OpenVOS STREAMS TCP/IP Migration Guide

Stratus Technologies
R418-06

Notice

The information contained in this document is subject to change without notice.

UNLESS EXPRESSLY SET FORTH IN A WRITTEN AGREEMENT SIGNED BY AN AUTHORIZED REPRESENTATIVE OF STRATUS TECHNOLOGIES, STRATUS MAKES NO WARRANTY OR REPRESENTATION OF ANY KIND WITH RESPECT TO THE INFORMATION CONTAINED HEREIN, INCLUDING WARRANTY OF MERCHANTABILITY AND FITNESS FOR A PURPOSE. Stratus Technologies assumes no responsibility or obligation of any kind for any errors contained herein or in connection with the furnishing, performance, or use of this document.

Software described in Stratus documents (a) is the property of Stratus Technologies Bermuda, Ltd. or the third party, (b) is furnished only under license, and (c) may be copied or used only as expressly permitted under the terms of the license.

Stratus documentation describes all supported features of the user interfaces and the application programming interfaces (API) developed by Stratus. Any undocumented features of these interfaces are intended solely for use by Stratus personnel and are subject to change without warning.

TCP Wrappers copyright information:

Manual Name: *OpenVOS STREAMS TCP/IP Migration Guide*

Part Number: R418
Revision Number: 06
OpenVOS Release Number: 17.1.0
Publication Date: April 2011

Stratus Technologies, Inc.
111 Powdermill Road
Maynard, Massachusetts 01754-3409

Contents

Tables

Preface

The *OpenVOS STREAMS TCP/IP Migration Guide* (R418) highlights differences between OS TCP/IP and STCP. It documents new and changed configuration commands and administrative procedures, changes to the application programming interface (API), and changes to the user interface.

This manual is intended for the following audience.

- System or network administrators should read Chapter 2. This chapter may also be useful to systems programmers who are designing distributed applications or networking software with specific configuration and/or administration requirements.
- Programmers who are writing C applications to run on an OpenVOS module should read Chapter 3.
- Users requiring FTP, TELNET, and/or TFTP on OpenVOS STREAMS-based I/O subsystems should read Chapter 4.

Before using the *OpenVOS STREAMS TCP/IP Migration Guide* (R418), you should be familiar with the following manuals.

- *OpenVOS STREAMS TCP/IP Administrator's Guide* (R419)
- *OpenVOS STREAMS TCP/IP Programmer's Guide* (R420)
- *VOS STREAMS TCP/IP User's Guide* (R421)

For information about OpenVOS STREAMS, see the *OpenVOS Communications Software: STREAMS Programmer's Guide* (R306).

For information about the OpenVOS Standard C implementation of the C language, see the *OpenVOS Standard C Reference Manual* (R363) and the *OpenVOS Standard C User's Guide* (R364).

Manual Version

This manual is a revision. Change bars, which appear in the margin, note the specific changes to text since the previous publication of this manual. Note, however, that change bars are not used in new chapters or appendixes.

This revision incorporates the following changes:

- The location of STCP header files that are not obsolete and are not for the compatibility library has changed. In earlier OpenVOS and VOS releases, these files were located in the `(master_disk)>system>stcp>include_library` directory or its subdirectories. These files are now located in the `(master_disk)>system>include_library` directory. Thus, references to the `(master_disk)>system>stcp>include_library` directory and its subdirectories have been removed or have changed, and the following information has been updated in Chapter 3:
  - “Function Prototype” on page 3-19
  - “Including Header Files” on page 3-19
- In Chapter 2, the section “Adding STCP to a Module Running OS TCP/IP” has been removed because releases of OpenVOS and VOS that run on ftServer modules do not support OS TCP/IP.
- In Chapter 3, the section “Adding the STCP Object Library to the Library Search Paths” has been removed because you no longer need to add the `(master_disk)>system>stcp>object_library>net` and `(master_disk)>system>stcp>object_library>common` directories to your object library search paths. Objects that were formerly in `(master_disk)>system>stcp>object_library>net` have been exported to the C runtime library (`(master_disk)>system>c_object_library`) and the POSIX runtime library (`(master_disk)>system>posix_object_library`). In addition, the functions in `(master_disk)>system>stcp>object_library>common` are no longer called by functions from any Stratus-supplied libraries and are no longer documented or supported.
- The following sections have been updated:
  - the introductory information in Chapter 3
  - “The STCP Response to the `accept` Function” on page 3-4
  - “Unsupported OS TCP/IP Socket Options” on page 3-10
  - “Adding the Compatibility Library to the Library Search Paths” on page 3-11
  - “Converting Source Code to STCP” on page 3-14

Manual Organization

This manual contains the following chapters.

Chapter 1 provides an overview of migrating from OS TCP/IP to STCP.

Chapter 2 identifies the differences between the OS TCP/IP and STCP administrative commands and procedures.

Chapter 3 explains how to convert a program from the OS TCP/IP API to the STCP API. It also describes differences between the OS TCP/IP and STCP APIs.

Chapter 4 discusses differences between the OS TCP/IP and STCP user interfaces.

Appendix A shows how STCP error codes map to OS TCP/IP error codes.

Related Manuals

See the *VOS Data-Link Multiplexer Administrator's Guide* (R425) for information about the STREAMS Data-Link Multiplexer (SDLMUX).

See the following Stratus manuals for information about OS TCP/IP.

- *VOS Communications Software: OS TCP/IP User's Guide* (R222)
- *VOS OS TCP/IP Administrator's Guide* (R223)
- *VOS Communications Software: OS TCP/IP Programmer's Manual* (R224)

See the following manuals for information about ftServer hardware.

- the site planning guide for your system:
  - *Stratus ftServer V 100, V 200, and V 400 Systems: Site Planning Guide* (R543)
  - *Stratus ftServer V 150, V 250, V 300, V 500, and V 502 Systems: Site Planning Guide* (R605)
  - *Stratus ftServer V 2302, V 4304, and V 6308 Systems: Site Planning Guide* (R624)
  - *Stratus ftServer V 2404, V 4408, and V 6408 Systems: Site Planning Guide* (R645)
- the operation and maintenance guide for your system:
  - *Stratus ftServer V 100 Systems: Operation and Maintenance Guide* (R581)
  - *Stratus ftServer V 200 and V 400 Systems: Operation and Maintenance Guide* (R557)

  - *Stratus ftServer V 150, V 250, V 300, V 500, and V 502 Systems: Operation and Maintenance Guide* (R606)
  - *Stratus ftServer V 2302, V 4304, and V 6308 Systems: Operation and Maintenance Guide* (R625)
  - *Stratus ftServer V 2404, V 4408, and V 6408 Systems: Operation and Maintenance Guide* (R646)

- *Stratus ftServer Systems Peripherals Site Planning Guide* (R582), which also provides information about disks and tapes

See the following Stratus manuals for information about OpenVOS administrative procedures.

