as relative byte positions in the conversation's data stream (with the first data byte in each direction being `1`). (The `-s` option would override this feature, causing the original sequence numbers to be output.)

- `bart.1023 > marge.login: P 2:21(19) ack 1 win 4096`

  BART sends MARGE 19 bytes of data (bytes 2 through 20 in the BART-to-MARGE side of the conversation). The `P` (PUSH) flag is set in the packet.

- `marge.login > bart.1023: P 1:2(1) ack 21 win 4077`

  MARGE indicates that it received data up to, but not including, byte 21. Most of this data is apparently sitting in the socket buffer, since MARGE's receive window decreased by 19 bytes. MARGE also sends one byte of data to BART in this packet.

- ```
  marge.login > bart.1023: P 2:3(1) ack 21 win 4077 urg 1
  marge.login > bart.1023: P 3:4(1) ack 21 win 4077 urg 1
  ```

  MARGE sends two bytes of urgent, pushed data to BART.

Monitoring UDP Packets

The UDP format is illustrated by this rwho packet:

```
molly.who > broadcast.who: udp 84
```

This says that port WHO on host MOLLY sent a UDP datagram to port WHO on host BROADCAST, the Internet broadcast address. The packet contained 84 bytes of user data.

Some UDP services are recognized (from the source or destination port number) and the higher-level protocol information is displayed; in particular, Domain Name System (DNS) service requests and Remote Procedure Calls to the Network File System (NFS).

Displaying Link Level Headers

The `-e` option of the TCPDUMP command displays the link level header of each dump line. On Ethernet systems, this displays the source and destination addresses, protocol, and packet length. For example, the following command displays the additional information shown in **bold** type:

```
$ tcpdump -e host bart -t
Getting stats.
tcpdump: listening on ESA0:
aa:0:4:0:3a:8 aa:0:4:0:15:a ip 62: bart.1023 > marge.login: S
176494692:176494692(0) win 24576 <mss 1460,wscale 0,eol> (DF)
aa:0:4:0:3a:8 aa:0:4:0:15:a ip 60: bart.1023 > marge.login: . ack
128739885 win 24576 (DF)
aa:0:4:0:3a:8 aa:0:4:0:15:a ip 60: bart.1023 > marge.login: P
0:1(1) ack 1 win 24576 (DF)
```

The link level headers begin with `aa`. (Note that time stamps were omitted since the `-t` option was used.)
```

On FDDI networks, the `-e` option displays the frame control field – the source and destination addresses, and the packet length – that governs the interpretation of the rest of the packet.

Normal packets (such as those containing IP datagrams) are asynchronous, with a priority value between 0 and 7; for example, `async4`. Such packets are assumed to contain an 802.2 Logical Link Control (LLC) packet; the LLC header is displayed if it is not an ISO datagram or so-called Subnetwork Access Protocol (SNAP) packet.

Monitoring ARP and RARP Packets

The following description assumes familiarity with the ARP and RARP protocols. See Chapter 6, *Common Interfaces*, the *Address Resolution Protocol (ARP)* and *Reverse Address Resolution Protocol (RARP)* subsections for further information.

Address Resolution Protocol (ARP) and Reverse Address Resolution Protocol (RARP) output shows the type of request and its arguments.

The format is intended to be self-explanatory. The following abbreviated sample output is taken from the start of an RLOGIN operation from host MARGE to BART:

```
$ tcpdump host bart -t
arp who-has bart.example.com tell marge.example.com
arp reply bart.example.com is-at aa:0:4:0:1f:8
```

The first line indicates that MARGE sent an ARP packet asking for BART's Ethernet address. BART replied with its Ethernet address, aa:0:4:0:1f:8.

You can also specify not to convert IP addresses to hostnames by using the `-n` option:

```
$ tcpdump host bart -t -n
arp who-has 128.3.254.6 tell 128.3.254.68
arp reply 128.3.254.6 is-at aa:0:4:0:1f:8
```

With the additional link level header information, the output would be as follows:

```
$ tcpdump host bart -t -e
aa:0:4:0:1:8 aa:0:4:0:15:a ip 82: arp who-has 128.3.254.6 tell
128.3.254.68
aa:0:4:0:1:8 aa:0:4:0:15:a ip 82: arp reply 128.3.254.6 is-at aa:0:4:0:1f:8
```

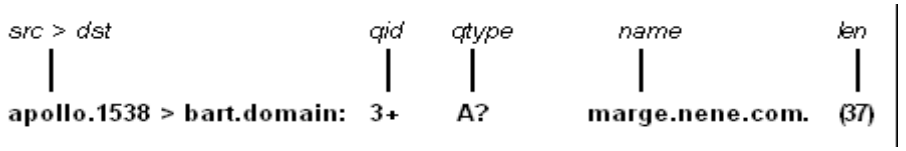
DNS Name Server Requests

The following description assumes familiarity with the DNS protocol. See Chapter 3, *Domain Name Services*, for further information.

Domain Name System (DNS) name server requests are formatted as:

src>dst: *qidop? Flags qtype qclassname* (*len*)

The below diagram shows an example in which APOLLO queries the domain server on BART for an address (type A) record associated with the name MARGE.NENE.COM.



- The source and destination hosts (*src >dst*) are given.
- The query id (*qid*) is 3 (the + indicates that the "recursion desired" flag was set).
- The query operation (*op?*) is omitted since it is the normal one, Query. If *op* had been anything else, it would appear between the 3 and the +.
- The query type (*qtype*) is A for Address record.
- The *qclass* is omitted since it is the normal one, C_IN. Any other *qclass* would have been displayed immediately after the A.
- The *name* is the domain name of target host MARGE.
- The query length (*len*) is 37 bytes, not including the UDP and IP protocol headers.

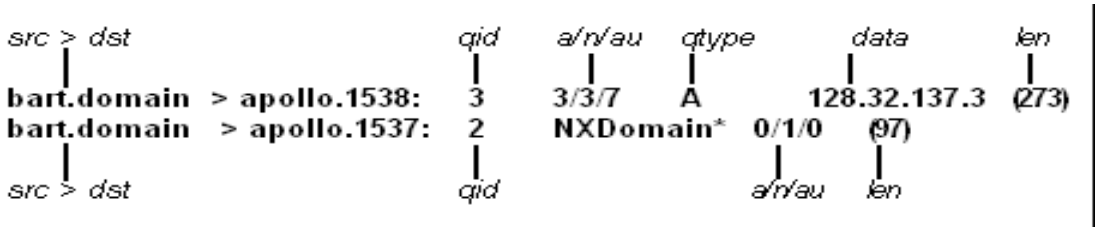
A few anomalies are checked and can result in extra fields enclosed in square brackets. If a query contains an answer, name server, or authority section, *ancount*, *nscount*, or *arcount* are displayed as [*n*a], [*n*n], or [*n*au], where *n* is the appropriate count. If any of the response bits are set (AA, RA, or *rcode*) or any of the "must be zero" bits are set in bytes two and three, [b2&3=*x*] is displayed, where *x* is the hex value of header bytes two and three.

DNS Name Server Responses

DNS name server responses are formatted as:

src> dst: *qid op rcode flags* a/n/au *qtype qclass data* (*len*)

The below diagram shows two examples.



In the first example, BART responds to query 3 from APOLLO with three answer records (a), three name server records (n), and seven authority records (au). The first answer record is type A (address) and its data is internet address 128.32.137.3. The total size of the response is 273 bytes, excluding UDP

and IP headers. The *op* (Query) and *rcode* (NoError) were omitted, as was the *class* (C_IN) of the A record.

In the second example, BART responds to query 2 with a response code of nonexistent domain (NXDomain) with no answers, one name server, and no authority records. The * indicates that the authoritative answer bit was set. Since there were no answers, there is no *qtype*, *qclass*, or *data* displayed.

Other flag characters that might appear are - (recursion available, RA, not set) and | (truncated message, TC, set). If the "question" section does not contain exactly one entry, [nq] is displayed.

Note that name server requests and responses tend to be large, and the default snapshot length of 68 bytes may not capture enough of the packet to display. Use the -s flag to increase the snapshot length if you need to seriously investigate name server traffic (-s 128 is known to be effective).

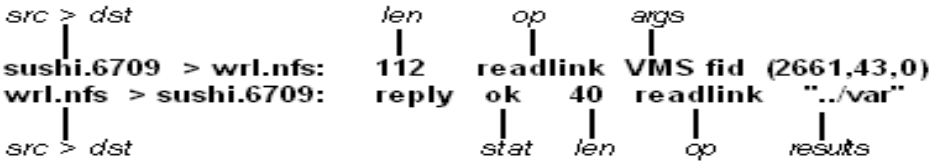
NFS Requests and Replies

The following description assumes familiarity with the Network File System (NFS). See Chapter 13, *NFS Client Management*, or Chapter 14, *NFS Server Management*, for further information.

NFS requests and replies are displayed as:

```
src.xid>dst.nfs: len op args
src.nfs >dst.xid: reply stat len op results
```

The below diagram shows an example. SUSHI sends a transaction with *id* 6709 to WRL (note that the number following the *src* host is the transaction ID, not the source port). The request was 112 bytes long, excluding the UDP and IP headers. The operation was a readlink (read symbolic link) on file handle (VMS fid) 2661,43,0. Host WRL replies ok with the contents of the link.



If you use the -v (verbose) option, additional information is displayed. For example:

```
sushi.1372a > wrl.nfs: 148 read VMS fid (2661,43,0) 8192 bytes @ 24576
wrl.nfs > sushi.1372a: reply ok 1472 read REG 100664 ids 417/0 sz 29388
```

In the first line, SUSHI queries WRL to read 8192 bytes from file handle 2661,43,0 at byte offset @24576. WRL responds with ok. The packet shown on the second line is the first fragment of the reply, and hence is only 1472 bytes long. (The other bytes follow in subsequent fragments, but these

fragments do not have NFS or even UDP headers and so might not be displayed, depending on the filter expression used.)

Because of the `-v` option, some of the file attributes returned in addition to the file data are displayed: the file type (`REG`, for regular file), the file mode (in octal, `100664`), the UID and GID (`417/0`), and the file size (`sz, 29388`). (The `-v` option also displays the IP header TTL, ID, and fragmentation fields, which were omitted from this example.) If you use `-v` more than once, even more details are displayed.

Note that NFS requests are very large and much of the detail is not displayed unless the snapshot length is increased. Try using `-s 192` to watch NFS traffic.

NFS reply packets do not explicitly identify the RPC operation. Instead, TCPDUMP keeps track of "recent" requests, and matches them to the responses using the transaction ID. If a response does not closely follow the corresponding request, it might be ignored.

TCPDUMP supports the following options/qualifiers enabling more decoding of RPC-based services.

R_RPC
-"R" all|udp|tcp
/RPC [=ALL|UDP|TCP]

For a UNIX-style option, the "R" must be uppercase and quoted.

The option/qualifier values are:

- `ALL` (default) decodes both UDP and TCP
- `UDP` decodes UDP only
- `TCP` decodes TCP only.

The following RPC protocols are decoded:

- For NFS:
  - Network Lock Manager (v.1)
  - Net Status
- For Portmapper:
  - Network Lock Manager (v.3)
  - Mount

The display with `/RPC=`*`qualifier`* used will look like this:
To display more detail, increase the size of `snapshot_size` using the `/SNAP` option.

```
$ netcu tcpdump /rpc=all /snap=8192 host flintstone
Tcpdump: listening on IPA0:
16:59:54.165987 test.example.com60e19a3d > construction.example.com.NFS:
```

```
udp(AUTH_UNIX: Machine=drilling.example.com,Uid=1111,Gid=22)
GETATTR(Fh=6070000.1000000.a00.2a8a 0200)
16:59:54.169893 test.example.com.NFS > test.example.com.60e19a3d:
udp GETATTR = OK,Type=DIR,Mode=040755,Nlink=3,Uid=1111,Gid=2,Size=3584,
Rdev=ffffa5a8,Blocks=8,Fsid=706,fileID=166442
```

IP Fragmentation

Fragmented Internet datagrams are displayed as:

(*fragid*:*size*@*offset*+)
(*fragid*:*size*@*offset*)

The first form, which includes the ending +, indicates that there are more fragments. The second form indicates the last fragment. The *id* parameter is the fragment ID, *size* is the fragment size (in bytes) excluding the IP header, and *offset* is this fragment's offset (in bytes) in the original datagram.

Information is displayed for each fragment. The first fragment contains the higher-level protocol header and the fragment information is displayed after the protocol information. Fragments after the first contain no higher-level protocol header, and the fragment information is displayed after the source and destination addresses. For example, here is part of an FTP from EXAMPLE.EDU to LBL-BART.ARPA over a CSNET connection that does not appear to handle 576-byte datagrams:

```
example.ftp-data> bart.1170: . 1024:1332(308) ack 1 win 4096 (frag
595a:328@0+)
example > bart: (frag 595a:204@328)
bart.1170 > example.ftp-data: . ack 1536 win 2560
```

- Addresses in the third line do not include port numbers, because the TCP protocol information is all in the first fragment, and it is not known what the port or sequence numbers will be when the later fragments are displayed.
- The TCP sequence information in the first line is displayed as if there were 308 bytes of user data when, in fact, there are 512 bytes (308 in the first fragment and 204 in the second). If you are looking for holes in the sequence space or trying to match up ACKs with packets, this can be misleading.

A packet with the IP do-not-fragment flag is marked with a trailing (DF).

TCPDUMP Command Reference

This section shows the format for and examples of the TCPDUMP command.

The first form of the TCPDUMP command is available on the DCL level as:

```
$ TCPDUMP:==$TCPWARE:TCPDUMP
$ TCPDUMP [options] [expressions]
```

The second form is on the Network Control Utility (NETCU) level. This form allows you to use OpenVMS qualifiers in place of (or in addition to) UNIX-style options. To use TCPDUMP on the NETCU level, enter the following:

```
$ NETCU TCPDUMP [qualifiers | options] [expressions]
```

The options and their qualifier equivalents are listed together.

TCPDUMP

TCPDUMP displays the headers of packets on a network interface that match the Boolean expression. The OpenVMS implementation currently works only with HP-compatible Ethernet cards. Some of the command line switches were changed from the UNIX version to support OpenVMS's case-insensitive command line.

PHY_IO privilege is required to use TCPDUMP, unless reading packets from a file. If using the TCPware drivers for packet capturing, LOG_IO and SYSPRV or BYPASS privileges are also needed.

Format

`TCPDUMP [ options | qualifiers] [expressions]`

Options and Qualifiers

> Only options are available if using TCPDUMP as a foreign command at the DCL prompt. You can mix and match options and qualifiers if using TCPDUMP as a NETCU command. In each case, the option is listed before its equivalent qualifier.

-a
/NO_RELATIVE_SEQUENCE_NUMBERS
Displays absolute, rather than relative, TCP sequence numbers. `/RELATIVE` is the default.

-b *bufcount*
/BUFFERS=*bufcount*
Sets the number of receive buffers for the Ethernet adapter to *`bufcount`*. The default is 255 on VAX, 175 on AXP and Itanium. You may need to lower the number if you receive the message `%SYSTEM-F-EXQUOTA, process quota exceeded`. This option does not apply to the TCPware drivers. Valid values are 1 to 255.

-c *exitcount*
/COUNT=*exitcount*
Exits after receiving *exitcount* packets.

-d
/HOSTNAMES=NOQUALIFIED
/NODOMAINS
Does not display host names as fully qualified domain names. For example, display `nic` instead of `nic.ddn.mil`. `/HOSTNAMES=QUALIFIED` is the default.

-e
/LINK_HEADERS
/ETHERNET_HEADER
Displays the link-level header on each dump line. (See *Displaying Link Level Headers*.)

-f
/HOSTNAMES=NOFOREIGN
/FOREIGN_NUMERICALLY
Displays "foreign" internet addresses numerically, not symbolically. `FOREIGN` is the default.

-g
/NETWORK
Translates network and broadcast addresses to domain names.

-i *interface*
/INTERFACE=*interface*
Listens on the *interface* interface. If unspecified, TCPDUMP searches for a configured interface (excluding loopback).

-j *file*
/FILTER_EXPRESSIONS=*file*
Uses the specified file as input for the filter expression. Any additional expressions given on the command line are ignored.

-k *packetype*
/PACKET_TYPE=*packetype*
Forces packets selected by the expression to be interpreted by the specified *packetype*. Currently known types are `rpc` (Remote Procedure Call), `rtp` (Real-Time Applications Protocol), `rtcp` (Real-Time Applications Control Protocol), `vat` (Visual Audio Tool), and `wb` (Distributed White Board).

-l
/LINE_BUFFERED
Makes `stdout` line buffered.

-n
/NOHOSTNAMES
/NUMERICALLY
Does not convert addresses (host addresses, port numbers, and so on) to names. (See *Monitoring ARP and RARP Packets* for an example.)

-o
/NOOPTIMIZER
Does not run the packet-matching code optimizer. This is useful only if you suspect a bug in the optimizer. The default is `/OPTIMIZER`.

-q
/SHOW=LESS
/QUIET
Quick output. Displays less protocol information so that output lines are shorter. The default is `/SHOW=NORMAL`.

-r *file*
/INPUT=*file*
/READ_BINARY=*file*
Reads packets from the file created with the `-w` option.

-"R" all|dup|tcp
/RPC [=ALL | UDP | TCP]
Displays RPC information. For UNIX-style options the "R" must be uppercase and in quotation marks.

-s *snapshotlength*
/DATA=*snapshotlength*
/SNAPSHOT_SIZE=*size*
Captures *snapshotlength* bytes of data from each packet, rather than the default of 68 bytes. 68 bytes is adequate for IP, ICMP, TCP, and UDP, but may truncate protocol information from name server and NFS packets. Packets truncated because of a limited snapshot are indicated in the output with [|*proto*], where *proto* is the name of the protocol level at which the truncation occurred.

If a snapshot of IP data is smaller than the actual packet size, the message truncated-ip - *n* bytes missing! appears. This is an informational message, and the data captured may be sufficient. If you need to see more data, use -s (or /DATA) to increase the snapshot length.

Note that taking larger snapshots both increases the amount of time it takes to process packets, and effectively decreases the amount of packet buffering. This can cause packets to be lost. You should limit *snapshotlength* to the smallest number that captures the protocol information you need.

-t
/TIMESTAMPS=*value*
/NOTIMESTAMPS
/NOTIME
Causes TCPDUMP to display a timestamp on each output line. Accepted values are UNIX, DELTA, and RELATIVE. The /NOTIMESTAMPS qualifier disables the timestamp. The default is /TIME=FORMAT.

-tt
/TIME=NOFORMAT
Displays an unformatted time stamp on each dump line. The default is /TIME=FORMAT.

-u
/OUTPUT=*file*
Redirects TCPDUMP output to an ASCII file.

-v
/SHOW=MORE
/VERBOSE

Verbose output. For example, the time-to-live (TTL) and type-of-service (TOS) information in an IP packet is displayed. (See *NFS Requests and Replies*.)

-vv
/SHOW=FULL
/FULL
"Hyperverbose" output. For example, additional fields are displayed from NFS reply packets.

-w *file*
/WRITE_BINARY=*file*
Writes the raw packets to a file, rather than parsing and displaying them out. The file can later be displayed with the `-r` option.

This file is written in libpcap format. When the interface specified is an Ethernet device the data in the file can be analyzed with Ethereal and similar tools.

-x
/HEXADECIMAL
Displays each packet (minus its link level header) in hexadecimal format. The smaller of the entire packet and *snapshotlength* byte values are displayed.

-z
/ASCII
Displays each packet in hexadecimal format with the ASCII equivalent.

Expressions

Recall that a TCPDUMP command consists of the following elements:

`TCPDUMP [ options | qualifiers] [expressions]`

Expressions select which packets to dump. Only packets for which the expression is true are dumped (or all packets if the expression is omitted). An expression breaks down into one or more primitives:

expression = primitive primitive ...

Primitives usually break down into a *qualifier* and an *address*:

primitive = *qualifier* address

Expression Qualifiers

There are three kinds of qualifiers – type, direction, and protocol:

- *Type* qualifiers define the ID name or number type. Possible types are:
    - host
    - net
    - port

  Examples of primitives using these qualifiers are `host foo`, `net 128.3`, and `port 20`. The default is `host` if the `type` qualifier is omitted, so that `foo` and `host foo` are equivalent. A host can be either an IP address or hostname. A **net** can either be a name from the `TCPWARE:NETWORKS.` file, or a network number, and can include the additional `mask` qualifier.

- Direction qualifiers specify a transfer direction, and are prepended to the *type* qualifier:
    - src
    - dst
    - src or dst (default)
    - src and dst

  Examples of primitives using these qualifiers are `src foo`, `dst net 128.3`, and `src` or `dst port ftp-data`. The default is `src or dst` if the direction qualifier is omitted, so that `port ftp-data` is true for a source or destination port.

- Protocol qualifiers restrict the match to a particular protocol and are prepended to the direction qualifier. Possible protocols are:
    - ether
    - fddi
    - icmp
    - ip
    - ipx
    - arp
    - rarp
    - decnet
    - lat
    - sca
    - moprc
    - mopdl
    - tcp
    - udp

Examples of primitives using these qualifiers are ether `src foo`, `arp net 128.3`, and `tcp port 21`. If the protocol qualifier is omitted, the default is all protocols consistent with the type. For example:

- `src foo` implies `ip`, `arp`, or `rarp`
- `net bar` implies `ip`, `arp`, or `rarp`
- `port 53` implies `tcp` or `udp`

You can be even more specific about the `ip` protocol by adding `proto` and the IP protocols `cmp`, `igrp`, `dp`, `nd`, or `cp`; for example, `ip proto cmp`. (Note that some protocols are preceded by backslashes to distinguish them from protocol qualifiers.) The same goes for the `ether` protocols `p`, `rp`, and `arp`, such as in `ether proto p`.

The `fddi` protocol qualifier is actually an alias for `ether`, and is treated identically as meaning the data link level used on the specified network interface. FDDI headers contain Ethernet-like source and destination addresses, and often contain Ethernet-like packet types, so you can filter on these FDDI fields just as with the analogous Ethernet fields. (FDDI headers also contain other fields, but you cannot name them explicitly in a filter expression.)

Other Expressions

There are additional special, "primitive" qualifiers that do not follow the pattern:

- `gateway`
- `broadcast`
- `less`
- `greater`

- arithmetic expressions

All of these are described in the *Primitives* subsection that follows.

More complex filter expressions are built up by using the words `and`, `or`, and `not` to combine primitives. For example, `host foo and not port ftp and not port ftp-data`. To save typing, identical qualifier lists can be omitted. For example, the following two expressions are identical, the first being the abbreviated form:

```
tcp dst port ftp or ftp-data or domain
tcp dst port ftp or tcp dst port ftp-data or tcp dst port domain
```

Expression Primitives

`dst [host]` *`host`*

IP destination field of the packet must be `host`.

src [host] *host*
IP source field of the packet must be *host*.

[host] *host*
IP source or destination of the packet must be *host*. Any of the above host expressions can be prepended with the keywords ip, arp, or rarp, as in ip host *host*, which is equivalent to ether proto ip and host *host*. If *host* is a name with multiple IP addresses, each address is checked for a match.

ether dst *ehost*
Ethernet destination address must be *ehost*, which is either a name from the TCPWARE:ETHERS. file, or a number.

ether src *ehost*
Ethernet source address must be *ehost*.

ether[host] *ehost*
Ethernet source or destination address must be *ehost*.

gateway *host*
Packet must use *host* as a gateway; that is, the Ethernet source or destination address was *host*, but neither the IP source nor destination was *host*. (This makes the expression equivalent to ether host *ehost* and not host *host*.) The *host* must be a name and must be found in both the TCPWARE:HOSTS. And TCPWARE:ETHERS. files.

dst net *net*
IP destination address of the packet must have a network ID of *net*.

src net *net*
IP source address of the packet must have a network ID of *net*.

net net
Either the IP source or destination address of the packet must have a network ID of *net*.

net *net* mask *mask*
IP address must match its network ID with the specified *mask*; can be qualified with `src` or `dst`.

net net/length
IP address must match its network ID with the specified mask *length*, in bits; can be qualified with `src` or `dst`.

dst port *port*
Packet must be IP/TCP or IP/UDP and have a destination port value, a number or name used in the `TCPWARE:SERVICES.` file. If a name, both the port number and protocol are checked. If a number or ambiguous name, only the port number is checked. For example, `dst port 513` displays both TCP/LOGIN and UDP/WHO traffic; `dst port domain` displays both TCP/DOMAIN and UDP/DOMAIN traffic.

src port *port*
Packet must have a source port value.

port *port*
Either the source or destination port of the packet must be *port*. Any of the above port expressions can be prepended with the keywords `tcp` or `udp`, as in `tcp src port` *port*, which matches only TCP packets whose source port is *port*.

less *length*
Packet must have a length less than or equal to *length* (equivalent to `len <=` *length*).

greater *length*
Packet must have a length greater than or equal to *length* (equivalent to `len >=` *length*).

ip proto protocol
Packet must be an IP packet of protocol type *protocol*, which can be a number or one of the names cmp, igrp, dp, nd, or cp. Note that tcp, udp, and icmp must be escaped using a backslash (F0>) to distinguish them from qualifiers.

***[ether]* broadcast**
Packet must be an Ethernet broadcast packet. The ether keyword is optional.

ip broadcast
Packet must be an IP broadcast packet. It checks for both the all-zeros and all-ones broadcast conventions and looks up the local subnet mask.

***[ether]* multicast**
Packet must be an Ethernet multicast packet (shorthand for ether[0] & 1 != 0).

ip multicast
Packet must be an IP multicast packet.

ether proto *protocol*
Packet must be of Ethernet type *protocol*, which can be a number or name like p, rp, or F30@Z7@Lam>rarp. Note that these identifiers must be escaped using a backslash (\).

In the case of FDDI (such as fddi proto arp), the protocol identification comes from the 802.2 Logical Link Control (LLC) header, which is usually layered on top of the FDDI header. TCPDUMP assumes, when filtering on the protocol identifier, that all FDDI packets include an LLC header, and that the LLC header is in so-called Subnetwork Access Protocol (SNAP) format.

decnet src *host*
DECNET source address must be *host*, which can be an address of the form 10.123, or a DECNET hostname.

decnet dst *host*
DECNET destination address must be *host*.

decnet host *host*
Either the DECNET source or destination address must be *host*.

ip, arp, rarp, decnet
Abbreviation for `ether proto` *protocol* for one of these protocols.

ipx { src | dst } *host*
With `ipx` alone, packet must be an IPX packet. With `src` or `dst`, the packet must come from or go to an ethernet host.

lat, moprc, mopdl
Abbreviation for `ether proto` *protocol* for one of these protocols. Note that TCPDUMP does not currently know how to parse these protocols.

tcp, udp, icmp
Abbreviation for `ether proto` *protocol* from one of these protocols.

expr *relational-operator expression*
The relation must hold, where *relational-operator* is one of >, <, >=, <=, =, !=, and *expression* is an arithmetic expression composed of integer constants (expressed in standard C syntax), the normal binary operators (+, −, *, /, &, |), a length operator, and special packet data accessors (see the following subsection).

Accessing Data Within a Packet
To access data inside a packet, use the syntax:

proto[*expr* : *size*]

- *proto* is one of `ether`, `fddi`, `ip`, `arp`, `rarp`, `tcp`, `udp`, or `icmp`, and indicates the protocol layer for the index operation.
- *expr* is the byte offset, relative to the indicated protocol layer.
- *size* is optional and indicates the number of bytes in the field of interest. It can be either `1`, `2`, or `4`, and defaults to `1`. The length operator, indicated by the keyword `len`, gives the length of the packet.

For example:

- `ether[0] & 1 != 0` catches all multicast traffic.
- `ip[0] & 0xf != 5` catches all IP packets with options.
- `ip[6:2] & 0x1fff = 0` catches only unfragmented datagrams and fragment zero of fragmented datagrams. This check is implicitly applied to the TCP and UDP index operations. For instance, `tcp[0]` always means the first byte of the TCP header, and never means the first byte of an intervening fragment.

Combining Primitives

You can combine primitives using:

- A parenthesized group of primitives and operators (parentheses must be escaped)
- Negation (`!` or `not`) (you must enclose expressions using `!` in quotes)
- Concatenation (`&` or `and`)
- Alternation (`|` or `or`)

Negation has the highest precedence. Alternation and concatenation have equal precedence and associate left to right. Note that explicit `and` tokens, not juxtaposition, are required for concatenation.

If the identifier omits a qualifier, the most recent qualifier applies. For example,

`not host vs and ace` is short for `not host vs and host ace`, which should not be confused with `not (host vs or ace)`.

Expression arguments can be passed to TCPDUMP as either a single argument or as multiple arguments, whichever is more convenient.

Examples

1. These identical examples display all packets arriving at or departing from BART.

```
$ tcpdump bart
$ tcpdump host bart
```

2. These identical examples display traffic between HELIOS and either HOT or ACE.

```
$ tcpdump helios and (hot or ace)
$ tcpdump host helios and (hot or ace)
```

3. These identical examples display all IP packets between ACE and any host except HELOIS.

```
$ tcpdump ip ace and not helios
$ tcpdump ip host ace and not helios
```

4. This example displays all traffic between local hosts and hosts at Berkeley.

```
$ tcpdump net ucb-ether
```

5. This example displays all FTP traffic through gateway SNUP (note that the expression is quoted because of the parentheses).

```
$ tcpdump "gateway snup and (port ftp or ftp-data)"
```

6. This example displays traffic neither sourced from nor destined for local hosts (if you gateway to one other net, this information should never make it onto your local net).

```
$ tcpdump ip and not net localnet
```

7. This example displays the start and end packets (the SYN and FIN packets) of each TCP conversation that involves a nonlocal host.

```
$ tcpdump "tcp[13] & 3 != 0 and not src and dst net localnet"
```

8. This example displays IP packets longer than 576 bytes sent through gateway HOMER.

```
$ tcpdump "gateway homer and ip[2:2] > 576"
```

9. This example displays IP broadcast or multicast packets not sent over Ethernet broadcast or multicast.

```
$ tcpdump "ether[0] & 1 = 0 and ip[16] >= 224"
```

10. This example displays all ICMP packets that are not echo requests/replies (PING packets).

```
$ tcpdump "icmp[0] != 8 and icmp[0] != 0"
```

TIME

TIME is the Time Protocol, defined by RFC 868. For a TCP connection, TIME returns a four-byte integer (in network byte order) of the current time in number of seconds since 1-JAN-1900 00:00 GMT. For UDP, when you receive a packet time port (37), TIME returns a four-byte datagram with the time (and discards the received datagram).

TRACEROUTE

This section describes the TRACEROUTE utility, which traces the path of an IP packet to an internet host. Only one user can run TRACEROUTE at a time. TRACEROUTE requires BYPASS or SYSPRV privileges.

Before using TRACEROUTE, enter the following to define the TRACEROUTE foreign command definition:

```
$ TRACEROUTE :== $TCPWARE:TRACEROUTE
```

To trace the path an IP packet follows to an internet host, enter the following format:

```
$ TRACEROUTE [-dnrv] [-w wait] [-m max_ttl][-q nqueries]-
_$ [-t tos] [-s src_addr] host
```

-n	Specifies to display internet addresses, not host names
-r	Specifies not to route (TRACEROUTE displays only directly "connected" hosts)
-v	Specifies verbose mode
-w *wait*	Is the number of seconds to wait between probes
-m *max_ttl*	Is the maximum time-to-live to set in or outgoing datagrams
-q *nqueries*	Is the number of probes sent
-t *tos*	Is the type of service (the default is 0)
-s *src_addr*	Is the source internet address to use (the default depends on the interface needed to reach the destination)
host	Is the name of the internet host whose path you want to trace

The below example shows a TRACEROUTE command and the resulting output.