- *OpenVOS System Administration: Administering and Customizing a System* (R281)
- *VOS System Administration: Starting Up and Shutting Down a Module or System* (R282)
- *OpenVOS System Administration: Registration and Security* (R283)
- *OpenVOS System Administration: Disk and Tape Administration* (R284)
- *OpenVOS System Administration: Backing Up and Restoring Data* (R285)
- *VOS System Administration: Administering the Spooler Facility* (R286)
- *OpenVOS System Administration: Configuring a System* (R287)

See the following Stratus manuals for additional OpenVOS information.

- *OpenVOS Commands Reference Manual* (R098)
- *OpenVOS Installation Guide* (R386)

See also the following third-party documentation.

- *TCP/IP Illustrated*. Volumes 1 and 2. Gary Wright and W. Richard Stevens (Reading, MA: Addison-Wesley Publishing Company, 1995).
- TCP/IP Request for Comments (RFCs), available on the Internet.

Notation Conventions

This manual uses the following notation conventions.

Warnings, Cautions, and Notes

Warnings, cautions, and notes provide special information and have the following meanings:

WARNING

A warning indicates a situation where failure to take or avoid a specified action could cause bodily harm or loss of life.

CAUTION

A caution indicates a situation where failure to take or avoid a specified action could damage a hardware device, program, system, or data.

NOTE

A note provides important information about the operation of a Stratus system.

Typographical Conventions

The following typographical conventions are used in this manual:

- Italics introduces or defines new terms. For example:

  The *master disk* is the name of the member disk from which the module was booted.

- Boldface emphasizes words in text. For example:

  Every module **must** have a copy of the `module_start_up.cm` file.

- Monospace represents text that would appear on your terminal's screen (such as commands, subroutines, code fragments, and names of files and directories). For example:

  ```
  change_current_dir (master_disk)>system>doc
  ```

- Monospace italic represents terms that are to be replaced by literal values. In the following example, the user must replace the monospace-italic term with a literal value.

```
list_users -module module_name
```

- Monospace bold represents user input in examples and figures that contain both user input and system output (which appears in monospace). For example:

```
display_access_list system_default

%dev#m1>system>acl>system_default

w  *.*
```

Online Documentation

The OpenVOS StrataDOC Web site is an online-documentation service provided by Stratus. It enables Stratus customers to view, search, download, print, and comment on OpenVOS technical manuals via a common Web browser. It also provides the latest updates and corrections available for the OpenVOS document set.

You can access the OpenVOS StrataDOC Web site, at no charge, at http://stratadoc.stratus.com. A copy of OpenVOS StrataDOC on supported media is included with this release. You can also order additional copies from Stratus.

For information about ordering OpenVOS StrataDOC on supported media, see the next section, “Ordering Manuals.”

Ordering Manuals

You can order manuals in the following ways.

- If your system is connected to the Remote Service Network (RSN™), issue the `maint_request` command at the system prompt. Complete the on-screen form with all of the information necessary to process your manual order.
- Contact the Stratus Customer Assistance Center (CAC), using either of the following methods:
  - From the ActiveService Manager (ASM) web site, log on to your ASM account. Click the `Manage Issues` button, enter your site ID in the `Create New Issue For Site` box, and then click the pencil icon to the right of the box. Fill out the forms and select `Update`.
  - Customers in North America can call the CAC at (800) 221-6588 or (800) 828-8513, 24 hours a day, 7 days a week. All other customers can contact their nearest Stratus sales office, CAC office, or distributor; see

http://www.stratus.com/support/cac/index.htm for Stratus CAC phone numbers outside the U.S.

Manual orders will be forwarded to Order Administration.

Commenting on This Manual

You can comment on this manual using one of the following methods. When you submit a comment, be sure to provide the manual's name and part number, a description of the problem, and the location in the manual where the affected text appears.

- From StrataDOC, click the `site feedback` link at the bottom of any page. In the pop-up window, answer the questions and click `Submit`.
- From any email client, send email to `comments@stratus.com`.
- From the ASM web site, log on to your ASM account and create a new issue as described in the preceding section, "Ordering Manuals."
- From an OpenVOS window, specify the command `comment_on_manual`. To use the `comment_on_manual` command, your system must be connected to the RSN. This command is documented in the manual *OpenVOS System Administration: Administering and Customizing a System* (R281) and the *OpenVOS Commands Reference Manual* (R098). You can use this command to send your comments, as follows.
  - If your comments are brief, type `comment_on_manual`, press Enter or Return, and complete the data-entry form that appears on your screen. When you have completed the form, press Enter.
  - If your comments are lengthy, save them in a file before you issue the command. Type `comment_on_manual` followed by `-form`, then press Enter or Return. Enter this manual's part number, `R418`, then enter the name of your comments file in the `-comments_path` field. Press the key that performs the `CYCLE` function to change the value of `-use_form` to `no` and then press Enter.

    NOTE

    If `comment_on_manual` does not accept the part number of this manual (which may occur if the manual is not yet registered in the `manual_info.table` file), you can use the `mail` request of the `maint_request` command to send your comments.

Your comments (along with your name) are sent to Stratus over the RSN.

Stratus welcomes any corrections and suggestions for improving this manual.

Chapter 1
Overview of Migrating from OS TCP/IP to STCP

STREAMS TCP/IP (STCP) is a STREAMS-based implementation of the standard Transmission Control Protocol/Internet Protocol (TCP/IP) family of communications protocols for OpenVOS. Unlike the older VOS OS TCP/IP product set, STCP functions within a standard STREAMS environment.

This manual highlights the differences between OS TCP/IP and STCP. It documents new and changed configuration commands and administrative procedures, changes to the application programming interface (API), and changes to the user interface.

This manual discusses the following topics.

- Chapter 2, ‘‘Commands and Administrative Procedures,’’ identifies the differences between the OS TCP/IP and STCP administrative commands and procedures.
- Chapter 3, ‘‘Converting OS TCP/IP Programs to STCP,’’ discusses the differences between the OS TCP/IP and STCP application programming interfaces, and describes the STCP compatibility library that eases the conversion of an application from OS TCP/IP to STCP.
- Chapter 4, ‘‘Differences in the FTP User Interface,’’ discusses the differences between the OS TCP/IP and STCP user interfaces.

See the following manuals for more information about STCP.

- The *OpenVOS STREAMS TCP/IP Administrator’s Guide* (R419) explains how to configure and manage STCP.
- The *OpenVOS STREAMS TCP/IP Programmer’s Guide* (R420) documents the STCP API.
- The *VOS STREAMS TCP/IP User’s Guide* (R421) explains how to use the File Transfer Program (FTP), TELNET, and Trivial File Transfer Program (TFTP) protocols with STCP.

Chapter 2
Commands and Administrative Procedures

This chapter, which contains the following sections, identifies major differences between OS TCP/IP and STCP commands and administrative procedures.

For more information about STCP commands and administrative procedures, see the *OpenVOS STREAMS TCP/IP Administrator's Guide* (R419). For more information about OS TCP/IP commands and administrative procedures, see the *VOS OS TCP/IP Administrator's Guide* (R223).

Basic Differences between OS TCP/IP and STCP

Table 2-1 summarizes the basic differences between OS TCP/IP and STCP.