```
$ TRACEROUTE nic.ddn.mil
```

```
TRACEROUTE to nic.ddn.mil (192.112.36.5), 30 hops max, 38 byte packets
 1  delta.example.com (192.168.95.126)  0 ms  0 ms  0 ms
 2  process-gw.example.com (192.168.138.1)  0 ms  10 ms  0 ms
 3  waltham-cr1.bbnplanet.net (131.192.148.49)  20 ms  0 ms  0 ms
 4  cambridge2-cr3.bbnplanet.net (131.192.27.2)  10 ms  10 ms  10 ms
 5  cambridge2-br1.bbnplanet.net (199.92.129.6)  10 ms  10 ms  10 ms
 6  cambridge1-br2.bbnplanet.net (4.0.2.25)  10 ms  20 ms  10 ms
 7  nyc1-br2.bbnplanet.net (4.0.1.121)  20 ms  30 ms  20 ms
 8  nyc2-br2.bbnplanet.net (4.0.1.154)  30 ms  90 ms  120 ms
 9  nynap.bbnplanet.net (4.0.1.26)  20 ms  40 ms  20 ms
10  niprnet.sprintnap.net (192.157.69.45)  420 ms  90 ms  90 ms
11  137.209.12.1 (137.209.12.1)  90 ms  90 ms  90 ms
12  * 198.26.119.26 (198.26.119.26)  90 ms  100 ms
13  nic.mil (192.112.36.5)  90 ms
```

30. Configuring Precision Time Protocol (PTP)

The Precision Time Protocol (PTP) service is an implementation of PTPv2 (IEEE 1588-2008). PTP is used to synchronize when clocks advance (tick) on a LAN computer network. It was designed primarily for instrumentation and control systems. On a local area network PTP is capable of synchronizing the clocks to an accuracy in the sub-microsecond range. In contrast, NTP is capable of accuracy in the tens of microseconds range. This higher degree of accuracy is achieved by using the time that the packet is received by the IP stack as part of the calculation to determine the difference between clocks. This means that the time between when the packet is received and when it is processed by the user mode application (PTP) can be included in the calculations when making adjustments to the skew between clocks. PTP is available for Alpha and Integrity systems running OpenVMS V7 and later.

When a PTP implementation starts up it is in the LISTEN state and it waits for an announcement from other systems. If no announcement happens within the `ptpengine:Announce_Receipt_Timeout`, then the system may move to MASTER state if it is configured to allow this. When in MASTER state it will announce its clock and the quality for other systems to observe. A system operating in LISTEN or SLAVE state will use the announced clock quality to choose the best master clock (BMC) from the announcements received. A system that is announcing a clock will also listen for other announcements and will stop making announcements and use another system as its master clock if a better one is detected. The quality of the clock is determined by the time source, how far it is removed from the source and the configuration parameters. The system with the reference clock is referred to as the Grand Master (GM). A potential master that has stopped making announcements will resume announcements when it stops receiving announcements from better masters.

UDP multicast is generally used on the local network to find other clocks, determine the best source and synchronize clocks. Currently PTP uses IPv4 only.

It is reasonable to run both NTP and PTP on a system, with NTP set as the `ptpengine:ptp_timesource`. This is reflected in the time source portion of the packets to let other systems know the quality of the clock. If the PTP configuration has `ntpengine:enabled = Y`, and the NTP configuration has `enable mode7` in the configuration PTP will attempt to disable NTP. NTP can also be set to be a failover mechanism for when a PTP master cannot be found.

PTP determines the offset from UTC on startup from the VMS system logical SYS$TIMEZONE_DIFFERENTIAL and uses a VMS system lock to synchronize with NTP for when the TDF changes due to entering or leaving day light saving time. The PTP implementation does NOT have any code to change the VMS system logicals or offset when the system changes between standard time and day light saving time. Systems that observe day light saving time should use NTP to control that.

Comparing PTP and NTP

PTP uses a significantly higher amount of CPU time and does a lot more I/O than NTP. The goal of PTP is to keep the clock synchronized with the selected grand master. The goal of NTP is to keep the time accurate based upon the time that the configured servers report. NTP uses UTC as its time standard, PTP uses TAI (International Atomic Time), there is a file (TCPWARE:LEAP-SECONDS.LIST) that contains the leap seconds necessary to convert one to the other. NTP uses all configured clocks that respond with reasonable time and delay characteristics to determine what the correct time should be. PTP chooses a single master clock based upon the following reported information:

1. Priority One Field (ptpengine:priority1) This is a configuration file item and lowest wins. Normally set to 128 for a potential master and 255 for slave only.
2. Clock Class. This is based upon configured value (ptpengine:clock_class) and modified based upon actual operating state.
3. Clock Accuracy. This is based upon configuration (ptpengine:ptp_clock_accuracy) and computed accuracy scaled to an enumerated value.
4. Clock Variance. This is based upon configuration (ptpengine:ptp_allan_variance) and computed variance scaled to an enumerated value.
5. Priority 2 field (ptpengine:priority2) Configuration file item which can be used to identify primary and backup clocks among otherwise identical, redundant Grandmaster clocks.
6. Source Port ID. A unique value, generally the Ethernet MAC address.

PTP is not available on VAX systems or AXP V6.

Enabling PTP in TCPware

1. Enable the PTP server in TCPware's network configuration utility:

```
$ NETCU CONFIGURE/SERVER
TCPware Server Configuration Utility V6.1
[Reading in configuration from TCPWARE:SERVICES.MASTER_SERVER]
SERVER-CONFIG>ENABLE PTP
```

```
SERVER-CONFIG>EXIT
[Writing configuration to
TCPWARE_COMMON_ROOT:[TCPWARE]SERVICES.MASTER_SERVER]
$
```

2. Create `TCPWARE:PTPD2.CONF` with information from the *Configuration* section of this chapter.

3. Create `TCPWARE:ETHER.` with ethernet (MAC) addresses and names of possible masters. Each line has the format *`ethernet-address nodename`*

```
aa:00:04:00:01:01 ptp-master.example.com
```

4. Restart the TCPware master server:

```
$ @TCPWARE:START_SERVER RESTART
```

Configuration

Minimal Example

```
; ============================================================================
; This is a minimal configuration for a PTPv2 slave
; ============================================================================
; interface has to be specified
ptpengine:interface=se0
; PTP domain
ptpengine:domain=0
; available presets are slaveonly, masteronly and masterslave (IEEE 1588)
ptpengine:preset=slaveonly
; multicast for both sync and delay requests - use hybrid for unicast delay
; requests
ptpengine:ip_mode=multicast
; status file providing an overview of ptpd's operation and statistics
global:log_status=y
; required if ip_mode is set to hybrid
;ptpengine:log_delayreq_interval=0
; uncomment this to log a timing log
;global:statistics_file=tcpware:ptpd2.stats
; always keep a new line at the end
```

Available Configuration Options

ptpengine:interface = *interface*
Network interface to use - `se0`, `se1` etc. (required).

ptpengine:backup_interface = *interface*
Backup network interface to use - `se0`, `se1` etc. When no grandmaster is available, slave will keep alternating between primary and secondary until a grandmaster is found.

ptpengine:preset = slaveonly
PTP engine preset. Possible values are:

`none`	Defaults, no clock class restrictions
`masteronly`	Master, passive when not best master (clock class 0..127)
`masterslave`	Full IEEE 1588 implementation: Master, slave when not best master (clock class 128..254)
`slaveonly`	Slave only (clock class 255 only). Default setting.

ptpengine:ip_mode = multicast
IP transmission mode. Possible values are:

`multicast`	uses multicast for all messages (the default)
`hybrid`	uses multicast for sync and announce, and unicast for delay request and response
`unicast`	uses unicast for all transmission.

When unicast mode is selected, destination IP(s) need to be configured using `ptpengine:unicast_destinations`.

ptpengine:unicast_negotiation = N
Enable unicast negotiation support using signaling messages.

ptpengine:disable_bmca = N
Disable Best Master Clock Algorithm for unicast masters: Only effective for `masteronly` preset - all Announce messages will be ignored and the clock will transition directly into MASTER state.

ptpengine:unicast_negotiation_listening = N
When unicast negotiation enabled on a master clock, reply to transmission requests also in LISTENING state.

ptpengine:delay_mechanism = E2E
Delay detection mode used - use `DELAY_DISABLED` for synchronization only (no full synchronization).
Options: `E2E P2P DELAY_DISABLED`

ptpengine:domain = 0
PTP domain number.

ptpengine:port_number = 1
PTP port number (part of PTP Port Identity - not UDP port). For ordinary clocks (single port), the default should be used, but when running multiple instances to simulate a boundary clock, the port number can be changed.

ptpengine:any_domain = N
Usability extension: if enabled, a slave-only clock will accept masters from any domain, while preferring the configured domain, and preferring lower domain number.

NOTE: this behavior is not part of the standard.

ptpengine:slave_only = Y
Slave only mode (sets clock class to 255, overriding value from preset).

ptpengine:inbound_latency = 0
Specify latency correction (nanoseconds) for incoming packets.

ptpengine:outbound_latency = 0
Specify latency correction (nanoseconds) for outgoing packets.

ptpengine:offset_shift = 0
Apply an arbitrary shift (nanoseconds) to offset from master when in slave state. Value can be positive or negative - useful for correcting for antenna latencies, delay asymmetry and IP stack latencies. This will not be visible in the offset from master value - only in the resulting clock correction.

ptpengine:always_respect_utc_offset = N
Compatibility option: In slave state, always respect UTC offset announced by best master, even if the `currrentUtcOffsetValid` flag is announced `FALSE`.

NOTE: this behavior is not part of the standard.

ptpengine:prefer_utc_offset_valid = N
Compatibility extension to BMC algorithm: when enabled, BMC for both master and save clocks will prefer masters announcing `currrentUtcOffsetValid` as `TRUE`.

NOTE: this behavior is not part of the standard.

ptpengine:require_utc_offset_valid = N
Compatibility option: when enabled, ptpd2 will ignore Announce messages from masters announcing `currentUtcOffsetValid` as `FALSE`.

NOTE: this behavior is not part of the standard.

ptpengine:unicast_grant_duration = 300
Time (seconds) unicast messages are requested for by slaves when using unicast negotiation, and maximum time unicast message transmission is granted to slaves by masters

ptpengine:log_announce_interval = 1

PTP announce message interval in master state. When using unicast negotiation, for slaves this is the minimum interval requested, and for masters this is the only interval granted. (expressed as log 2 i.e. -1=0.5s, 0=1s, 1=2s etc.)

ptpengine:log_announce_interval_max = 5
Maximum Announce message interval requested by slaves when using unicast negotiation, (expressed as log 2 i.e. -1=0.5s, 0=1s, 1=2s etc.)

ptpengine:announce_receipt_timeout = 6
PTP announce receipt timeout announced in master state.

ptpengine:announce_receipt_grace_period = 0
PTP announce receipt timeout grace period in slave state: when announce receipt timeout occurs, disqualify current best GM, then wait *n* times announce receipt timeout before resetting. Allows for a seamless GM failover when standby GMs are slow to react. When set to 0, this option is not used.

ptpengine:log_sync_interval = 0
PTP sync message interval in master state. When using unicast negotiation, for slaves this is the minimum interval requested, and for masters this is the only interval granted. (expressed as log 2 i.e. -1=0.5s, 0=1s, 1=2s etc.)

ptpengine:log_sync_interval_max = 5
Maximum Sync message interval requested by slaves when using unicast negotiation, (expressed as log 2 i.e. -1=0.5s, 0=1s, 1=2s etc.)

ptpengine:log_delayreq_override = N
Override the Delay Request interval announced by best master.

ptpengine:log_delayreq_auto = Y
Automatically override the Delay Request interval if the announced value is 127 (0X7F), such as in unicast messages (unless using unicast negotiation).

ptpengine:log_delayreq_interval_initial = 0
Delay request interval used before receiving first delay response (expressed as log 2 i.e. -1=0.5s, 0=1s, 1=2s etc.)

ptpengine:log_delayreq_interval = 0
Minimum delay request interval announced when in master state, in slave state overrides the master interval, required in hybrid mode. When using unicast negotiation, for slaves this is the minimum interval requested, and for masters this is the minimum interval granted. (expressed as log 2 i.e. -1=0.5s, 0=1s, 1=2s etc.)

ptpengine:log_delayreq_interval_max = 5
Maximum Delay Response interval requested by slaves when using unicast negotiation, (expressed as log 2 i.e. -1=0.5s, 0=1s, 1=2s etc.)

ptpengine:log_peer_delayreq_interval = 1
Minimum peer delay request message interval in peer to peer delay mode. When using unicast negotiation, this is the minimum interval requested, and the only interval granted. (expressed as log 2 i.e. -1=0.5s, 0=1s, 1=2s etc.)

ptpengine:log_peer_delayreq_interval_max = 5
Maximum Peer Delay Response interval requested by slaves when using unicast negotiation (expressed as log 2 i.e. -1=0.5s, 0=1s, 1=2s etc.)

ptpengine:foreignrecord_capacity = 5
Foreign master record size (Maximum number of foreign masters).

ptpengine:ptp_allan_variance = 65535
Specify Allan variance announced in master state.

ptpengine:ptp_clock_accuracy = ACC_UNKNOWN
Clock accuracy range announced in master state.
Options: `ACC_25NS ACC_100NS ACC_250NS ACC_1US ACC_2.5US ACC_10US`

ACC_25US ACC_100US ACC_250US ACC_1MS ACC_2.5MS ACC_10MS ACC_25MS ACC_100MS ACC_250MS ACC_1S ACC_10S ACC_10SPLUS ACC_UNKNOWN

ptpengine:utc_offset = 0
Underlying time source UTC offset announced in master state.

ptpengine:utc_offset_valid = N
Underlying time source UTC offset validity announced in master state.

ptpengine:time_traceable = N
Underlying time source time traceability announced in master state.

ptpengine:frequency_traceable = N
Underlying time source frequency traceability announced in master state.

ptpengine:ptp_timescale = PTP
Time scale announced in master state (with ARB, UTC properties are ignored by slaves). When clock class is set to 13 (application specific), this value is ignored and ARB is used.
Options: PTP ARB

ptpengine:ptp_timesource = INTERNAL_OSCILLATOR
Time source announced in master state.
Options: ATOMIC_CLOCK GPS TERRESTRIAL_RADIO PTP NTP HAND_SET OTHER INTERNAL_OSCILLATOR

Note that case matters.

ptpengine:clock_class = 255
Clock class - announced in master state. Always 255 for slave-only. Minimum, maximum and default values are controlled by presets. If set to 13 (application specific time source), announced time scale is always set to ARB. This setting controls the states a PTP port can be in. If below 128, port will only be in MASTER or PASSIVE states (master only). If above 127, port will be in MASTER or SLAVE states.

ptpengine:priority1 = 128
Priority 1 announced in master state, used for Best Master Clock selection.

ptpengine:priority2 = 128
Priority 2 announced in master state, used for Best Master Clock selection.

ptpengine:max_listen = 5
Number of consecutive resets to LISTENING before full network reset

ptpengine:unicast_destinations =
Specify unicast slave addresses for unicast master operation, or unicast master addresses for slave operation. Format is: comma, tab or space-separated IPv4 unicast addresses, one or more. For a slave, when unicast negotiation is used, setting this is mandatory.

ptpengine:unicast_domains =
This is only used by slave-only clocks using unicast destinations to allow for each master to be in a separate domain, such as with Telecom Profile. The number of entries should match the number of unicast destinations, otherwise unconfigured domains or domains set to 0 are set to domain configured in `ptpengine:domain`. The format is a comma, tab, or space-separated list of 8-bit unsigned integers (0 … 255)

ptpengine:unicast_local_preference =
Specify a local preference for each configured unicast destination (`ptpengine:unicast_destinations`). This is only used by slave-only clocks using unicast destinations to allow for each master's BMC selection to be influenced by the slave, such as with Telecom Profile. The number of entries should match the number of unicast destinations, otherwise unconfigured preference is set to 0 (highest). The format is a comma, tab or space-separated list of 8-bit unsigned integers (0 ... 255).

ptpengine:unicast_peer_destination =
Specify peer unicast address for P2P unicast. Mandatory when running unicast mode and P2P delay mode.

ptpengine:management_set_enable = N
Enable handling of PTP management messages.

ptpengine:management_enable = Y
Accept `SET` and `COMMAND` management messages.

ptpengine:igmp_refresh = Y
Send explicit IGMP joins between engine resets and periodically in master state.

ptpengine:master_igmp_refresh_interval = 60
Periodic IGMP join interval (seconds) in master state when running IPv4 multicast: when set below 10 or when `ptpengine:igmp_refresh` is disabled, this setting has no effect.

ptpengine:multicast_ttl = 64
Multicast time to live for multicast PTP packets (ignored and set to 1 for peer to peer messages).

ptpengine:ip_dscp = 0
`DiffServ CodepPoint` for packet prioritization (decimal). When set to zero, this option is not used. Use 46 for Expedited Forwarding (0x2e).

ptpengine:sync_stat_filter_enable = N
Enable statistical filter for Sync messages.

ptpengine:sync_stat_filter_type = min
Type of filter used for Sync message filtering. Options: `none mean min max absmin absmax median`

ptpengine:sync_stat_filter_window = 4
Number of samples used for the Sync statistical filter

ptpengine:sync_stat_filter_window_type = sliding
Sample window type used for Sync message statistical filter. Delay Response outlier filter action. Sliding window is continuous, interval passes every n-th sample only. Options: `sliding interval`

ptpengine:delay_stat_filter_enable = N
Enable statistical filter for Delay messages.

ptpengine:delay_stat_filter_type = min
Type of filter used for Delay message statistical filter. Options: `none mean min max absmin absmax median`

ptpengine:delay_stat_filter_window = 4
Number of samples used for the Delay statistical filter

ptpengine:delay_stat_filter_window_type = sliding
Sample window type used for Delay message statistical filter. Sliding window is continuous, interval passes every n-th sample only. Options: `sliding interval`

ptpengine:delay_outlier_filter_enable = N
Enable outlier filter for the Delay Response component in slave state.

ptpengine:delay_outlier_filter_action = discard
Delay Response outlier filter action. If set to 'filter', outliers are replaced with moving average. Options: `discard filter`

ptpengine:delay_outlier_filter_capacity = 20
Number of samples in the Delay Response outlier filter buffer.

ptpengine:delay_outlier_filter_threshold = 1.000000

Delay Response outlier filter threshold (multiplier for Peirce's maximum standard deviation). When set below 1.0, filter is tighter, when set above 1.0, filter is looser than standard Peirce's test. When autotune is enabled, this is the starting threshold.

ptpengine:delay_outlier_filter_always_filter = N
Always run the Delay Response outlier filter, even if clock is being slewed at maximum rate

ptpengine:delay_outlier_filter_autotune_enable = Y
Enable automatic threshold control for Delay Response outlier filter.

ptpengine:delay_outlier_filter_autotune_minpercent = 20
Delay Response outlier filter autotune low watermark - minimum percentage of discarded samples in the update period before filter is tightened by the autotune step value.

ptpengine:delay_outlier_filter_autotune_maxpercent = 95
Delay Response outlier filter autotune high watermark - maximum percentage of discarded samples in the update period before filter is loosened by the autotune step value.

ptpengine:delay_outlier_autotune_step = 0.100000
The value the Delay Response outlier filter threshold is increased or decreased by when auto-tuning.

ptpengine:delay_outlier_filter_autotune_minthreshold = 0.100000
Minimum Delay Response filter threshold value used when auto-tuning.

ptpengine:delay_outlier_filter_autotune_maxthreshold = 5.000000
Maximum Delay Response filter threshold value used when auto-tuning.

ptpengine:delay_outlier_filter_stepdetect_enable = N
Enable Delay filter step detection (delaySM) to block when certain level exceeded.

ptpengine:delay_outlier_filter_stepdetect_threshold = 1000000
Delay Response step detection threshold. Step detection is performed only when delaySM is below this threshold (nanoseconds).

ptpengine:delay_outlier_filter_stepdetect_level = 500000
Delay Response step level. When step detection enabled and operational, delaySM above this level (nanosecond) is considered a clock step and updates are paused.

ptpengine:delay_outlier_filter_stepdetect_credit = 200
Initial credit (number of samples) the Delay step detection filter can block for. When credit is exhausted, filter stops blocking. Credit is gradually restored.

ptpengine:delay_outlier_filter_stepdetect_credit_increment = 10
Amount of credit for the Delay step detection filter restored every full sample window

ptpengine:delay_outlier_weight = 1.000000
Delay Response outlier weight: if an outlier is detected, determines the amount of its deviation from mean that is used to build the standard deviation statistics and influence further outlier detection. When set to 1.0, the outlier is used as is.

ptpengine:sync_outlier_filter_enable = N
Enable outlier filter for the Sync component in slave state.

ptpengine:sync_outlier_filter_action = discard
Sync outlier filter action. If set to 'filter', outliers are replaced with moving average. Options: `discard filter`

ptpengine:sync_outlier_filter_capacity = 20
Number of samples in the Sync outlier filter buffer.

ptpengine:sync_outlier_filter_threshold = 1.000000

Sync outlier filter threshold: multiplier for the Peirce's maximum standard deviation. When set below 1.0, filter is tighter, when set above 1.0, filter is looser than standard Peirce's test.

ptpengine:sync_outlier_filter_always_filter = N
Always run the Sync outlier filter, even if clock is being slewed at maximum rate

ptpengine:sync_outlier_filter_autotune_enable = Y
Enable automatic threshold control for Sync outlier filter.

ptpengine:sync_outlier_filter_autotune_minpercent = 20
Sync outlier filter autotune low watermark - minimum percentage of discarded samples in the update period before filter is tightened by the autotune step value.

ptpengine:sync_outlier_filter_autotune_maxpercent = 95
Sync outlier filter autotune high watermark - maximum percentage of discarded samples in the update period before filter is loosened by the autotune step value.

ptpengine:sync_outlier_autotune_step = 0.100000
Value the Sync outlier filter threshold is increased or decreased by when auto-tuning.

ptpengine:sync_outlier_filter_autotune_minthreshold = 0.100000
Minimum Sync outlier filter threshold value used when auto-tuning

ptpengine:sync_outlier_filter_autotune_maxthreshold = 5.000000
Maximum Sync outlier filter threshold value used when auto-tuning

ptpengine:sync_outlier_filter_stepdetect_enable = N
Enable Sync filter step detection (delayMS) to block when certain level exceeded.

ptpengine:sync_outlier_filter_stepdetect_threshold = 1000000

Sync step detection threshold. Step detection is performed only when delayMS is below this threshold (nanoseconds)

ptpengine:sync_outlier_filter_stepdetect_level = 500000
Sync step level. When step detection enabled and operational, delayMS above this level (nanosecond) is considered a clock step and updates are paused

ptpengine:sync_outlier_filter_stepdetect_credit = 200
Initial credit (number of samples) the Sync step detection filter can block for. When credit is exhausted, filter stops blocking. Credit is gradually restored

ptpengine:sync_outlier_filter_stepdetect_credit_increment = 10
Amount of credit for the Sync step detection filter restored every full sample window

ptpengine:sync_outlier_weight = 1.000000
Sync outlier weight: if an outlier is detected, this value determines the amount of its deviation from mean that is used to build the standard deviation statistics and influence further outlier detection. When set to 1.0, the outlier is used as is.

ptpengine:calibration_delay = 0
Delay between moving to slave state and enabling clock updates (seconds). This allows one-way delay to stabilize before starting clock updates. Activated when going into slave state and during slave's GM failover.

ptpengine:idle_timeout = 120
PTP idle timeout: if PTPd is in SLAVE state and there have been no clock updates for this amount of time, PTPd releases clock control. Measured in seconds.

ptpengine:panic_mode = N
Enable panic mode: when offset from master is above 1 second, stop updating the clock for a period of time and then step the clock if offset remains above 1 second.

ptpengine:panic_mode_duration = 2
Duration (minutes) of the panic mode period (no clock updates) when offset above 1 second detected.

ptpengine:panic_mode_release_clock = N
When entering panic mode, release clock control while panic mode lasts. If `ntpengine:*` configured, this will fail over to NTP, if not set, PTP will hold clock control during panic mode.

ptpengine:panic_mode_exit_threshold = 0
Do not exit panic mode until offset drops below this value (nanoseconds).

0 = not used.

ptpengine:pid_as_clock_identity = N
Use PTPd's process ID as the middle part of the PTP clock ID - useful for running multiple instances.

ptpengine:ntp_failover = N
Fail over to NTP when PTP time sync not available – requires `ntpengine:enabled`, but does not require the rest of NTP configuration: will warn instead of failing over if cannot control ntpd.

ptpengine:ntp_failover_timeout = 120
NTP failover timeout in seconds: time between PTP slave going into LISTENING state and releasing clock control.

0 = fail over immediately.

ptpengine:prefer_ntp = N
Prefer NTP time synchronization. Only use PTP when NTP not available, could be used when NTP runs with a local GPS receiver or another reference

ptpengine:panic_mode_ntp = N
Same as `ptpengine:panic_mode_release_clock`

ptpengine:timing_acl_permit =
Permit access control list for timing packets. Format is a series of comma, space or tab separated network prefixes: IPv4 addresses or full CIDR notation a.b.c.d/x, where a.b.c.d is the subnet and x is the decimal mask, or a.b.c.d/v.x.y.z where a.b.c.d is the subnet and v.x.y.z is the 4-octet mask. The match is performed on the source IP address of the incoming messages. No addresses are allowed unless an ACL is defined.

ptpengine:timing_acl_deny =
Deny access control list for timing packets. Format is a series of comma, space or tab separated network prefixes: IPv4 addresses or full CIDR notation a.b.c.d/x, where a.b.c.d is the subnet and x is the decimal mask, or a.b.c.d/v.x.y.z where a.b.c.d is the subnet and v.x.y.z is the 4-octet mask. The match is performed on the source IP address of the incoming messages. No addresses are allowed unless an acl is defined.

ptpengine:management_acl_permit =
Permit access control list for management messages. Format is a series of comma, space or tab separated network prefixes: IPv4 addresses or full CIDR notation a.b.c.d/x, where a.b.c.d is the subnet and x is the decimal mask, or a.b.c.d/v.x.y.z where a.b.c.d is the subnet and v.x.y.z is the 4-octet mask. The match is performed on the source IP address of the incoming messages. No addresses are allowed unless an acl is defined.

ptpengine:management_acl_deny =
Deny access control list for management messages. Format is a series of comma, space or tab separated network prefixes: IPv4 addresses or full CIDR notation a.b.c.d/x, where a.b.c.d is the subnet and x is the decimal mask, or a.b.c.d/v.x.y.z where a.b.c.d is the subnet and v.x.y.z is the 4-octet mask. The match is performed on the source IP address of the incoming messages. No addresses are allowed unless an acl is defined.

ptpengine:timing_acl_order = permit-deny
Order in which permit and deny access lists are evaluated for timing packets. The IP address that the packet came from is checked for a match in the permit list and deny list. When the order is permit-deny (default) the address is only permitted if it is in the permit list and is not in the deny list. For deny-permit the address is allowed if it is not in the deny list or if it is in the permit list, otherwise it is rejected.

Options: `permit-deny deny-permit`

ptpengine:management_acl_order = permit-deny
Order in which permit and deny access lists are evaluated for management messages. The IP address that the packet came from is checked for a match in the permit list and deny list. When the order is `permit-deny` (default) the address is only permitted if it is in the permit list and is not in the deny list. For deny-permit the address is allowed if it is not in the deny list or if it is in the permit list, otherwise it is rejected.

Options: `permit-deny deny-permit`

ptpengine:sync_sequence_checking = N
When enabled, Sync messages will only be accepted if sequence ID is increasing. This is limited to 50 dropped messages.

ptpengine:clock_update_timeout = 0
If set to non-zero, timeout in seconds, after which the slave resets if no clock updates made.

clock:no_adjust = N
Do not adjust the clock.

clock:no_reset = N
Do not step the clock - only slew.

clock:step_startup_force = N
Force clock step on first sync after startup regardless of offset and `clock:no_reset`

clock:step_startup = N
Step clock on startup if offset >= 1 second, ignoring panic mode and `clock:no_reset`

clock:drift_handling = preserve
Observed drift handling method between servo restarts:

`reset`: set to zero (not recommended)

`preserve`: use kernel value,

`file`: load/save to drift file on startup/shutdown, use kernel value in between. To specify drift file, use the `clock:drift_file` setting.

Options: `reset preserve file`

clock:drift_file = TCPWARE:ptpd2_kernelclock.drift
Specify drift file.

clock:leap_second_pause_period = 5
Time (seconds) before and after midnight that clock updates should be suspended for during a leap second event. The total duration of the pause is twice the configured duration.

clock:leap_second_notice_period = 43200
Time (seconds) before midnight that PTPd starts announcing the leap second if it's running as master

clock:leap_seconds_file =
Specify leap second file location - up to date version can be downloaded from:

http://www.ietf.org/timezones/data/leap-seconds.list

clock:leap_second_handling = accept
Behavior during a leap second event:

`accept`: inform the OS kernel of the event

`ignore`: do nothing - ends up with a 1-second offset which is then slewed

`step`: similar to ignore, but steps the clock immediately after the leap second event

`smear`: do not inform kernel, gradually introduce the leap second before the event by modifying clock offset (see `clock:leap_second_smear_period`)

Options: `accept ignore step smear`

clock:leap_second_smear_period = 86400
Time period (Seconds) over which the leap second is introduced before the event.

Example: when set to 86400 (24 hours), an extra 11.5 microseconds is added every second

clock:max_offset_ppm = 500
Maximum absolute frequency shift which can be applied to the clock servo when slewing the clock. Expressed in parts per million (1 ppm = shift of 1 ms per second). Values above 512 will use the tick duration correction to allow even faster slewing. Default maximum is 512 without using tick.

servo:dt_method = constant
How servo update interval (delta t) is calculated:

`none`: servo not corrected for update interval (dt always 1),

`constant`: constant value (target servo update rate - sync interval for PTP,

`measured`: servo measures how often it's updated and uses this interval.

Options: `none constant measured`

servo:delayfilter_stiffness = 6
One-way delay filter stiffness.

servo:kp = 0.100000
Clock servo PI controller proportional component gain (kP).

servo:ki = 0.001000
Clock servo PI controller integral component gain (kI).

servo:dt_max = 5.000000
Maximum servo update interval (delta t) when using measured servo update interval (`servo:dt_method = measured`), specified as sync interval multiplier.

servo:stability_detection = N

Enable clock synchronization servo stability detection (based on standard deviation of the observed drift value) - drift will be saved to drift file / cached when considered stable, also clock stability status will be logged.

servo:stability_threshold = 10.000000
Specify the observed drift standard deviation threshold in parts per billion (ppb) - if standard deviation is within the threshold, servo is considered stable.

servo:stability_period = 1
Specify for how many statistics update intervals the observed drift standard deviation has to stay within threshold to be considered stable.

servo:stability_timeout = 10
Specify after how many minutes without stabilization servo is considered unstable. Assists with logging servo stability information and allows to preserve observed drift if servo cannot stabilize.

servo:max_delay = 0
Do accept master to slave delay (delayMS - from Sync message) or slave to master delay (delaySM - from Delay messages) if greater than this value (nanoseconds). 0 = not used.

servo:max_delay_max_rejected = 0
Maximum number of consecutive delay measurements exceeding maxDelay threshold, before slave is reset.

servo:max_delay_stable_only = N
If `servo:max_delay` is set, perform the check only if clock servo has stabilized.

servo:max_offset = 0
Do not reset the clock if offset from master is greater than this value (nanoseconds). 0 = not used.

global:lock_file =TCPWARE:PTPD2.LOCK

Lock file location.

global:auto_lockfile = N
Use mode specific and interface specific lock file (overrides `global:lock_file`).

global:lock_directory = TCPWARE:
Lock file directory: used with automatic mode-specific lock files, also used when no lock file is specified. When lock file is specified, it's expected to be an absolute path.

global:ignore_lock = N
Skip lock file checking and locking.

global:quality_file =
File used to record data about sync packets. Enables recording when set.

global:quality_file_max_size = 0
Maximum sync packet record file size (in kB) - file will be truncated if size exceeds the limit. 0 - no limit.

global:quality_file_max_files = 0
Enable log rotation of the sync packet record file up to n files.

0 - do not rotate.

global:quality_file_truncate = N
Truncate the sync packet record file every time it is (re) opened (on startup or restart).

global:status_file = TCPWARE:ptpd2.status
File used to log ptpd2 status information.

global:log_status = N

Enable / disable writing status information to file.

global:status_update_interval = 1
Status file update interval in seconds.

global:log_level = LOG_ALL
Specify log level (only messages at this priority or higher will be logged). The minimal level is `LOG_ERR`.

Options: `LOG_ERR LOG_WARNING LOG_NOTICE LOG_INFO LOG_ALL`

global:statistics_file =
Specify statistics log file path. Setting this enables logging of statistics but can be overridden with `global:log_statistics`.

global:statistics_log_interval = 0
Log timing statistics every *n* seconds for Sync and Delay messages

(0 - log all).

global:statistics_file_max_size = 0
Maximum statistics log file size (in kB) - log file will be truncated if size exceeds the limit. 0 - no limit.

global:statistics_file_max_files = 0
Enable log rotation of the statistics file up to *n* files.

0 - do not rotate.

global:statistics_file_truncate = N
Truncate the statistics file every time it is (re) opened (startup and restart).

global:dump_packets = N

Dump the contents of every PTP packet

`global:log_statistics = N`
Log timing statistics for every PTP packet received

`global:statistics_timestamp_format = datetime`
Timestamp format used when logging timing statistics

(when global:log_statistics is enabled):

`datetime` - formatted date and time: YYYY-MM-DD hh:mm:ss.uuuuuu

`unix` - Unix timestamp with nanoseconds: s.ns

`both` - Formatted date and time, followed by UNIX timestamp (adds one extra field to the log)

Options: `datetime unix both`

`global:statistics_update_interval = 30`
Clock synchronization statistics update interval in seconds.

`global:periodic_updates = N`
Log a status update every time statistics are updated (`global:statistics_update_interval`). The updates are logged even when ptpd is configured without statistics support.

`global:timingdomain_election_delay = 15`
Delay (seconds) before releasing a time service (NTP or PTP) and electing a new one to control a clock. 0 = elect immediately

`global:enable_snmp=N/Y`
Use the AgentX protocol to connect to the SNMP agent on the system to provide information about the server.

`ntpengine:enabled = N`

Enable NTPd integration.

ntpengine:control_enabled = N
Enable control over local NTPd daemon.

ntpengine:check_interval = 15
NTP control check interval in seconds.

ntpengine:key_id = 0
NTP key number - must be configured as a trusted control key in `ntp.conf`, and be non-zero for the `ntpengine:control_enabled` setting to take effect.

ntpengine:key =
NTP key (plain text, max. 20 characters) - must match the key configured in ntpd's keys file, and must be non-zero for the `ntpengine:control_enabled` setting to take effect.

TCPware NetControl PTP Commands

The following commands are available in TCPware's `NETCONTROL` utility to manage the PTP server.

DEBUG

Set debugging level

```
$ mult netc ptp debug 0
Connected to NETCONTROL server on "LOCALHOST"
bigboote.example.com Network Control V6.1 at Fri 16-Aug-2019 1:36PM-EDT
Debug level now set to 0
```

HELP

List of PTP control commands

```
$ mult netc ptp help
Connected to NETCONTROL server on "LOCALHOST"
bigboote.example.com Network Control V6.1 at Fri 16-Aug-2019 1:41PM-EDT
debug - set debugging level
help - this help information
noop - no operation
ptp-control-version - version of netcontrol control
reload - restart PTP
restart - restart PTP
show - show operating information about PTP
shutdown - shutdown PTP
start - start PTP
version - version of PTP
```

RELOAD

Restart/reload PTP

```
$ mult netc ptp reload
Connected to NETCONTROL server on "LOCALHOST"
bigboote.example.com Network Control V6.1 at Fri 16-Aug-2019 1:37PM-EDT
PTP server restarting
PTP server restart requested
```

SHOW
Show operating information about PTP

```
$ mult netc ptp show
Connected to NETCONTROL server on "LOCALHOST"
bigboote.example.com Network Control V6.1 at Fri 16-Aug-2019 1:40PM-EDT
Offset From Master 0 0
Mean Path Delay 0 0
observed parent clock phase change rate 0
Grand Master Identity aa 0 4 ff fe 0 ae 8 (sys1.example.com)
Grand Master clock quality 248 254 65535
Steps Removed 0
Clock Class 248
Clock Accuracy 254
Offset Scaled Log Variance 0
Domain Number 0
End of Show PTP
```

SHUTDOWN
Shutdown PTP

```
$ mult netc ptp shutdown
Connected to NETCONTROL server on "LOCALHOST"
sys1.example.com Network Control V6.1 at Fri 16-Aug-2019 1:40PM-EDT
Starting shutdown of PTP server
PTP server shutdown
```

START
Start PTP

```
$ mult netc ptp shutdown
Connected to NETCONTROL server on "LOCALHOST"
bigboote.example.com Network Control V6.1 at Fri 16-Aug-2019 1:40PM-EDT
Starting shutdown of PTP server
PTP server shutdown
```

VERSION
Version of TCPware PTP and PTP

```
$ mult netc ptp version
Connected to NETCONTROL server on "LOCALHOST"
bigboote.example.com Network Control V6.1 at Fri 16-Aug-2019 1:36PM-EDT
PTP for TCPware V1.0
PTP server version = 2.0(1)
```

Files

TCPWARE:PTPD2.CONF
System specific configuration

TCPWARE:ETHER.
Optional file containing a list of ethernet mac addresses and host names for displaying. One line per host name in the format

```
XX:XX:XX:XX:XX:XX name_to_be_displayed
```

TCPWARE:LEAP-SECONDS.LIST
List of dates in which a leap second has been added to adjust from Coordinated Universal Time (UTC) to International Atomic Time (TAI).

Appendix A. NFS-to-OpenVMS Filename Mapping

This appendix is a supplement to the NFS client and NFS server chapters in this guide. Mapping is done for the NFSv2 and NFSv3 clients when presenting the device as an OpenVMS ODS-2 device. When the NFSv3 client can present the mounted device as an OpenVMS ODS-5 device there is no filename mapping done other than to observe the process's parameter for case sensitivity.

The following filename mapping rules are necessary because:

- The NFSv2 client must map (translate) special characters in NFS server filenames that are not valid in OpenVMS filenames.
- The server must map special characters in filenames users create on the NFS system client host that are not valid in OpenVMS filenames.

OpenVMS ODS-2disk filenames can be 39 characters long (as can file extensions) and include only the following characters: `0` through `9`, `A` through `Z`, `a` through `z`, dollar sign (`$`), hyphen (`-`), and underscore (_).

OpenVMS files also include version numbers (after the semicolon) that cannot exceed the value 32767. The server preserves these version numbers, and hard links the highest numbered version to an unversioned filename (see the *Note* at the end of this section).

NFS filenames can have any of the ASCII characters except the null character (octal 000) and the slash (`/`, or octal `057`), which delimits directory levels. UNIX filenames can have up to 255 characters.

The client and the server use the same filename mapping schemes. Four types of mapping schemes are available:

- SRI International mapping, the default scheme between UNIX and OpenVMS systems
- ODS5 mapping, for OpenVMS systems exporting an ODS5 disk. There is almost no mapping needed as most of the characters are allowed. If the system or disk does not support ODS5, then this defaults to SRI mapping.
- PATHWORKS non-case-sensitive mapping (NFS server only)
- PATHWORKS case-sensitive mapping (NFS server only)

The below table shows the default SRI International mapping.

ASCII character...	Is mapped in OpenVMS to...	With octal value...
Ctrl/A (soh)	$4A	001
Ctrl/B (stx)	$4B	002
Ctrl/C (etx)	$4C	003
Ctrl/D (eot)	$4D	004
Ctrl/E (enq)	$4E	005
Ctrl/F (ack)	$4F	006
Ctrl/G (bel)	$4G	007
Ctrl/H (bs)	$4H	010
Ctrl/I (ht)	$4I	011
Ctrl/J (nl)	$4J	012
Ctrl/K (vt)	$4K	013
Ctrl/L (np)	$4L	014
Ctrl/M (cr)	$4M	015
Ctrl/N (so)	$4N	016
Ctrl/O (si)	$4O	017
Ctrl/P (dle)	$4P	020
Ctrl/Q (dc1)	$4Q	021
Ctrl/R (dc2)	$4R	022

Ctrl/S (dc3)	$4S	023
Ctrl/T (dc4)	$4T	024
Ctrl/U (nak)	$4U	025
Ctrl/V (syn)	$4V	026
Ctrl/W (etb)	$4W	027
Ctrl/X (can)	$4X	030
Ctrl/Y (em)	$4Y	031
Ctrl/Z (sub)	$4Z	032
Ctrl/[(esc)	$6B	033
Ctrl/(fs)	$6C	034
Ctrl/] (gs)	$6D	035
Ctrl/^ (rs)	$6E	036
Ctrl/_ (us)	$6F	037
SPACE (sp)	$7A	040
!	$5A	041
"	$5B	042
#	$5C	043
$	$$ (See Rule 9 in next table)	044
%	$5E	045

&	$5F	046
'	$5G	047
(	$5H	050
)	$5I	051
*	$5J	052
+	$5K	053
,	$5L	054
-	same	055
.	. or $5N (See next table)	056
/	not mapped (directory delimiter)	057
0 to 9	same	060 to 071
:	$5Z	072
;	$7B	073
<	$7C	074
=	$7D	075
>	$7E	076
?	$7F	077
@	$8A	100
A to Z	same	101 to 132

[	$8B	133
	$8C	134
]	$8D	135
^	$8E	136
_	same	137
`	$9A	140
a to z	same	141 to 172
{	$9B	173
|	$9C	174
}	$9D	175
~	$9E	176
DEL	$9F	177
octal 200 to ¿	$200 to $277	200 to 277 (Multinational)
À to octal 377	$300 to $377	300 to 377 (Multinational)

The NFS-to-OpenVMS filename translation rules below are based on the character mapping scheme in the above table. The OpenVMS-to-NFS mapping rules are the converse of these rules.

Rule	What happens to filenames from NFS to OpenVMS...
1	Lowercase characters become uppercase (unless Rule 2 applies; see also Rule 3): `foobar.txt` becomes `FOOBAR.TXT;1`

2	Initial uppercase characters or a sequence of case-shifted characters get a `$` prefix: `CaseShiftedFile` becomes `$C$ASE$S$HIFTED$F$ILE.;1`
3	An unversioned file gets a version number preceded by a semicolon: `foobar.txt` becomes `FOOBAR.TXT;1`
4	If a filename does not include a file extension dot (`.`), it gets one before the version number semicolon: `foobar` becomes `FOOBAR.;1`
5	After being translated, if a filename (or its extension after the dot) has more than 39 characters or the version number (after the semicolon) is greater than 32767, the file will not show up in an OpenVMS listing.
6	The first dot in a filename is preserved, unless the result fails the 39-character extension limit test in Rule 5 (if so, the dot becomes `$5N`). Each successive dot becomes `$5N`, unless the filename exceeds the limits in Rule 5. `more.file.txt` becomes `MORE.FILE$5NTXT;1`
7	If the filename is a directory name, each dot in it becomes `$5N` and the filename gets the `.DIR` extension: `dot.directory.list` becomes `DOT$5NDIRECTORY$5NLIST.DIR;1`
8	Invalid OpenVMS characters become the escape character sequences in the second column of the above mapping table (`$` followed by a digit and a letter): `special#character&file` becomes `SPECIAL$5CCHARACTER$5FFILE.;1` (`#` becomes `$5C` and `&` becomes `$5F`)

9	Any existing `$` becomes `$$` (plus any `$` added due to Rule 2 or 8 above): `dollar$Sign$5cfile` becomes `DOLLAR$$$S$IGN$$5CFILE.;1`

Note: Many UNIX applications use only unversioned filenames and simply overwrite existing files. Normally, if you create a file with a new version number in OpenVMS, the client and server include the version number when displayed on the NFS server. Also, if `foobar;9` is the highest-numbered version in NFS, the client creates an unversioned `foobar` file and hard-links to it. You can limit this versioning by specifying the `/NOVERSION` qualifier with the client `NFSMOUNT` command; this limits the number of versions created to one, unless an accompanying Attributes Data File (ADF) specifies otherwise.

Keep in mind that the highest numbered version is not necessarily the most recent one, such as when you create a file explicitly with a lower version number than an existing one. For example, the client and server normally do not append a version number at the end of a foobar;1 file and just keep it as foobar, unless you explicitly specify foobar;1 with an already existing higher version of the file.

Appendix B. TCPware Logicals

The following table lists the TCPware logicals in alphabetical order:

FTP_STARTUP
Defines FTP_STARTUP to point to the FTP_STARTUP.COM file.

```
$ DEFINE /SYSTEM /EXECUTIVE FTP_STARTUP SYS$MANAGER:FTP_STARTUP.COM
```

Client users can override this startup file by creating their own. Including the command DEFINE /PROCESS FTP_STARTUP in a user's LOGIN.COM file overrides any DEFINE /SYSTEM /EXEC command in the SYS$MANAGER:SYSTARTUP_V5.COM file.

NETCU_STARTUP
Defines NETCU_STARTUP to point to the NETCUSTART.COM file.

For example, you can include the following in your LOGIN.COM file:

```
ASSIGN SYS$LOGIN:NETCUSTART.COM NETCU_STARTUP
```

When you start NETCU, NETCU_STARTUP points to the specified file (SYS$LOGIN:NETCUSTART.COM for example) and processes all the commands.

Note: The system ignores all commands following an EXIT or QUIT command in the file. NETCU ignores any "commented-out" command lines in files (such as SERVICES.COM) that are used as input to NETCU. The commented-out line in the file should begin with the !, the #, or the ; character. NETCU does not execute the command line until you remove the character.

SSH_DIR
Points to the directory where SSH's master server log file is kept. Normally, this is TCPWARE_COMMON:[TCPWARE].

SSH2_DIR
Points to the directory where the SSH master server log file is kept. Normally, this is TCPWARE_COMMON:[TCPWARE_SSH2].

SSH_EXE
Points to the directory where SSH executables are kept. Normally, this is TCPWARE_COMMON:[TCPWARE]. It is defined through @TCPWARE:CNFNET SSH. The configuration procedure should write these to the common configuration file and check the values at start up and delete them at shutdown.

SSH_LOG
Points to the directory where the log files are kept. Normally, this is TCPWARE_COMMON:[TCPWARE.LOG]. It is defined through @TCPWARE:CNFNET SSH. The configuration procedure writes these to the common configuration file and check the values at start up and delete them at shutdown.

SSH_MAX_SESSIONS
This is set to the maximum number of concurrent SSH sessions allowed to the server system. If SSH_MAX_SESSIONS is not defined, the default is 9999. Setting SSH_MAX_SESSIONS to zero (0) will cause an error. The value must be between 1 and 9999. It is defined through @TCPWARE:CNFNET SSH. The configuration procedure should write these to the common configuration file and check the values at start up and delete them at shutdown.

TCPWARE_SSH_SFTP_SERVER_DEBUG
Enables debugging messages for the SFTP-SERVER2 image that provides service to SCP2 commands that use the SFTP protocol. When this is defined, the file SFTP-SERVER.LOG is created in the user's login directory. These files are not purged. Larger values yield more debugging information.

TCPWARE_SSH_SCP_SERVER_DEBUG
Enables debugging messages for the SCP-SERVER1 image that provides service to SCP2 commands that use the RCP over SSH2 protocol. When this is defined, the file SCP-SERVER.LOG is created in the user's login directory. These files are not purged. Larger values yield more debugging information.

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.

TCPWARE_DOMAINLIST
Allows you to set up to six domains in a search list, as well as the minimum number of dots to recognize in a host name to make it fully qualified. The client reads this information from this logical through CNFNET.

TCPWARE_DOMAINNAME
Specifies the internet addresses of up to three name servers the client can query. The client reads this information from this logical through CNFNET.

TCPWARE_FTP_220_REPLY
Defines a message displayed when a user connects to the server and can log in. This message replaces the default message.

For example, you can define the welcome text equivalence string as follows:

```
$ DEFINE /SYSTEM /EXECUTIVE TCPWARE_FTP_220_REPLY -
_$ "**AUTHORIZED USE ONLY **",-
_$ "bart.example.com (192.168.34.56)", -
_$ "TCPware FTPD V6.1 (c) 2021 Process Software"
```

Alternately, you can include the last three equivalence strings in an FTP_WELCOME.TXT file and define the logical as follows:

```
$ DEFINE /SYSTEM /EXECUTIVE TCPWARE_FTP_220_REPLY -
_$ "@SYS$MANAGER:FTP_WELCOME.TXT"
```

In either case, when a user connects to a host, the message appears as follows:

```
220-** AUTHORIZED USE ONLY **
220-bart.example.com (192.168.34.56)
```

```
220 TCPware FTPD V6.1 (c) 2021 Process Software
_Username []:
```

TCPWARE_FTP_221_REPLY
Defines a message to appear when a user ends the FTP session. If not defined, TCPware uses the default message. You can define a text string or file.

```
$ DEFINE /SYSTEM /EXECUTIVE TCPWARE_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
```

TCPWARE_FTP_230_REPLY
Defines a message to appear when a user successfully logs in. If not defined, TCPware uses the default message. You can define a text string or file. For example:

```
$ DEFINE /SYSTEM /EXECUTIVE TCPWARE_FTP_230_REPLY "Login successful"
```

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

```
230 Login successful
```

TCPWARE_FTP_421_REPLY
Defines a message sent when a user connects 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. You can define a text string or file.