Table 2-1. Basic Differences between OS TCP/IP and STCP *(Page 1 of 2)*

Issue/Feature	OS TCP/IP	STCP
Protocol environment	Kernel	STREAMS
Product design	Three products: OS TCP/IP Protocol Support, Utilities, and Application Support	One product containing protocol stack, utilities, and application programming components
Supported hardware platforms	XA/R and Continuum	XA/R, Continuum, and ftServer
Support for IP multicast level 2 (send and receive)	Not available	Available

Table 2-1. Basic Differences between OS TCP/IP and STCP *(Page 2 of 2)*

Issue/Feature	OS TCP/IP	STCP
Support for related products	Network File System (NFS™), SMTP Gateway of Office/2000, and Open StrataLINK	OpenVOS Open StrataLINK is supported by STCP, but NFS† and the SMTP Gateway of Office/2000 are not.
Support for variable-length subnet configurations (VLSN)	Not available	Available
Support for Simple Network Management Protocol (SNMP)	Not available	Available with the `snmpd` daemon or the product EMANATE for VOS
Support for dynamically loading and unloading the stack	Support for dynamically loading the stack is available; support for dynamically unloading the stack is not available	Both are available with the `start_stcp.cm` and `stop_stcp.cm` command macros
Support for the Open Shortest Path First (OSPF) protocol	Not available	Available
Support for developing POSIX-compliant socket applications	Not available	Available
TCP Wrappers functionality	Not available	Available

† Remote Procedure Calls (RPC) may be a suitable replacement for NFS; see the *Software Release Bulletin: RPC and XDR for VOS* (R584).

NOTE

STCP also supports clonable drivers and clonable login and slave devices for TELNET. You can use these clone devices **only** if all modules in the system are running VOS Release 14.1.0 or later.

Differences in Commands

Table 2-2 summarizes the differences between OS TCP/IP and STCP commands.

Table 2-2. Differences between OS TCP/IP and STCP Commands *(Page 1 of 5)*

OS TCP/IP Command	STCP Command	Functionality Differences
`arp`	`arp`	In OS TCP/IP, `arp` displays only one ARP entry per command; in STCP, `arp` displays the entire ARP table with the `-all` argument. In STCP, `arp` can specify a network/subnet with the `-network` argument; in OS TCP/IP, `arp` does not offer this argument. The output of the STCP version includes additional fields not supported in the OS TCP/IP version.
`ftp`	`ftp`	The STCP version offers an `-autologin` argument; the OS TCP/IP version does not. (Note that OS TCP/IP does support automatic login, however.)
`ifconfig`	`ifconfig`	You issue the OS TCP/IP version twice to add and then start the network interfaces. The command appears twice for each logical interface in the `module_start_up.cm` file. You issue the STCP version once for each logical interface in the `start_stcp.cm` command macro to start each network interface. (The command is issued after the stack-related commands.) The OS TCP/IP and STCP versions both enable you to add, delete, or check the status of a network interface during the current bootload, after all components are running. Both versions can also mark the state of the interface `up` or `down` with the `-state` argument. However, the STCP version does not require you to mark the state `down` before you delete the interface, or mark the state `up` after you add an interface. STCP also supports the `-all` argument, which displays a list of all network interfaces configured for STCP on the module. STCP also supports the `-alias` argument, which adds an additional IP address to a network interface.

Table 2-2. Differences between OS TCP/IP and STCP Commands *(Page 2 of 5)*

OS TCP/IP Command	STCP Command	Functionality Differences
`ifconfig` (continued)	`ifconfig` (continued)	The STCP version enables you to set flags such as `-forwb` (forward broadcast) and `-kalive` (keepalive). The OS TCP/IP version allows you to set such flags but does not provide control arguments that allow you to shut keepalive off regardless of what the program wants and that allow forwarding but not forwarding of broadcasts. The OS TCP/IP version supports the `-device` argument; the STCP version does not.
Not available	`IP_forwarding`	This command enables you to display or change the setting of the IP forwarding feature on the module (which determines whether the module can act as a gateway/router). By default, the module can act as a gateway/router.
`netstat`	`netstat`	The STCP version supports the `-network` and `-multicast` arguments; the OS TCP/IP version does not support these arguments but does support the `-brief`, `-summary`, `-display_zeros`, and `-device` arguments, which the STCP version does not support. The STCP version does not support the `-routing -numeric` format that displays route information with network interfaces; instead, the `-all` argument of the `ifconfig` command displays a list of all of the network interfaces configured for STCP. Issuing the STCP version with the `-interface` argument and an interface name provides output similar to that of the OS TCP/IP version issued with the `-detail` argument. The STCP version supports the `-PCB_addr` argument, which performs the same function as the OS TCP/IP version's `-A` argument. The STCP version produces output in a different format, and the output includes the subnet mask. The STCP and OS TCP/IP versions produce different protocol statistics.

Table 2-2. Differences between OS TCP/IP and STCP Commands *(Page 3 of 5)*

OS TCP/IP Command	STCP Command	Functionality Differences
Not available	`ospfd`	This is a new STCP command that starts the OSPF daemon. OSPF is a link-state routing protocol. For more information about OSPF, refer to the *OpenVOS STREAMS TCP/IP Administrator's Guide* (R419).
Not available	`omon`	This is a new STCP command that allows an administrator to monitor OSPF on a module that is running the `ospfd` daemon.
`packet_monitor`	`packet_monitor`	The OS TCP/IP version supports the `-queue` argument from the main arguments and the `-loopback` argument when you specify a filter; the STCP version does not. The `-pkt_hdr` and `-hex_header` arguments are secret arguments with the OS TCP/IP version but appear in the STCP version's display form. The STCP version also accepts UNIX® style abbreviations for most of the arguments shown in the display form (for example, you can specify `-v` instead of `-verbose`).
`ping`	`ping`	The OS TCP/IP version can ping only one host at a time and sends 20 packets by default. The STCP version can ping several hosts at once and sends 4 packets by default. The OS TCP/IP version supports the `-rt_bypass`, `-device`, and `-verbose` arguments; the STCP version does not. The STCP version supports the `-silent_return`, `-run_forever`†, and `-timeout` arguments; the OS TCP/IP version does not. In addition, the STCP version includes the `-record_route` argument, which provides support for recording and manipulating routes to a destination if a `.flash_ping` file (an empty file) created by the system administrator exists; the OS TCP/IP version does not provide this support.

Table 2-2. Differences between OS TCP/IP and STCP Commands *(Page 4 of 5)*

OS TCP/IP Command	STCP Command	Functionality Differences
`route`	`route`	The STCP version supports more options than the OS TCP/IP version. The STCP version enables you to print the routing table, change a route entry, and specify a subnet mask (for example, to specify a mask for a variable-length subnet configuration). The OS TCP/IP version supports the *`route_type`*, `-device`, and `-numeric` arguments, while the STCP version adds options to the *`command`* argument, eliminates the *`route_type`*, `-device`, and `-numeric` arguments, and adds the `-default_gateway` argument instead of specifying the value `default` in the *`destination`* argument. (When you specify a default gateway with STCP, an asterisk (`*`) must precede the `-default_gateway` argument to represent the destination.)
`sethost`	`hostname`	The OS TCP/IP version supports the `-device` argument; the STCP version does not. Otherwise, these two commands perform the same function: to define the module's host name. The STCP `hostname` command generates a file called `host`. (**Never delete this file.**)
`telnet`	`telnet`	The command interface is the same.
Not available	`telnet_admin`	This is a new STCP command for adding, listing, modifying, or deleting TELNET services that are available on the module. This command updates both the `services` and `telnetservice` database files.