```
$ DEFINE /SYSTEM /EXECUTIVE TCPWARE_FTP_421_REPLY-
_$ "System maintenance in progress until 17:30"
```

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

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

TCPWARE_FTP_421_REPLY has precedence over TCPWARE_FTP_220_REPLY.

TCPWARE_FTP_ACCESS
TCPWARE_FTP_*username*_ACCESS
These SYSTEM logical names are used to specify the types of access that the user of the FTP server is not allowed to perform. TCPWARE_FTP_ACCESS controls all users that do not have TCPWARE_FTP_*username*_ACCESS defined. The values are:

- D - Delete
- L - List (Directory)
- R - Read
- S - Spawn
- W - Write

```
$ DEFINE/SYSTEM/EXECUTIVE TCPWARE_FTP_ANONYMOUS_ACCESS WDS
```

will prevent the user ANONYMOUS from storing files on the system, deleting files that are present on the system or using the site specific spawn command.

TCPWARE_FTP_ADD_CC_ON_FIXED_RECORD_FILES
When the logical TCPWARE_FTP_ADD_CC_ON_FIXED_RECORD_FILES is defined to TRUE and a file is transferred as TYPE IMAGE with QUOTE SITE RMS BLOCK OFF in effect, the FTP server will separate the records of a fixed length record file with the linefeed character. This is useful for avoiding the explicit conversion necessary when transferring the file to a non-VMS system with an FTP client that is not able to do record mode transfers.

TCPWARE_FTP_ALL_VERSIONS
Requests the NLST and LIST commands to display all versions of the specified files. If TCPWARE_FTP_ALL_VERSIONS is defined, TCPWARE_FTP_STRIP_VERSION has no effect.

TCPWARE_FTP_ALL_VERSIONS is ignored if the FTP server is in UNIX emulation mode.

TCPWARE_FTP_ALLOWCAPTIVE
By default, the FTP server does not allow file transfers for CAPTIVE accounts. Defining this logical allows CAPTIVE accounts to use all FTP commands except SITE SPAWN.

```
$ DEFINE /SYSTEM /EXECUTIVE TCPWARE_FTP_ALLOWCAPTIVE " "
```

You must modify the CAPTIVE account procedure to allow the FTP server to start the data transfer process. The procedure can check if the logical TT is equal to TCPWARE:FTPSERVER_DTP.COM and exit out of the login procedure:

```
$! Check if this is the TCPware FTP data transfer process:
$ IF F$LOGICAL("TT") .EQS. "TCPWARE:FTPSERVER_DTP.COM" THEN EXIT
$! Refuse other network connections (such as DECnet):
$ IF F$MODE() .EQS. "NETWORK" THEN LOGOUT
$! (or allow by using "...THEN EXIT" above)
$! Remainder of CAPTIVE procedure follows:
$....
```

TCPWARE_FTP_ANONYMOUS_230_REPLY
Defines a message to appear when an ANONYMOUS user successfully logs in. If not defined, TCPware uses the default message. You can define a text string or file.

```
$ DEFINE/SYSTEM/EXECUTIVE TCPWARE_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
```

TCPWARE_FTP_ANONYMOUS_RIGHTS

Defines write, rename, and delete access rights for the ANONYMOUS FTP user in addition to read access.

```
$ DEFINE/SYS/EXEC/NOLOG TCPWARE_FTP_ANONYMOUS_RIGHTS -
_$ "WRITE,RENAME,DELETE"
```

TCPWARE_FTP_ANONYMOUS_ROOT
Defines access restrictions for users logged in as ANONYMOUS. For example, you can set access restrictions for users logged in as ANONYMOUS to allow access to just the ANONYMOUS$USER directory and its subdirectories:

```
$ DEFINE /SYSTEM/EXECUTIVE TCPWARE_FTP_ANONYMOUS_ROOT -
_$ ANONYMOUS$USER:
```

If not set, the FTP server defaults to the setting in TCPWARE_FTP_ROOT if it exists.

TCPWARE_FTP_DISALLOW_UNIX_STYLE
Controls whether UNIX style filename parsing is done. If not defined, it defaults to TRUE (UNIX-style life specifications are not allowed). Defining to FALSE allows file specifications with the "/" character in them to be treated as UNIX file specification.

```
$ DEFINE /SYSTEM/NOLOG/EXECUTIVE TCPWARE_FTP_DISALLOW_UNIX_STYLE ?
```

TCPWARE_FTP_DONT_REPORT_FILESIZE
If this logical is defined, the reporting of the estimate of the number of bytes to be transferred in the 150 response line is suppressed. Some FTP clients expect this number to be exact. The FTP server is unable to determine an exact count without processing the entire file, so an estimate of the number of bytes used to store the file is returned. The inaccuracy comes from the differences in the way OpenVMS records and line breaks are handled. The ? in the logical represents where defined values go.

```
$ DEFINE/SYSTEM/EXEC TCPWARE_FTP_DONT_REPORT_FILESIZE ?
```

TCPWARE_FTP_EXTENSION_QUANTITY

Defines the default allocation/extension quantity for new files and appends. The `?` in the logical represents where defined values go. Defined values must be numeric.

```
$ DEFINE /SYSTEM/NOLOG/EXECUTIVE TCPWARE_FTP_EXTENSION_QUANTITY ?
```

TCPWARE_FTP_GETHOST_MAX_TIME

When a new connection arrives at the FTP server it attempts to resolve the name of the host that originated the connection. If this process takes a long time, it can stall all other connections, both active and new. To adjust how long the FTP server is allowed to take to look up the host name, set the logical `TCPWARE_FTP_GETHOST_MAX_TIME` to the VMS delta time that can elapse before it gives up. The default value 10 seconds (`0 0:0:10`).

TCPWARE_FTP_IDLE_TIMEOUT

Changes the timeout for FTP connection attempts to something other than the default of 10 minutes. The FTP server checks the timeout when you enter and complete a command. You can set this logical any time, and it effectively changes the idle timeout for open, non-idling connections as well as for any future ones. Make sure to use delta time for the time syntax.

```
$ DEFINE /SYSTEM/EXECUTIVE TCPWARE_FTP_IDLE_TIMEOUT "0 00:20:00"
```

This example changes the idle timeout to 20 minutes. The default is 10 minutes if no time is specified. Setting the value to 0 disables idle timeout.

TCPWARE_FTP_IGNORE_UNIX_DASH_OPTIONS

By default, the FTP server ignores Unix-style dash options on `LIST` and `NLST` when in Unix mode (for example, the "`-l`" in "`ls -l`"). Define this to be `FALSE` to tell the FTP server to pay attention to Unix-style dash options.

```
$ DEFINE /SYSTEM/EXEC TCPWARE_FTP_IGNORE_UNIX_DASH_OPTIONS FALSE
```

TCPWARE_FTP_KEEP_DIR_EXT

Sometimes the FTP server strips the `.DIR` extension from the file name of a directory when the `NLST` function is requested. The FTP server looks for `TCPWARE_FTPD_KEEP_DIR_EXT` and, if defined, does not remove the `.DIR` extension.

```
$ DEFINE /SYSTEM/EXECUTIVE TCPWARE_FTPD_KEEP_DIR_EXT TRUE
```

To return to the default behavior, remove this logical.

TCPWARE_FTP_MAXIMUM_CONNECTION_WAIT
A VMS delta time for how long the FTP client (or programming library) should wait for the 220 response after connecting to the FTP server.

TCPWARE_FTP_NOKEEPALIVES
When `TCPWARE_FTP_NOKEEPALIVES` is defined, the FTP server will not send keepalives on the control channel. The `KEEPALIVE` command allows the FTP client program to toggle, regardless of whether it desires keepalives to be sent on the control channel. The `SET [NO]KEEPALIVE` command allows the FTP client to explicitly set whether it desires keepalives on the control channel

TCPWARE_FTP_LOGFILE
Defines a specific name of a log file. Use this if you suspect break-ins to the FTP server.

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

This logical must be defined before TCPware FTP is started (or FTP must be restarted after defining it for it to take effect).

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.

Specifies the name of the file to which ALL commands and responses to ANONYMOUS FTP services are logged. If `TCPWARE_FTP_LOG_ALL_USERS` is also defined, then commands and responses for all users are logged.

TCPWARE_FTP_LOG_ALL_USERS

This logical causes all commands and responses to be logged to the file defined by TCPWARE_FTP_LOGFILE. The default (when this logical is not defined) is to just log the commands and responses for anonymous users.

```
$ DEFINE TCPWARE_FTP_LOG_ALL_USERS
```

TCPWARE_FTP_MAX_SERVERS
Allows the maximum number of servers to be set. The default is 10000.

```
$ DEFINE /SYSTEM /EXECUTIVE TCPWARE_FTP_MAX_SERVERS "1500"
```

TCPWARE_FTP_MAXREC
The FTP client and the FTP server check the record size of an ASCII transfer and disallow more than 8192 byte records. Define this logical to override the default of 8192. The definition of this logical is commented out but defined in the FTP_CONTROL.COM file as follows:

```
$ !DEFINE /SYSTEM /NOLOG /EXECUTIVE TCPWARE_FTP_MAXREC 8192
```

TCPWARE_FTP_MESSAGE_FILE
Defines the message file the FTP user sees when connecting to the server or moving between directories. The definition of this logical is commented out but defined in the FTP_CONTROL.COM file as follows:

```
$ !DEFINE /SYSTEM /NOLOG /EXECUTIVE TCPWARE_FTP_MESSAGE_FILE ".MESSAGE"
```

TCPWARE_FTP_ONLY_BREAK_ON_CRLF
If this logical is set and an ASCII file is transferred, a new line is created in the file upon receipt of a carriage return/line feed sequence.

If this logical is not set and an ASCII file is transferred, a new line is created upon receipt of either a carriage return/line feed sequence or a line feed.

TCPWARE_FTP_PASSWORD_WARNING_MESSAGE

The logical TCPWARE_FTP_PASSWORD_WARNING_MESSAGE defines the message that the FTP server displays when the user's password is going to expire within the warning time. If the amount of time before the password expires is to be displayed, use a %s in the logical.

```
$ DEFINE/SYSTEM/EXEC TCPWARE_FTP_PASSWORD_WARNING_MESSAGE "%s"
$ DEFINE/SYSTEM/EXEC TCPWARE_FTP_PASSWORD_WARNING_MESSAGE "message text string"
```

TCPWARE_FTP_PASSWORD_WARNING_TIME

The logical TCPWARE_FTP_PASSWORD_WARNING_TIME uses the VMS delta time to specify the minimum remaining lifetime for the user's password. If the remaining lifetime is greater than the VMS delta time then no message is displayed. It is necessary to define this value to enable checking for the remaining lifetime of a password.

```
$ DEFINE/SYSTEM/EXEC @TCPWARE_FTP_PASSWORD_WARNING_TIME "dddd hh:mm:ss.hh"
```

TCPWARE_FTP_RECEIVE_THRESHOLD

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. Defined values can be either alpha or numeric.

```
$ DEFINE TCPWARE_FTP_RECEIVE_THRESHOLD ?
```

TCPWARE_FTP_RECODE_NONVMS_FILE_NAMES

If this logical is defined, and the FTP server is not operating in UNIX mode, it recodes filenames that are not legal OpenVMS file names in the same manner that it would normally recode filenames when operating in UNIX mode. This is useful for handling filenames with multiple dots (.), spaces, and other characters that VMS does not allow in filenames while retaining the OpenVMS directory syntax.

```
$ DEFINE TCPWARE_FTP_RECODE_NONVMS_FILE_NAMES filename
```

TCPWARE_FTP_ROOT

Defines the system-wide default directory access restrictions for client users. The logical may be defined as a single directory or a search list of directories.

For example, you can restrict all users logged in via FTP to the COMMON$USER directory and its subdirectories:

```
$ DEFINE /SYSTEM /EXECUTIVE TCPWARE_FTP_ROOT COMMON$USER:
```

The default directory is not set to the value of this logical or to the value of TCPWARE_FTP_*username*_ROOT.

TCPWARE_FTP_*username*_ROOT

The TCPWARE_FTP_*username*_ROOT (system level, executive mode) logical defines access restrictions for an FTP client logging in as *username*. The logical may be defined as a single directory or a search list of directories.

For example, you can restrict user CLARK to the COMMON$USER:[CLARK] directory and its subdirectories, as follows:

```
$ DEFINE/SYSTEM/EXEC TCPWARE_FTP_CLARK_ROOT COMMON$USER:[CLARK]
```

Because the FTP server restricts access by default to the directory setting in the TCPWARE_FTP_ROOT logical (described earlier), if it exists, you may want to use the special wildcard (*) setting with the TCPWARE_FTP_*username*_ROOT logical to bypass the default for *username*. For example, to restrict the bulk of users to DISK$SYS_LOGIN, restrict users KATE and PAUL to ENG$DISK, but allow SYSTEM full access to locations covered by its account, define the following logicals:

```
$ DEFINE/SYSTEM/EXEC TCPWARE_FTP_ROOT DISK$SYS_LOGIN ! default
$ DEFINE/SYSTEM/EXEC TCPWARE_FTP_KATE_ROOT ENG$DISK  ! limits KATE
```

```
$ DEFINE/SYSTEM/EXEC TCPWARE_FTP_PAUL_ROOT ENG$DISK  ! limits PAUL
$ DEFINE/SYSTEM/EXEC TCPWARE_FTP_SYSTEM_ROOT *       ! full SYSTEM
```

ANONYMOUS user access restrictions are described under TCPWARE_FTP_ANONYMOUS_ROOT.

The user is not placed automatically in this directory upon successful login.

TCPWARE_FTP_SEMANTICS_FIXED_IGNORE_CC

If this logical is defined to TRUE, then GET operations of fixed lengths record files will not have a carriage return/line feed added to the end of each record. The *?* in the logical represents where defined values go. Defined values can be either alpha or numeric.

```
$ DEFINE TCPWARE_FTP_SEMANTICS_FIXED_IGNORE_CC ?
```

TCPWARE_FTP_SEMANTICS_VARIABLE_IGNORE_CC

When this logical is defined to TRUE, files with variable length records and carriage return carriage control will NOT have a new line character inserted after each line when the file is transferred in image (binary) mode. The default is TRUE and is defined in FTPSERVER_DTP.COM.

```
$ DEFINE TCPWARE_FTP_SEMANTICS_VARIABLE_IGNORE_CC FALSE
```

Users can change this value by defining it in their LOGIN.COM file, or it can be defined on a system-wide basis if this is desired for all users.

TCPWARE_FTP_SEND_FEAT_ON_CONNECT

By default, the FTP client sends the FEAT command upon connecting to a server. This can be disabled by defining this logical as FALSE.

```
$ DEFINE TCPWARE_FTP_SEND_FEAT_ON_CONNECT FALSE
```

When this is disabled the FTP client will not be able to detect the support of optional features such as TLS, REST STREAM, and others and these features may not work correctly if there is an attempt to use them.

TCPWARE_FTP_SERVER_DATA_PORT_RANGE
Specifies the upper and lower port boundaries that are to be used in passive data connections. The string should contain two numbers separated by a space. The *?* in the logical represents where defined values go.

```
$ DEFINE TCPWARE_FTP_SERVER_DATA_PORT_RANGE ?
```

TCPWARE_FTP_SERVER_LOG_LIMIT
By setting this logical 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 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 TCPWARE_FTP_SERVER_LOG_LIMIT 42
```

TCPWARE_FTP_SERVER_RELAXED_PORT_COMMAND
The server compares the IP network address value specified in the PORT command with the IP network address of the IP address it is receiving commands from. If these are not in agreement, the PORT command is not accepted. Some multi-homed clients, and clients that can do third-party transfers, send values that do not match. Defining this logical allows the PORT command to be accepted for these clients by disabling this check. The *?* in the logical represents where defined values go. Defined values can be either alpha or numeric.

```
$ DEFINE TCPWARE_FTP_SERVER_RELAXED_PORT_COMMAND ?
```

TCPWARE_FTP_STRIP_VERSION
Causes VMS mode output to have no versions. The *?* in the logical represents where defined values go. Defined values can be either alpha or numeric.

```
$ DEFINE /SYSTEM /NOLOG /EXECUTIVE TCPWARE_FTP_STRIP_VERSION ?
```

TCPWARE_FTP_SYST_BANNER
When this logical is defined the system banner is not displayed in response to the `STATUS` command. When this logical is not defined the format of the banner varies depending upon whether the FTP server is operating in UNIX mode or VMS mode.

```
$ DEFINE /SYSTEM /NOLOG /EXECUTIVE TCPWARE_FTP_SYST_BANNER
```

TCPWARE_FTP_UNIX_STYLE_BY_DEFAULT
Starts the FTP server in UNIX emulation mode. The *?* in the logical represents where defined values go. Defined values can be either alpha or numeric.

```
$ DEFINE /SYSTEM/NOLOG/EXECUTIVE TCPWARE_FTP_UNIX_STYLE_BY_DEFAULT ?
```

When sending the command from a non-OpenVMS client, a space is required between the file specification and the qualifier.

```
$ GET filename /LOG
```

To disable this requirement:

```
$ DEFINE /SYSTEM /EXECUTIVE_MODE TCPWARE_FTPD_NOUNIX_SYNTAX "TRUE"
```

This logical has no effect if TCPWARE_FTP_DISALLOW_UNIX_STYLE is not set to FALSE.