Table 2-2. Differences between OS TCP/IP and STCP Commands *(Page 5 of 5)*

OS TCP/IP Command	STCP Command	Functionality Differences
`tftp`	`tftp`	The command interface is the same, but the default directories for using the `put` and `get` subcommands are different. With the OS TCP/IP version: • If you specify the `put` subcommand and do not specify a directory, `tftp` puts the file in the directory from which `tftpd` was issued. • If you specify the `get` subcommand and do not specify a directory, `tftp` gets the file from the directory in which `tftp.pm` resides. With the STCP version: • If you specify the `put` or `get` subcommand and do not specify a directory, `tftp` puts the file in or gets the file from the default directory, which is `(master_disk)>system>stcp>tftp_default`.
Not available	`traceroute`	This is a new STCP command that traces the route that an IP packet follows to the specified network host.

† In STCP, a ping packet is sent out as soon as the reply from the previous packet arrives. As a result, if you specify the `ping` command's `-run_forever` argument, the command will flood the network with packets instead of sending just one packet per second.

Differences in Configuration Procedures

Table 2-3 summarizes the major configuration differences between OS TCP/IP and STCP.

Table 2-3. Differences between OS TCP/IP and STCP Configuration Procedures *(Page 1 of 3)*

Configuration Procedure	OS TCP/IP	STCP
Creating device entries for the protocol stack	One entry for the OS TCP/IP protocol driver.	One entry for each of the following drivers: STCP, UDP, IP, and Loopback.
Defining network interfaces for STCP	Device entries are based on hardware type/driver (and partnering, as of VOS Release 13.2.0). Issue a set of `ifconfig` commands during startup to add and start each logical interface.	Device entries are based on hardware type and partnering. Issue one `ifconfig` command from the `start_stcp.cm` command macro to add and start each logical interface.
Loading drivers	The protocol driver and the OS TCP/IP kernel-loadable library are explicitly loaded from the directory `(master_disk)>system>tcp_os`. Prior to VOS Release 13.4.0, lower-level drivers such as `gdl.pm`, `bdl.pm`, and `dlmux.pm` were explicitly loaded from `(master_disk)>system>tcp_os`; in VOS Release 13.4.0 and later, they are explicitly loaded from `(master_disk)>system>kernel_loadable_library`. Associated communications drivers (such as OpenVOS STREAMS and `telnet_al`) must be explicitly loaded.	All STREAMS-based drivers are loaded automatically from the `(master_disk)>system>kernel_loadable_library` directory. (DLMUX-associated drivers and components of the supporting protocol stack are loaded when the `start_stcp.cm` command macro executes; SDLMUX-associated drivers are loaded with the command `dlmux_admin` *`device_name`* `init_sdlmux.`) Only the non-STREAMS-based lower-level drivers associated with DLMUX (for example, `gdl.pm`, `bdl.pm`, and `dlmux.pm`) must be explicitly loaded from the `(master_disk)>system>kernel_loadable_library` directory. (The DLMUX-associated drivers rely on the STREAMS AD driver to communicate with STCP.)

Table 2-3. Differences between OS TCP/IP and STCP Configuration Procedures *(Page 2 of 3)*

Configuration Procedure	OS TCP/IP	STCP
Establishing a link to the module's default device	Required procedure; `-device` argument required for many commands. To establish such a link, you use the `link` command with a device name. You specify the following value for the *`link_name`* argument of the `link` command: `(master_disk)>system>tcp_os>tcp_device`.	Not applicable. STCP uses the fixed device name format (`#tcp.m1`), which eliminates the need for the symbolic link.
Setting default library paths	The default library path is `(master_disk)>system>tcp_os>`*`library_name`*.	The default library path is `(master_disk)>system>stcp>`*`library_name`*.
Defining the module's host name in the `module_start_up.cm` file	Issue the `sethost` command.	Issue the `hostname` command.
Starting processes at startup	Start the `inetd` process, which manages the `ftpd`, `os_telnet`, `telnet_msd`, `tftpd`, and `bootpd` processes. Start outgoing slave OS TELNET processes from startup.	Start the `ftpd`, `ospfd`, `telnetd`, `stcp_inetd`, and `snmpd` processes from the `start_stcp.cm` command macro, which creates various log files that you can view while the processes are running. If you need BOOTP and TFTP services for boot-file requests, you can have `stcp_inetd` invoke these processes. Note that the `stcp_inetd` process does **not** start the `ftpd` and `telnetd` processes.

Table 2-3. Differences between OS TCP/IP and STCP Configuration Procedures *(Page 3 of 3)*

Configuration Procedure	OS TCP/IP	STCP
Editing database files	Includes various database files.	Includes more database files than OS TCP/IP, such as `snmpconf` and `telnetservice`.
	You edit the `inetd.conf` database file to invoke the daemons for FTP (`ftpd`) and TELNET (`os_telnet` or `telnet_msd`).	The `inetd.conf` database file does not have entries for FTP (`ftpd`) or TELNET (`telnetd`), since these services are not associated with `stcp_inetd`.
	The template database files, whose names have the suffix `.base`, must be copied to the `tcp_os` directory using names without the suffix.	The template database files reside in a separate `templates` directory. You must copy them to the `stcp` directory using the same names.
	You use **either** a Domain Name Service (DNS) or a `hosts` file, not both.	You can use both DNS and a `hosts` file.

NOTE

You can restrict access to STREAMS drivers and modules by specifying an access list for each driver or module. For information about how to set access to STREAMS drivers and modules, see the manual *OpenVOS System Administration: Registration and Security* (R283).

Differences in TELNET Administration

Table 2-4 summarizes the differences in TELNET administration between OS TCP/IP and STCP. Note that you cannot use any of the software components from OS TCP/IP TELNET for STCP TELNET, including the OS TCP/IP servers and the OS TCP/IP access-layer device driver. STCP uses the `telnetd` server and the STREAMS TLI software.