TCPWARE_FTP_UNIX_STYLE_CASE_INSENSITIVE
Allows UNIX style filename handling to be case insensitive. The `?` in the logical represents where defined values go. Defined values can be either alpha or numeric.

```
$ DEFINE /SYSTEM/NOLOG/EXEC TCPWARE_FTP_UNIX_STYLE_CASE_INSENSITIVE
?
```

TCPWARE_FTP_USE_SRI_ENCODING_ON_ODS5
This logical can be defined to `1`, `TRUE` or `YES` to cause the filename encoding used for UNIX-style filenames on ODS-2 disks to be used on ODS-5 disks. This also sets the default case of letters in filenames to lowercase and ignores the stored case.

TCPWARE_FTP_WINDOW
The FTP client and the FTP server set the TCP window size of the data connection to either:

- The value of this logical if you define it (minimum is 512 bytes; maximum is 1,048,576 bytes)
- The larger of 32,768 bytes and the default TCP window size

The ? in the logical represents where defined values go. Defined value should be numeric.

```
$ DEFINE /SYSTEM /NOLOG /EXECUTIVE TCPWARE_FTP_WINDOW ?
```

TCPWARE_IMAP_UPDATE_LOGIN_TIME
If this logical is defined (to any value), then IMAP updates the user's "`Last login: (non-interactive)`" field on the server with the last time the user downloaded his/her mail via an IMAP client.

TCPWARE_LPD_DEFAULT_USER
Defines a default OpenVMS username for remote users connecting to the local LPD server. Used only when you define a remote host in the LPD access file, and the remote username is not mapped to a specific OpenVMS username.

TCPWARE_LPD_OPTIONS
Determines if the server handles batch queues.

TCPWARE_LPD_*qname*_*_FORM
Defines the form used for print jobs. This is similar to `TCPWARE_LPD_`*qname*`_PARAMETER`.

Use `TCPWARE_LPD_*_FORM` to define the form for all queues.

Note: A specific queue setting overrides the global setting for that queue.

TCPWARE_LPD_*qname*_OPTION
Specifies additional PRINT command qualifiers to pass to the specified print queue:

/BURST, /FEED, /FLAG, /FORM, /HEADER, /LOWERCASE, /PASSALL, /PRIORITY, /RESTART, /SPACE, /TRAILER

Use TCPWARE_LPD_*_OPTION to define the option for all queues.

Note: A specific queue setting overrides the global setting for that queue.

TCPWARE_LPD_*qname*_*_PARAMETER
Defines the specified parameters when the remote user submits a print request to the OpenVMS print system (*qname* is the queue name).

The first equivalence string for the logical (if defined) is the first parameter; the second is the second parameter; and so on, up to eight parameters.

Use TCPWARE_LPD_*_PARAMETER to define the parameter for all queues.

Note: A specific queue setting overrides the global setting for that queue.

TCPWARE_LPD_*qname*_*_QUEUE
Defines the print queues for an alias queue name (*qname*). Supports clients that may not allow standard OpenVMS queue names as the remote printer.

TCPWARE_LPD_SPOOL
Points to the work directory for the LPD server. This directory holds temporary files.

TCPWARE_LPR_PRINTER
Defines the default remote printer for the LPR, LPRM, and LPQ commands. Define your own TCPWARE_LPR_PRINTER logical in a LOGIN.COM file.

TCPWARE_LPR_*qname*_PRINTER
TCPWARE_LPR_*qname*_PRINTER_DEFAULT
Defines the absolute printer for the PRINT command. You cannot override this logical when submitting a print job. Use to restrict printing to one printer per queue.

TCPWARE_LPR_QUEUES
Lists the names of all TCPware print symbiont queues. Defined only if you defined one or more print queues.

TCPWARE_LPR_SPOOL
Points to the work directory for the PRINT command. This directory holds temporary files.

TCPWARE_LPRSM
The TCPWARE_LPRSMB print symbiont provides similar retry interval and timeout tuning logicals as those for TCPWARE_VMSLPRSMB. The TCPWARE_LPRSMB logicals are:

- TCPWARE_LPRSMB_*_RETRY_INTERVAL
- TCPWARE_LPRSMB_*qname*_RETRY_INTERVAL
- TCPWARE_LPRSMB_*_TIMEOUT
- TCPWARE_LPRSMB_*qname*_TIMEOUT
- TCPWARE_LPRSM_*qname*_PRECONN

TCPWARE_NAMED_MAX_CACHE_TTL
NAMED checks the SYSTEM EXECUTIVE logical table for this logical value and sets the maximum cache time (in seconds) to be that value. Use this logical to override the default one week (604800 seconds) to a maximum cache time more appropriate for your system.

```
$ DEFINE /SYSTEM /EXECUTIVE TCPWARE_NAMED_MAX_CACHE_TTL 86400
```

The server reads this logical the next time it starts. If you do not want to wait for the server to start, you can make the change to the running server by using the NETCU SET NAMED MAX_TTL command. Any data now written to the cache remains there for 86400 seconds (one day).

TCPWARE_NAMESERVERS
When an application needs to resolve a host name or internet address, the client queries the first name server this logical defines. The client continues to query the other name servers on its list until it receives an answer, or the list is exhausted.

TCPWARE_NFS_ACCESS_IDENTIFIER

Specifies the name of a rights identifier you want assigned to all NFS users. You can then modify the access control lists (ACLs) of files to grant or deny access to holders of the rights identifier. The default is null (no rights identifier).

OpenVMS files protected by ACLs should have the UIC-based protection mask set to allow file access and the ACL set to deny access.

TCPWARE_NFS_DFLT_GID
TCPWARE_NFS_DFLT_UID
Specifies the default UID and GID. The server uses these defaults in the following cases:

- Receives a request from a user without a PROXY mapping and who is also the superuser (UID=0, and any GID). The server replaces the superuser UID and GID with the default UID and GID.
- Processes a `get attributes` request and cannot find a file's owner UIC in the PROXY database. The server uses the default UID and GID instead.

TCPWARE_NFS_DIRLIFE_TIMER
Sets when to delete internal directory cache data structures. Specify the interval as OpenVMS delta time. The default is 3 minutes.

TCPWARE_NFS_DIRREAD_LIMIT
Sets the maximum size in bytes for each file read while processing a `get attributes` request. If the estimated file size exceeds this value, TCPware does not read the file to determine its exact size and returns an estimated size instead. The estimated file size is always larger than the exact size. The `-1` default turns off file size estimation.

This parameter applies only to filesystems exported with the `/CONVERT` option (the default). A value of `0` disables TCPware from determining exact file sizes on requests.

This parameter may provide the NFS client with inexact file sizes. This is not a problem but may affect some applications.

TCPWARE_NFS_DIRTIME_TIMER
Sets a time interval that determines when the server updates the directory access time between NFS operations. Specify the interval as an OpenVMS delta time. The default is 30 seconds.

TCPWARE_NFS_DYNAMIC_EXPORT
Reloads updates to the shared database on the cluster automatically when you set this logical to CLUSTER:

```
$ DEFINE /SYSTEM /EXECUTIVE TCPWARE_NFS_DYNAMIC_EXPORT CLUSTER
```

The server uses locks to communicate changes to all the servers on the cluster. The default is LOCAL (not to use locks).

TCPWARE_NFS_DYNAMIC_PROXY
Enables dynamic PROXY database reloading.

```
$ DEFINE /SYSTEM /EXECUTIVE -
_$ TCPWARE_NFS_DYNAMIC_PROXY keyword[,keyword]
```

The *keywords* are

- CLIENT - enables client reloading
- SERVER - enables server reloading
- NOCLIENT and NOSERVER - when used with the ADD PROXY or REMOVE PROXY commands overrides the logical setting

TCPWARE_NFS_FILE_CACHE_SIZE
Determines the maximum number of files allowed to have attributes in cache at any one time. The number must be larger than the SYSGEN parameter CHANNELCNT. The value must also be larger than the number of combined TCP and UDP threads.

TCPWARE_NFS_LOG_CLASS
Enables the type of information written to the log file TCPWARE:NFSSERVER.LOG. This parameter is a bit mask value (in decimal).

TCPWARE_NFS_NOCHECKSUM
Enables or disables checksum generation for UDP datagrams. This parameter is a Boolean value. When the value is 0 (false), the server generates checksums for outgoing datagrams. When the value is 1 (true), the server does not generate checksums. Enabling checksums maintains data integrity and is the default.

Note: Disabling checksums may increase system performance but could have an adverse affect on certain NFS clients.

TCPWARE_NFS_OPENFILE_TIMER
Sets a time interval (in delta time) a file remains open after you last accessed it. You do not need to open and close it for each request. The default is six seconds.

TCPWARE_NFS_PORT
Sets the TCP and UDP port through which the NFS and MOUNT protocols receive data.

TCPWARE_NFS_SECURITY
Enables various security features. This parameter is a decimal bit mask value.

TCPWARE_NFS_TCP_THREADS
Controls the number of simultaneously serviced requests received over TCP connections the server can support. The server requires a thread for each TCP request it receives. This thread is active for the amount of time it takes the server to receive the request, perform the operation, and send a reply to the client.

The more threads the server supports, the better the performance.

Note: The number of threads has no impact on the number of TCP connections the server supports.

TCPWARE_NFS_UDP_THREADS
This is similar to the `NFS_TCP_THREADS` parameter but relates to UDP threads.

TCPWARE_NFS_XID_CACHE_SIZE
Sets the maximum number of XID cache entries. The XID cache prevents the system from transmitting false error messages for operations such as delete, create, rename, and set attributes.

Set the `NFS_XID_CACHE_SIZE` parameter to at least twice (2 times) the largest of the number of:

- NFS clients using the NFS Server
- UDP threads (as set by the `NFS_UDP_THREADS` parameter)
- TCP threads (as set by the `NFS_TCP_THREADS` parameter)

The parameter sets the size of both the UDP and TCP XID caches (each protocol has a separate XID cache).

TCPWARE_POP3_UPDATE_LOGIN_TIME
If this logical is defined (to any value), then POP3 updates the user's "Last login: (non-interactive)" field on the server with the last time the user downloaded his/her mail via an POP3 client.

TCPWARE_PPPD_DEBUG_LEVEL
When you specify the `DEBUG` (or `-D`) option, it debugs at level 5 (display up to warning and significant events). For more informational and debugging information, raise the debug level to 7.

TCPWARE_PPPD_OPCOM_LEVEL
For a detached process, raise the message level for OPCOM messages. By default, it is set to 4 to report fatal and error messages. Raise it to 5 to monitor the significant events in PPPD, or even higher for more detail.

TCPWARE_QUOTE
Defines the quote for the server. This logical can be either a string or a filename that includes the quote text. Prefix a filename with the `@` sign and enclose the definition or filename in quotation marks.

You need SYSNAM or SYSPRV privileges to define the system-wide logical.

```
$ DEFINE/SYSTEM/EXECUTIVE TCPWARE_QUOTE "Quote-of-the-day"
$ DEFINE/SYSTEM/EXECUTIVE TCPWARE_QUOTE "@SYS$MANAGER:QUOTE.TXT"
$ DEFINE/SYSTEM/EXECUTIVE TCPWARE_QUOTE "Today's quote is",-
_$ "@SYS$MANAGER:QUOTE.TXT"
```

TCPWARE_RCMD_FLAGS
Set this logical to `1` (default is `0`) to disable user-specified `SYS$LOGIN:.RHOSTS` files (and use the `HOSTS.EQUIV` file only).

TCPWARE_RCMD_OUTPUT
Sets up a log file for incoming R Services such as RCP and RSH to log messages in the `RCMD.LOG` file:

```
$ DEFINE /SYSTEM /EXECUTIVE TCPWARE_RCMD_OUTPUT RCMD.LOG
```

TCPWARE_RES_OPTIONS *ndots ndots*
Sets up to six domains in a search list, as well as the minimum number of dots to recognize in a host name to make it fully qualified. The client reads this information from two logicals you set through CNFNET.

TCPWARE_RES_RETRANS_MIN
Specifies minimum retransmit time value in seconds.

TCPWARE_RES_RETRIES
Specifies retry count.

TCPWARE_SCP_VMS_MODE_BY_DEFAULT
When this logical is defined to `True`, `Yes`, or `1`, the SCP command defaults to `/VMS` if neither `/NOVMS` nor `/TRANSLATE_VMS` are specified.

TCPWARE_SCP2_CONNECT_TIMEOUT
This logical defines a number specifying how long SCP2 should wait for a response to the `INITIALIZE` command from the server program. This is a VMS delta time number. The default is 2 minutes.

TCPWARE_SCP2_VMS_MODE_BY_DEFAULT
When defined to `TRUE`, `YES`, or `1`, this logical chooses the `/VMS` qualifier if `/TRANSLATE_VMS` or `/NOVMS` has not been specified.

TCPWARE_SFTP_CASE_INSENSITIVE
This logical causes SFTP to treat filenames in a case insensitive manner when it is defined to `TRUE`, `YES`, or `1`.

TCPWARE_SFTP_FALLBACK_TO_CBT
When this logical is defined to `TRUE`, `YES`, or `1`, and files are being transferred in VMS mode, a contiguous file will be created as contiguous best try if there is insufficient space to create it as contiguous.

TCPWARE_SFTP_FILE_ESTIMATE_THRESHOLD
This logical controls the minimum number of blocks that a text file must be for an estimated transfer size to be returned instead of an exact size. The default is to estimate the transfer size for all text files.

TCPWARE_SFTP_DEFAULT_FILE_TYPE_REGULAR

If this logical is defined to `TRUE`, `YES` or `1`, then the SFTP server will use a default file type of `REGULAR` instead of `UNKNOWN` for `OPEN` operations. This can correct problems with filenames without a `.` (dot) in them getting `.dir` added to them. The filename will appear with a `.` at the end of the name in directory listings.

TCPWARE_SFTP_MAXIMUM_PROTOCOL_VERSION
This logical can be used to limit the version of the SSH File Transfer Protocol that the SFTP client and server use. This can sometimes provide a work-around for problems encountered with different implementations of the protocol. The default value is 4. Protocol versions 2 and 3 are also used by popular implementations.

TCPWARE_SFTP_NEWLINE_STYLE
This logical controls the newline style that SFTP uses. Which can be helpful in transferring text files. The values are: UNIX <lf>, VMS <lf>, MAC <cr>. If the logical is not defined, or defined to any other value, then <cr><lf> will be used for the text line separator as documented in the SSH File Transfer specification.

TCPWARE_SFTP_ODS2_SRI_ENCODING
This logical controls whether SRI encoding is used for filenames on VMS ODS-2 disks. If the logical is not defined, or is defined to `TRUE`, `YES`, or `1` then SRI encoding is used on ODS-2 disks for filenames that contain uppercase letters and special characters.

TCPWARE_SFTP_RETURN_ALQ
When defined to `TRUE`, `YES` or `1` and files are being transferred in VMS mode, this logical causes the allocation quantity to be transmitted when a file is transferred. Normally this value is only sent when necessary to avoid having an excessive amount of space allocated to a file when it is transferred from a disk with a large allocation cluster to a disk with a small allocation cluster.

TCPWARE_SFTP_TRANSLATE_VMS_FILE_TYPES *number*
When this logical is defined, the SFTP server will translate text files to stream linefeed format so that they are compatible with UNIX systems. The number is a bit mask, with the following definitions:

- bit 0 (value 1) FIXED format files should be translated
- bit 1 (value 2) VARIABLE format files should be translated
- bit 2 (value 4) VARIABLE, FIXED CONTROL (VFC) files should be translated.

These values can be added together to specify combinations of file types. Due to the way the SCP2 client is implemented, this logical also serves as a default for the SCP2 client.

The SCP-SERVER1 program always translates FIXED, VARIABLE and VFC files as it is designed to service requests that come from UNIX systems that use the OpenSSH implementation.

TCPWARE_SFTP_VMS_ALL_VERSIONS
This logical controls whether all versions of a file are returned. The values `TRUE`, `YES` or `1` will return all versions, any other value is to only return the name of the file without a version. The default is to return only one filename without the version number.

TCPWARE_SLIP_*n*
The START/IP command *line-specific-information* parameter provides the OpenVMS device name for the SLIP line. If you omit this parameter, TCPware assumes that the `TCPWARE_SLIP_`*`n`* system logical (where *`n`* is the controller number) defines the device.

TCPWARE_LOCALDOMAIN
Specifies the default local domain name to be used when building To: addresses on outgoing messages.

For example, to have messages sent to SMTP%"Joe@construction" to be delivered to SMTP%"Joe@construction.example.com", `TCPWARE_LOCALDOMAIN` would be defined as "example.com".

TCPWARE_NAMESERVERS
List of IP addresses for DNS lookups.

TCPWARE_SMTP_A1_NAME
Used in forming the username portion of return addresses for ALL-IN-1 users.

TCPWARE_SMTP_ACCEPT_UNIX_LF
Tells the SMTP agents to accept lines sent by some UNIX systems that are terminated with a linefeed only (instead of the proper carriage-return, linefeed combination).

TCPWARE_SMTP_ALLOW_USER_FROM
Allows users to override their From: address on outgoing mail by specifying `/FROM=xxx@yyy` as the first line of outgoing mail messages.

TCPWARE_SMTP_ALLOW_VIRTUAL_DOMAIN

Allows the use of virtual domains in TCPware SMTP environment. Without this logical defined, incoming aliases are assumed to be local addresses only. If your system supports multiple virtual domains and uses in the alias file to reroute traffic based on those domains, you must define this logical.

TCPWARE_SMTP_AM_DOMAIN
Domain name used when forming return addresses for ALL-IN-1 users.

TCPWARE_SMTP_AM_NAME
Used in forming the username portion of return addresses for ALL-IN-1 users.

TCPWARE_SMTP_APPEND_FORWARDER_TO_MX
Specifies that the default SMTP forwarder, if defined, is appended to the end of an MX list for a target host when delivering outgoing mail.

TCPWARE_SMTP_BATCH_QUEUE
Points to the TCPware SMTP queue.

TCPWARE_SMTP_DECNET_DOMAIN
Specifies a DECnet name used in the creation of return addresses.

TCPWARE_SMTP_DELIVERY_RECEIPTS
Enables or disables delivery receipts (value is `TRUE` or `FALSE`).

TCPWARE_SMTP_DISABLE_DELIVERY_RECEIPT_DISCLAIMER
When deliver receipts are enabled, a disclaimer is included in all such receipts telling the sender that the message has been delivered, but not necessarily read. Defining this logical prevents the disclaimer from being included.

TCPWARE_SMTP_DISABLE_FOLDER_DELIVERY
Disables TCPware SMTP's ability to deliver messages to user-defined folders in their VMS Mail files.

TCPWARE_SMTP_DISABLE_PSIMAIL
If defined, causes mail sent to `PSI%` users to be returned with `NOSUCHUSER`.

TCPWARE_SMTP_ENVELOPE_FROM_HOST
Specifies the host name to be used in the SMTP envelope MAIL FROM: line. If not defined, the default system host name is used.

TCPWARE_SMTP_FORWARDER

Specifies the domain name of the system to which all outgoing mail is forwarded for further delivery.

TCPWARE_SMTP_FROM_HOST
Specifies the local host name used when forming From: address on outgoing messages. If this logical is not defined, the system host name is used.

TCPWARE_SMTP_HEADER_ORG
Specifies the text for an Organization: header in outgoing mail.

TCPWARE_SMTP_HEADER_RETURN_RECEIPT_TO
Generates a Return-Receipt-To: header in outgoing mail. Requires the `TCPWARE_SMTP_RETURN_RECEIPT_TO_HEADER_ENABLE` logical to be defined.

TCPWARE_SMTP_HEADER_SYS
Specifies the text for a System: header in outgoing mail.

TCPWARE_SMTP_HOST_ALIAS_FILE
Points to the file containing a list of all the host names that should be considered local for this node for incoming mail delivery.

TCPWARE_SMTP_HOST_NAME
Specifies all the local host names for this node. Used to specify all virtual domains handled by this node. Alternatively, the node names can be stored in the file `TCPWARE:SMTP_HOST_ALIASES`.

TCPWARE_SMTP_LOG
Specifies the output filename. If not defined, the name defaults to `TCPWARE:TCPWARE_SMTP_LOG.`*`queuename`*.

TCPWARE_SMTP_MAXIMUM_822_TO_LENGTH
Sets the maximum length of the RFC822 To: header line when delivering incoming mail to VMS Mail users.

TCPWARE_SMTP_MRGATE_NAME
Specifies the name of the Message Router gateway.

TCPWARE_SMTP_NON_LOCAL_FORWARDER
Specifies the name of a forwarder system for non-local outgoing mail.

TCPWARE_SMTP_NO_USER_REPLY_TO

Disallows the use of user-defined Reply-To: headers in outgoing mail.
TCPWARE_SMTP_POSTMASTER Specifies the address of the system-wide postmaster.
TCPWARE_SMTP_REJECT_INVALID_DOMAINS Tells the SMTP server to reject mail from domains whose names and addresses cannot be resolved in a reverse lookup.
TCPWARE_SMTP_REPLY_TO Specifies an address for a Reply-To: header in outgoing mail.
TCPWARE_SMTP_RESENT_HEADERS Causes the inclusion of "Resent-*" headers in mail forwarded from a VMS Mail account using `SET FORWARD` in VMS Mail.
TCPWARE_SMTP_RETRY_INTERVAL Specifies the retry interval for messages waiting for an attempted redelivery. The time is specified as a delta time.
TCPWARE_SMTP_RETURN_INTERVAL Specifies the amount of time a given message delivery should be retried before giving up and bouncing the message back to the sender. The time is specified as a delta time.
TCPWARE_SMTP_RETURN_MSG Specifies an input filename for the return message SMTP sends when a mail message bounces.
TCPWARE_SMTP_RETURN_RECEIPT_TO_HEADER_ENABLE Enables the Return-Receipt-To: header if the `TCPWARE_SMTP_HEADER_RETURN_RECEIPT_TO` logical is also defined.
TCPWARE_SMTP_SEND_CLASS Specifies the VMS broadcast class for "New mail" notifications. The default is `USER16`.
TCPWARE_SMTP_SERVER_DISABLE_VRFYEXPN Disables the VRFY and EXPN commands in bitmask format to the SMTP server. Bit 0 = VRFY; Bit 1 = EXPN.
TCPWARE_SMTP_SERVER_LOG

Enables debug logs for the SMTP server.

TCPWARE_SMTP_SERVER_RCPT_CHECK_HOST
The host name to be used in checking for local host when passing messages through the reject rules.

TCPWARE_SMTP_SERVER_REJECT_FILE
Points to the file containing the rejection rules.

TCPWARE_SMTP_SERVER_REJECT_INFO
Specifies the level of OPCOM messages generated by the rejection rules for incoming SMTP mail. If not defined, no messages are generated.

TCPWARE_SMTP_SUPPRESS_VENDOR
Suppresses the vendor name in the SMTP server welcome banner. Define this logical to hide the fact that the system is a VMS system running TCPware.

TCPWARE_SMTP_SYMBIONT_LOG
Enables debug logs for the SMTP symbiont.

TCPWARE_SMTP_SYMBIONT_PURGWS_TIMER
Specifies how often the SMTP symbiont purges its working set to free up unneeded memory. The time is specified as a delta time.

TCPWARE_SMTP_WINDOW_SIZE
Specifies the window size used in TCP connections when delivering mail.

TCPWARE_SNMP_DEBUG
SNMP subagent developers uses this logical to set certain debug masks.

```
$ DEFINE TCPWARE_SNMP_DEBUG mask
```

TCPWARE_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 VMS 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.

TCPWARE_SSH_ALLOW_PREEXPIRED_PW

(SSH1) allows logging in to an account when the password has been pre-expired. This applies to all users and circumvents normal VMS 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.

TCPWARE_SSH_KEYGEN_MIN_PW_LEN
(SSH1) defines the minimum passphrase length when one is to be set in SSHKEYGEN. If not defined, defaults to zero. Defined by @TCPWARE2CNFNET SSH.

TCPWARE_SSH_PARAMETERS_*n*
These parameters are used to start SSHD_MASTER. They are parameters set by @TCPWARE:CNFNET SSH.

TCPWARE_SSH_USE_SYSGEN_LGI
(SSH1) if defined, causes SSHD to use the VMS SYSGEN value of LGI_PWD_TMO to set the login grace time, overriding anything specified in the command line or the configuration file.

TCPWARE_SVCORDER
Contains the list of services used in the order specified.

Use the values "bind,local" (the default if the logical is not defined) and "local,bind" (uses DNS if the Hosts database lookup fails).

TCPWARE_VMSMAIL_HEADER_CONTROL
Specifies how many RFC822 headers are included in mail delivered to VMS Mail users. Values can be ALL, MAJOR, and NONE.

TCPWARE_VMSMAIL_LOCASE_USERNAME
Lowercases the username portion of outgoing addresses.

TCPWARE_VMSMAIL_NO_EXQUOTA
Delivers incoming mail to local VMS Mail users without using EXQUOTA.

TCPWARE_VMSMAIL_REPLY_CONTROL
Specifies which header to use to determine the sender of a message ("Reply-To:" or "From:").

TCPWARE_VMSMAIL_USE_RFC822_TO_HEADER

Sets the maximum length of the RFC822 To: header line when sending outgoing mail. The default is 1024. The range can be set from 256 to 65535.

TCPWARE_TCLB_BIAS
Define this logical with a multiplier and an addend as two values of the logical. Both are real numbers.

You can use these values to bias a load offered to the host. For example, the following command doubles the observed load and adds 1.5 users:

```
$ DEFINE /SYSTEM TCPWARE_TCLB_BIAS "2.0","1.5"
```

TCPware re-translates this logical before it sends each response. This means that some other process can change it dynamically or you can set it statically.

TCPWARE_TELNET_WINDOW
Specifies the window size that the TELNET server offers to the peer. The default value is 4096. If the value is less than 512, TELNET uses 4096.

TCPWARE_TELNETD_DEFCHAR
Sets up the default terminal characteristics for TELNET sessions. You can avoid having to change the `SYSGEN TTY_DEFCHAR` and `TTY_DEFCHAR2` fields system-wide. This logical forces the hangup bit set. To prevent the forcing of the hangup bit set, use the `TCPWARE_TELNETD_NO_FORCED_HANGUP` logical.

TCPWARE_TELNETD_FLAGS
Setting either bit 0 or 1 can improve server performance and reduce system processing overhead. The default value is 1.

Note: Doing so means you are not adhering to the TELNET protocol.

TCPWARE_TELNETD_INTRO_MSG
Defines a special message that appears whenever a user attempts access to the host through TELNET. Use this logical to issue warnings such as "Authorized Use Only" for remote logins.

TCPWARE_TIMED_EXCLUDE

Determines the networks excluded from clock synchronization, either in network addresses or names.

TCPWARE_TIMED_INCLUDE
Determines the networks included in clock synchronization, either in network addresses or names.

TCPWARE_TIMED_MODE
Determines if the current host is a MASTER, FIXED MASTER, or SLAVE.

- MASTER (primary) - broadcasts time synchronization requests, calculates the time differences and averages, and sends "adjust time" messages.
- FIXED MASTER (fixed primary) - provides absolute time stamps to newly started dependent TIMED hosts.
- SLAVE (dependent) - is the recipient of primary "adjust time" messages.

TCPWARE_TIMEZONE
This logical can have two equivalence strings:

- *+hhmmss*
  hh are the hours *mm* are the minutes *ss* are the seconds offset from the universal time (UT).

  + is for east of the central meridian, − is for west. For example: +04:00:00 is four hours east of the central meridian at Greenwich.

  Another example: eastern standard time (EST) is five hours west of UT, so the offset is −0500.

- *name* an optional name for the time zone. For example: EDT for Eastern Daylight time. Can be one of the following:
  Universal Time - UT, UTC, or GMT
  North American Time - EST, EDT, CST, CDT, MST, MDT, PST, PDT
  Military Time - Any single uppercase letter A through Z except J (this format is not recommended)

  Any other character sequence

  The *name* is not validated and may be used by applications to report the local time zone.

TCPWARE_TSSYM_*qname*
Defines the parameters normally set with the /ON qualifier. Since you cannot use /AUTOSTART_ON together with the /ON qualifier to initialize a terminal server print queue, you need to define TCPWARE_TSSYM_*qname* for this purpose.

```
$ DEFINE /SYSTEM TCPWARE_TSSYM_qname "host,port[,option...]"
```

TCPWARE_TSSYM_*_ RETRY_INTERVAL
Defines the interval at which the symbiont retries to make a connection to a printer after an attempt fails. The default is 0::15 (15 seconds delta time).

TCPWARE_TSSYM_*_TIMEOUT
Defines the time it takes for a print job to abort if the connection to the printer is never established. The default timeout is infinite (it never times out).

TCPWARE_TSSYM_*qname*_RETRY_INTERVAL
Same as `TCPWARE_TSSYM_*_RETRY_INTERVAL`, but for a specific queue only, and overrides `TCPWARE_TSSYM_*_RETRY_INTERVAL`.

TCPWARE_TSSYM_*qname*_TIMEOUT
Same as `TCPWARE_TSSYM_*_TIMEOUT`, but for a specific queue only, and overrides `TCPWARE_TSSYM_*_TIMEOUT`.

TCPWARE_VMSLPRSMB_*qname*_PRECONN
Makes the connection to the printer *before* processing the file. Normal behavior is to make the connection to the printer *after* processing the file.

TCPWARE_VMSLPRSMB_*qname*_RETRY_INTERVAL
Same as `TCPWARE_VMSLPRSMB_*_RETRY_INTERVAL`, but for a specific queue only, and overrides `TCPWARE_VMSLPRSMB_*_RETRY_INTERVAL`.

TCPWARE_VMSLPRSMB_*qname*_TIMEOUT
Same as `TCPWARE_VMSLPRSMB_*_TIMEOUT`, but for a specific queue only, and overrides `TCPWARE_VMSLPRSMB_*_TIMEOUT`.

TCPWARE_VMSLPRSMB_*_RETRY_INTERVAL
Defines the interval at which the symbiont retries to make a connection to a printer after an attempt fails. The default value for a retry interval is 2 minutes (`:2` in delta time).

Note: A connection failure can take 1.5 minutes to time out, which is not included in this interval value.

TCPWARE_VMSLPRSMB_*_TIMEOUT Defines the time it takes for a print job to abort if the connection to the printer is never established. The default timeout is infinite (it never times out).
UCX$DEVICE Defined as `BG:` (the name of the UCX device drive).
UCX$INET_HOST Defined to be the host name (the same setting as `TCPWARE_DOMAINNAME`).
UCX$IPC_SHR Provides the linkage to the TCPware version of the `UCX$IPC_SHR` run-time library.

Appendix C. DNSSEC

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.

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.

It's required to 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 :== $tcpware: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.

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 public key (in the `.key` file) is used for signature verification. 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. **Process Software 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 public keys can be inserted into the zone file by pasting in their contents, or better yet by including the `.key` file using `$include` statements. For example, to insert the public key for `child-example`, add the following `$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 ;
```

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 :== $tcpware: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`, should look something like this:

```
; 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.

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 gets established, a typical DNSSEC configuration will look something like the following. It has one or more public keys 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 which 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.

> **Note:** 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. It does this by verifying, via signed and validated NSEC/NSEC3 records, that 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`".

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.

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 "tcpware_common:[tcpware]";
};
```

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.

Dynamic DNS update method

To insert the keys via dynamic update:

```
$ nsupdate :== $tcpware: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 won't 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.

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 a `rndc sign` *`zonename`* or `rndc loadkeys` *`zonename`* command.

`auto-dnssecmaintain` 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 a `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.

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 which 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. The octet meanings are:

- 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

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.

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.

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.

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.

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.

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.

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).

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 so as to spread the re-sign load over time rather than all at once.

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.

Dynamic Trust Anchor Management

BIND 9.7.0 introduced 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.

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.

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 which 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.