Table 2-4. Differences between OS TCP/IP and STCP TELNET Administration *(Page 1 of 2)*

OS TCP/IP TELNET Administration	STCP TELNET Administration
The client and the following two servers are available with the OS TCP/IP Utilities product: – OS TELNET server (`os_telnet`) – Multisession (MST) server (`telnet_msd`)	The client and the STCP TELNET server are available with the STCP product (all components are in one package). STCP provides two different TELNET servers, `telnetd` and `telnet_msd`.
You can use the `inetd` process to start an OS TELNET or MST server process.	You start the `telnetd` process and the `telnet_msd` process with a `start_process` command from the `start_stcp.cm` command macro, not via the STCP `inetd` process (`stcp_inetd`).
You can configure and start multiple server processes to handle different TELNET services (for example, login versus incoming slave).	You start only one TELNET server process (`telnetd`) and one TELET MSD server process (`telnet_msd`), but you can configure each server process to handle different login and slave services. When the `telnetd` and `telnet_msd` processes run simultaneously, the processes must be listening for requests at different ports.
You define the services with their own servers in the `inetd.conf` and `services` database files.	You define the services managed by the TELNET server in the `telnetservice` and `services` database files before the stack is started or via the `telnet_admin` command after the stack is started. (The `telnet_admin` command updates the `telnetservice` and `services` files. This command can add, delete, modify, or retrieve the status of a login or incoming slave service.)
For device configuration, you must define an entry for the `telnet_al` access-layer device driver and all login and slave devices. Unique device prefixes are associated with each server process, as are network port numbers. For OS TELNET, the `parameters` field of the login or slave device entry must identify the access layer as `telnet_al`.	For device configuration, you must define one entry for the `tli_al` access-layer device driver, two entries for the TELNET pipe devices, and all login and slave devices. You can create one clonable device entry for each service or multiple entries that use a common device prefix. In either case, you must define the clonable device name or common device prefix in the `telnetservice` database file entry for the service. Configuring the `telnet_msd` server for STCP is the same as for OS TCP/IP.

Table 2-4. Differences between OS TCP/IP and STCP TELNET Administration *(Page 2 of 2)*

OS TCP/IP TELNET Administration	STCP TELNET Administration
Prior to VOS Release 14.1.0, you needed to issue `configure_comm_protocol` commands for `streams` (the STREAMS driver) and `telnet_al` (the OS TELNET access-layer device driver) to use the `os_telnet` daemon. In VOS Release 14.1.0 and later, you must issue the `configure_comm_protocol` command only for `telnet_al` to use `os_telnet`.	You do not use `configure_comm_protocol` commands; all of the required STCP TELNET (TLI-related) software components are autoloaded for you.
An application can attach and open multiple slave devices by specifying devices whose device names contain the device prefix associated with the OS TELNET server process (that is, several slave devices could use the same prefix for the server process).	An application can attach and open multiple slave devices by specifying either the clonable device name associated with the service in each call or device names that contain the device prefix associated with the service.

STCP-Related Requests of the `analyze_system` Subsystem

The `analyze_system` subsystem has several STCP-related requests. Four important ones are the following:

- `dump_sdlmux`—This request displays information about the structures associated with an SDLMUX group that has been initialized. Specifically, the request displays the names of the two network interface cards in one (initialized) SDLMUX group.
- `set_stcp_param` and `list_stcp_params`—These requests allow you to list and set various STCP parameters.
- `stcp_meters`—This request displays various STCP meters in order to allow you to analyze STCP performance and activity.

For more information about these requests and about the `analyze_system` subsystem, see the *OpenVOS System Analysis Manual* (R073).

Chapter 3
Converting OS TCP/IP Programs to STCP

When you convert OS TCP/IP programs to run on OpenVOS modules configured with STCP (and not OS TCP/IP), you must first consider various differences in OS TCP/IP and STCP programming related to functions, error handling, and other issues. After you understand these differences, you need to decide which of the following two methods you want to use to convert the programs.

NOTE

This chapter describes converting only OS TCP/IP applications to use STCP. You should write any ported Unix applications and any new applications to be POSIX compliant, and you should enable POSIX socket support. For information on POSIX compliance, see *Installing a Major or Update Release of a Product* (R217) and *OpenVOS POSIX.1 Reference Guide* (R502).

- Binding existing object files (compiled for OS TCP/IP) using the STCP compatibility library—The STCP compatibility library provides STCP objects for OS TCP/IP functions and returns the VOS-style error codes typically used in OS TCP/IP programs. By using the compatibility library, you can convert some programs quickly and easily, though other programs may require more conversion time and result in slower performance, and you might still need to modify the source code.
- Modifying the source code, then compiling and binding for STCP—Stratus recommends this method because using the compatibility library requires more time and, typically, results in slower performance.You can incorporate additional features of STCP if you modify the source code and then recompile and bind using STCP libraries. If the OS TCP/IP application was originally ported from Unix, you can enable POSIX sockets and then revert the code back to the UNIX version.

This chapter presents this information in the following sections:

For complete information about STCP programming, see the *OpenVOS STREAMS TCP/IP Programmer's Guide* (R420).

Programming Issues

Before you convert OS TCP/IP programs to STCP, you must understand various differences between OS TCP/IP and STCP programming in functions, error handling, and other issues. The following sections describe these issues:

Multiple Calls to the `connect` Function

If an OS TCP/IP program contains multiple calls to the `connect` function in order to wait for a connection to complete in non-blocking mode, you must modify the code. OS TCP/IP and STCP do not return corresponding `errno` values with multiple calls to the `connect` function, as Table 3-1 shows.

Table 3-1. The connect Function and errno Values

Calls to the connect Function	OS TCP/IP errno Return Value	STCP errno Return Value
First connect call	e$caller_must_wait (1277)	EINPROGRESS
Subsequent connect calls while not connected	e$action_in_progress (2505)	EALREADY
First connect call after connected	0 code returned; errno unchanged	EISCONN
Subsequent connect calls after connected	e$tcp_socket_connected (4940)	EISCONN

In the OS TCP/IP program, you need to change the following lines (or lines similar to the following):

```
code = connect (...);
if (code == 0)          /* connect now complete */
```

How you change the lines depends on how you choose to convert your program.

- If you choose to modify the source code, and then compile and bind for STCP, replace the lines in the OS TCP/IP program with the following lines for STCP:

  ```
  code = connect (...);
  if (code == -1 && errno == EISCONN)
  ```

- If you choose to bind existing object files (compiled for OS TCP/IP) using the STCP compatibility library, replace the lines in the OS TCP/IP program with the following lines for STCP; in this case, the program will run successfully with OS TCP/IP or STCP:

  ```
  code = connect (...);
  if ((code == 0)||
       (code == -1 && errno == e$tcp_socket_connected))
  ```

  When a connection is completed, STCP returns -1 with errno EISCONN, and OS TCP/IP returns 0 after an initial call to the connect function results in the error EINPROGRESS (with STCP) or e$caller_must_wait (1277) (with OS TCP/IP).

The select Function

When you convert an OS TCP/IP program containing the select function to STCP, you should replace the select function with the poll function under some conditions.

If the program accesses 1024 or more sockets and other file descriptors, and you are building for OpenVOS Standard C (that is, not for POSIX-compliance), do not use the `select` function. You should replace the `select` function with the `poll` function, which provides the same capability as the `select` function with an interface that is more efficient and reliable.

The STCP Response to the `accept` Function

The sequence of requests and responses that STCP uses to acknowledge client SYN requests is different from the sequence used by many other TCP implementations (including OS TCP/IP). In a typical TCP implementation (including OS TCP/IP, and including STCP, when the application is built in POSIX socket mode), the server immediately acknowledges a client's SYN request with a SYN/ACK response for any socket that is listening (that is, a socket for which the server has called the `listen` function). With STCP, when the application is not built in POSIX socket mode, the server acknowledges the client's SYN request with a SYN/ACK response only after the server calls the `accept` function. This sequence, called *lazy accept*, results in the following problems.

- A client that expects a quick response with OS TCP/IP may need to wait longer with STCP, since the response with STCP comes after the server has issued the `accept` function.

  To solve this problem, change the client code to use non-blocking mode. This enables the client to initiate and then poll all required connections, thereby gaining control as each connection is complete. By using non-blocking mode, the client does not need to wait for each server response.

- In STCP, if a server places a socket in non-blocking mode before it issues the `accept` function, the server may gain control after calling `accept` but before the client has completed the three-way handshake (that is, before the client has sent the ACK response to the server's SYN/ACK response). Before VOS Release 14.7.0, the server could gain control before the client completed the three-way handshake even when the socket was in blocking mode.

  In VOS Release 14.7.0, you can solve this problem in one of two ways. The simplest solution is to place the listening socket in blocking mode immediately before it calls `accept`. In blocking mode, `accept` returns only after the three-way handshake is complete and the socket is fully connected. The disadvantage of this solution is that if the three-way handshake cannot be completed, blocking can continue for up to 75 seconds. If you use this solution, you must return the listening socket to non-blocking mode.

  A more complex solution is to leave the listening socket in non-blocking mode. How you implement this solution depends on the logic of the application. If the application calls `select` and then waits for the client to send it something (that is, the application does **not** immediately send the client a message after `accept`

completes), you do not need to make additional changes. The `select` function will not indicate that the accepted socket is ready for reading until the client has sent it a message or the three-way handshake has failed. If, on the other hand, the logic of the applications calls for it to send a message as soon as `accept` completes, the application must be able to handle the `ENOTCONN` error from `send` or the application must have special code that tests the connection before using it.

The `connect` Function and Asynchronous Connections

Beginning in VOS Release 14.6.1, you can establish asynchronous connections with the STCP `connect` function. Previously, only OS TCP/IP provided this functionality.

In addition, processes initiating STCP connections can be interrupted at any time during the connection, whether the socket is in default blocking mode or in non-blocking mode. Previously, once a connection was initiated, the process could not be interrupted until the connect completed or until a time-out occurred (up to four minutes later). (See the *OpenVOS STREAMS TCP/IP Programmer's Guide* (R420) for additional information on using the `connect` function.)

The `recv` and `recvfrom` Functions and Datagram Size

OS TCP/IP and STCP treat differently the `recv` and `recvfrom` functions on a socket of the type `SOCK_DGRAM` when the byte length of the datagram exceeds the length requested on the `recv` or `recvfrom` call. OS TCP/IP sends the remaining bytes of the datagram in the next call to receive another datagram. STCP discards the remaining bytes, as required by POSIX, in order to maintain a one-to-one relationship between datagrams sent and received.

Keepalive Functionality

Both OS TCP/IP and STCP provide keepalive functionality that tests whether the TCP stack is functioning; however, keepalive functionality does not confirm that a program is running correctly. You can use keepalive functionality to allow a program to clear connections that have failed (for example, when a peer crashes), and to free the socket for another use.

NOTES

1. Keepalive functionality with STCP (or OS TCP/IP) is not intended to continually monitor a connection. It is not a program heartbeat test for connectivity because the time interval for sending the first probe packet (default value is two hours) is too long. Reducing the time interval to something more useful (for example, five minutes) may cause connections to break during

transient internet failures, or it may result in other problems with network load or performance.

2. RFC 1122 recommends that the time interval for sending the first keepalive probe packet be two hours.

The behavior of keepalive functionality and how programmers and administrators control it is different with STCP than it is with OS TCP/IP. For example, with STCP, administrators can set the default timing of the keepalive probes for all sockets. If you need to convert an OS TCP/IP program that contains the SO_KEEPALIVE option of the `getsockopt` and `setsockopt` functions, you must read the following sections:

- ‘‘Parameters of the SO_KEEPALIVE Socket Option’’ on page 3-6
- ‘‘Modifying a Program That Contains the SO_KEEPALIVE Option’’ on page 3-7
- ‘‘Keepalive Functionality on an STCP Interface’’ on page 3-8

Parameters of the SO_KEEPALIVE Socket Option

With STCP, various parameters control the behavior of the SO_KEEPALIVE socket option. Before running a program that creates sockets, an administrator can set values for these parameters. STCP assigns these values when a program creates a socket; it does not assign new values to sockets that are already allocated.

An administrator can list and set values of these parameters by using requests of the `analyze_system` command: the `list_stcp_params` request lists values and the `set_stcp_param` request sets a value. Table 3-2 lists and describes the parameters.

Table 3-2. Keepalive Parameters

Parameter	Description
`keepalive_time`	Sets the time interval, in minutes, when STCP sends the first probe packet after the last packet is received from the remote peer. If, for example, `keepalive_time` is set to the value `150`, STCP sends the first probe packet 2.5 hours (150 minutes) after receiving the last packet from the remote peer. The range of values is `1` to `480`; the default value is `120` minutes.
`keepalive_tries`	Sets the number of probe packets that SO_KEEPALIVE sends. The range of values is `1` to `25`; the default value is `9`.
`check_if_dead`	Sets the time interval, in seconds, between the second and subsequent probe packets. The time interval between the first and second probe packets is eight seconds. The range of values is `30` to `360`; the default value is `75` seconds.

For example, the following `analyze_system` request enables STCP to send a total of five probe packets, where the first probe packet is sent 60 minutes after the last packet

is received from the remote peer, the second probe packet is sent after eight seconds (a constant interval), and subsequent probe packets are sent at two-minute intervals.

```
as: set_stcp_param keepalive_time 60
Changing keepalive time interval (keepalive_time)
      from 120 min to 60 min

as: set_stcp_param keepalive_tries 5
Changing keepalive tries (keepalive_tries)
      from 9 to 5

as: set_stcp_param check_if_dead 120
Changing keepalive check dead time (check_if_dead)
      from 75 sec to 120 sec
```

The program must also include a call to the `setsockopt` function to set the SO_KEEPALIVE option, and keepalive must be enabled on the STCP interface (it is by default—see ‘‘Keepalive Functionality on an STCP Interface’’ on page 3-8).

Modifying a Program That Contains the SO_KEEPALIVE Option

With OS TCP/IP, you can use the SO_KEEPALIVE option of the `setsockopt` function to turn on or off keepalive probes for a specific socket by using an interface for which this functionality is enabled. You do not need to modify an OS TCP/IP program that uses this feature when you convert the program to STCP. STCP sends the first probe packet at the default time interval of two hours (after the last packet is received from its peer). However, with STCP, programmers can control the timing of keepalive probes for a specific socket. To use the STCP functionality in an OS TCP/IP program, you must modify the program.

Replace the `int` variable whose value is 0 or 1 with `struct linger keepalive`. The SO_KEEPALIVE and SO_LINGER options both use the `linger` structure, which allows you to specify one `int` variable to indicate on-off and a second `int` variable to indicate a time value.

If the value of the on-off `int` variable is non-zero, then the value of the time `int` variable defines the number of seconds that STCP waits before it sends the first probe for this socket (after the last packet is received from its peer). If you want the socket to use a non-default value for the timing of the first probe (for example, 1 hour), the program must pass a `linger` structure and set the `keepalive.1_linger` field to `3600`. Similarly, if the program passes a structure rather than an `int` variable to the `getsockopt` function for the SO_KEEPALIVE option, then the keepalive time set for this socket will be returned; otherwise, only the on-off state is returned in the `int`.

For example, the following lines (or lines similar to the following) of the program enable keepalive with default values:

```
int on = 1;
int setsockopt(s, SOL_SOCKET, SO_KEEPALIVE, (char *)&on,
      sizeof(on));
```

To enable keepalive for 60 minutes, use the following lines:

```
struct linger keepalive ;
keepalive.1_onoff = 1;
keepalive.1_linger = 3600; /* non-default value of 1 hour */
      setsockopt(s, SOL_SOCKET, SO_KEEPALIVE, (char *)
      &keepalive, sizeof(keepalive));
```

Keepalive Functionality on an STCP Interface

Keepalive functionality is enabled on the STCP interface by default when an administrator adds an interface using the `ifconfig` command. If an administrator has added an interface using the `ifconfig` command with the `-no_kalive` argument, the SO_KEEPALIVE option has no effect.

To determine if keepalive is enabled, issue the `ifconfig` command. If it is enabled, KEEPALIVE appears in the command output, as in the following example.

```
ifconfig #sdlmux2

%s#sdlmux2: <UP, BROADCAST, RUNNING, NOFORWARDBROADCAST,
KEEPALIVE> 164.152.77.6 netmask 0xfffffe00 broadcast
164.152.77.255
```

If KEEPALIVE does not appear, you cannot use the keepalive functionality on this STCP interface.

Spurious Notifies

OS TCP/IP programs typically use the `get_socket_event` function for asynchronous events in order to associate an event ID with a socket, and then the program uses the `s$wait_event` subroutine to wait for activity on that socket. POSIX uses the `poll` and `select` functions for this functionality, though STCP also supports the `get_socket_event` function. So, you can rewrite OS TCP/IP programs to use the POSIX functions, or you can continue to use the `s$wait_event` subroutine. If you continue to use `s$wait_event`, you must confirm that your program deals properly with spurious notifies.

A *spurious notify* occurs when control is returned to a program even though the event that the program is waiting for has not occurred. You should always code for the

possibility of a spurious notify, and then the program can continue to wait using the `s$wait_event` subroutine. A spurious notify can occur in OS TCP/IP as well as in STCP, though a spurious notify with OS TCP/IP is extremely rare. Improperly coded OS TCP/IP programs may be able to run successfully without the problem being noticed. Because of the STREAMS environment of STCP, spurious notifies are more common and, thus, converted programs may fail when they had not failed in OS TCP/IP.

To check for the possibility of spurious notifies, write the program to confirm that the event it is waiting for has actually occurred (that is, allow for an error on the subsequent call), and then to return to call the `s$wait_event` subroutine if the event has not occurred.

OS TCP/IP Load Balancing

OS TCP/IP is unique in providing load-balancing functionality in programming. Typical TCP implementations (including STCP) do not provide this functionality. With STCP, if two programs bind to the same port, the first application that binds to the port receives the packets arriving on both interfaces, while the second program never receives any packets. If your OS TCP/IP program depends on load balancing occurring, you must change the program.

Error Handling

STCP uses the error files `error_codes.incl.c` and `errno.h` (in the `(master_disk)>system>include_library` directory) for functions that perform error handling. The `errno.h` file defines the VOS values of the POSIX error codes that STCP uses.

OS TCP/IP uses the OS TCP/IP-specific error file `tcp_errno.h` (in the `(master_disk)>system>stcp>include_library>compat` directory) for error handling. If an OS TCP/IP program references POSIX error codes by using the `tcp_errno.h` file to map the error codes, simply replace `tcp_errno.h` with `errno.h`. If you are writing a new program, you should include `errno.h`.

If, however, an OS TCP/IP program uses VOS-style error codes explicitly, then you can use the `map_stcp_error` function, which maps the POSIX error codes that STCP returns to the VOS-style error codes that your program is checking. For example, the `map_stcp_error` function maps the POSIX error code `EWOULDBLOCK` (`6054`) to the corresponding VOS-style error code returned by OS TCP, `e$caller_must_wait` (`1277`).

See the *OpenVOS STREAMS TCP/IP Programmer's Guide* (R420) for more information about the `map_stcp_error` function.

Unsupported OS TCP/IP Socket Options

You must remove the SO_DELSEND OS TCP/IP socket option from your program because the STCP compatibility and standard libraries do not support it.

IP-Level Options

STCP supports the following IP-level options, which OS TCP/IP does not support. The *OpenVOS STREAMS TCP/IP Programmer's Guide* (R420) describes these IP-level options.

```
IP_ADD_MEMBERSHIP
IP_DROP_MEMBERSHIP
IP_MULTICAST_IF
IP_MULTICAST_LOOP
IP_MULTICAST_TTL
```

Using the STCP Compatibility Library

Before you choose the method of using the STCP compatibility library to convert OS TCP/IP programs to STCP, consider the following issues:

- The compatibility library does not provide support for the transfer of control between processes.
- You cannot use the `net_ioctl` function to administer the TCP stack. Some applications use `ioctl` functions to administer OS TCP/IP, though this capability is not documented. The compatibility library supports only documented uses of `net_ioctl`.
- STCP does not support the OS TCP/IP functions `set_device` or `get_device`. If an OS TCP/IP program uses these functions to specify a TCP device on another module in the network, you must rewrite the program. However, if an OS TCP/IP program uses these functions to specify a local TCP device, you can use the compatibility library with the program if you set the `device_flag` variable to `1`. (See "Converting a Program That Contains the `get_device` or `set_device` Function" on page 3-12 for more information.)
- The STCP compatibility library does not define the OS TCP/IP `faccept` function. If an OS TCP/IP program uses this function, replace it with a call to the `accept` function.
- The STCP compatibility library does not define the OS TCP/IP `BOOLEAN` constant. To use this constant in STCP programs, add the following line to your header files.

  ```
  #define BOOLEAN unsigned int /* {TRUE, FALSE} */
  ```

- Functions handled by the compatibility library typically return the VOS-style error codes that the OS TCP/IP library uses. However, a function may occasionally

return a POSIX error code, and a program needs to be able to process those codes. (See “Error Handling and the Compatibility Library” on page 3-13 for more information.)

To convert an OS TCP/IP program to STCP by using the compatibility library (`complib`), you must bind the program with the compatibility library (see “Binding a Program with the Compatibility Library” on page 3-11), and then test and troubleshoot it (see “Testing the Converted Program” on page 3-12).

Binding a Program with the Compatibility Library

To bind your OS TCP/IP program with the compatibility library, perform the following steps:

1. Add the compatibility library to the object-library search paths (see “Adding the Compatibility Library to the Library Search Paths” on page 3-11).
2. Bind the program.
3. If your program uses the `set_device` or `get_device` function, you need to modify your program or set the `device_flag` variable to `1` (see “Converting a Program That Contains the `get_device` or `set_device` Function” on page 3-12).
4. After you bind the program, test it to ensure that it performs correctly, and troubleshoot it, if necessary (see “Testing the Converted Program” on page 3-12).

Adding the Compatibility Library to the Library Search Paths

You must add the compatibility library and the STCP object library to the object-library search paths of the process that you use to bind your OS TCP/IP program.

To add the compatibility library, issue the following command for your process, before binding your OS TCP/IP program:

```
add_library_path object >system>stcp>object_library>complib
   -before >system>c_object_library
```

NOTE

Do not add the compatibility library to the object-library search paths of a process whose object-library search paths include the POSIX runtime library, which is the `(master_disk)>system>posix_object_library` directory.

The `module_start_up.cm` file should not include command lines to add the compatibility library to the default object-library search paths. Many applications may specifically not require the compatibility library.

To avoid search-path ambiguities, delete OS TCP/IP object libraries from the object-library search paths by using the `delete_library_path` command.

For information about the `add_library_path` and `delete_library_path` commands, see the *OpenVOS Commands Reference Manual* (R098).

Converting a Program That Contains the `get_device` or `set_device` Function

STCP does not support the OS TCP/IP functions `set_device` or `get_device`. In OS TCP/IP, the `get_device` and `set_device` functions allow you to specify a TCP device on another module in your network. In STCP, you cannot access a TCP device on another module in this way, so you must rewrite such OS TCP/IP programs. If you use the STCP compatibility library with such an OS TCP/IP program, you receive the error `e$operation_not_supported` (3984).

You can use the compatibility library with an OS TCP/IP program that contains the `get_device` or `set_device` function to specify a TCP device on the **local** module (that is, the TCP device is on the module that is running the OS TCP/IP program) if you set the `device_flag` variable in your program module to `1`. To do so, issue the following command (where *`pm_name`* is the name of your program module):

```
set_external_variable device_flag -in pm_name -to 1 -type integer
```

If you set the `device_flag` variable to `1` and bind your program with the compatibility library, the `set_device` function returns `0` or an error if the specified device is not a local TCP device, and the `get_device` function returns `0` and the name of the local TCP device.

Testing the Converted Program

After binding a program, you must functionally test it to check that the program runs correctly. If it fails the tests, perform the following steps to resolve the problem:

1. Check that your include-library paths do **not** contain STCP libraries but do contain the following:

   ```
   >system>tcp_os>include_library
   >system>include_library
   ```

2. Compile the OS TCP/IP program, as described in the *VOS Communications Software: OS TCP/IP Programmer's Manual* (R224).
3. Check that the object-library paths are correct (see "Adding the Compatibility Library to the Library Search Paths" on page 3-11). If these paths are not correct, correct them and rebind the program.

If the program still fails functional tests, examine the program code for the following possible problems.

- The `net_ioctl` function supports only the `FIONBIO`, `FIONREAD`, or `SIOCATMARK` request. Remove any other uses of `net_ioctl`. In addition, you cannot use the `net_ioctl` function to administer the STCP stack. (See ‘‘Using the STCP Compatibility Library’’ on page 3-10.)
- If the program uses the OS TCP/IP functions `get_device` or `set_device` to specify a TCP device on another module in the network, or if the program uses these functions to specify a local TCP device but the `device_flag` variable is not set to `1` in the program module, these functions cause errors (see ‘‘Converting a Program That Contains the `get_device` or `set_device` Function’’ on page 3-12).
- A function may occasionally return a POSIX-defined `errno` value, and the program needs to be able to process those codes (see ‘‘Error Handling and the Compatibility Library’’ on page 3-13).
- The objects for some functions provide functionality that is different from the OS TCP/IP implementation. For information about these differences, see ‘‘OS TCP/IP Functions and the Compatibility Library’’ on page 3-13.

Resolving these issues typically requires rewriting sections of the program code. When you have rewritten the code, compile and bind the program (see ‘‘Binding a Program with the Compatibility Library’’ on page 3-11).

Error Handling and the Compatibility Library

If an OS TCP/IP program is coded to check for specific `errno` values, you may need to modify the program. In most cases, the compatibility library returns VOS-style error codes; however, a function may occasionally return a POSIX error code, and a program needs to be able to process those codes. You can rewrite the program to include, for example, a common error routine or macro that checks error codes.

OS TCP/IP Functions and the Compatibility Library

The compatibility library provides limited support for some OS TCP/IP functions. If your code contains certain OS TCP/IP functions and you bind a program with the compatibility library, the object functions simply return the error code `e$operation_not_supported` (3984). (If you compile such an OS TCP/IP program with the STCP standard libraries, the program produces compiler errors.) Objects in the compatibility library for some other functions have limitations that are not present in the OS TCP/IP function objects; Table 3-3 describes these limitations.

Table 3-3. Support Limitations of OS TCP/IP Functions in the Compatibility Library

Function	Limitation
`bind`	You can use the `bind` function only with the address of an interface on the current module.
`get_device`	When the `device_flag` variable is set to `1` on the program module, this function returns `0` and the name of the local TCP device. Otherwise, this function returns an error, and you should remove it and rewrite your program to use only local TCP devices.
`getsockbase`	Returns the `BASE_SOCKET_ID` constant. This function is not meaningful to STCP, but the compatibility library always returns the `BASE_SOCKET_ID` constant to ensure portability of programs that call the function.
`getsockopt`	Supports a limited set of options (see “Unsupported OS TCP/IP Socket Options” on page 3-10).
`net_ioctl`	You cannot use the `net_ioctl` function to administer the TCP stack. Some applications use `ioctl` functions to administer OS TCP/IP even though this capability is not documented. The compatibility library supports only documented uses of `net_ioctl`.
`set_device`	If the function call specifies a local TCP device, and if the `device_flag` variable is set to `1` on the program module, this function returns `0`. Otherwise, this function returns an error, and you should remove it and rewrite your program to use only local TCP devices.
`setsockopt`	Supports a limited set of options (see “Unsupported OS TCP/IP Socket Options” on page 3-10).

Converting Source Code to STCP

You may prefer to convert the OS TCP/IP source code to run natively on STCP rather than binding existing object files (compiled for OS TCP/IP) using the STCP compatibility library. When you modify the source code, you can incorporate additional features of STCP, which you can use with either the OpenVOS Standard C runtime product or with the POSIX-compliant runtime product. So, for example, if you convert the source code, you can create a program that is POSIX-compliant.

The following sections provide information about converting source code to STCP:

- “Creating Source Code That Is POSIX and ANSI-C Compliant” on page 3-15
- “Basic Steps to Convert Source Code to STCP” on page 3-16

Creating Source Code That Is POSIX and ANSI-C Compliant

The STCP programming interface complies with the POSIX and ANSI-C standards. POSIX refers to the IEEE POSIX standard, which is a system application program interface (API). POSIX support enables OpenVOS programmers to port applications that conform to the POSIX standard, with minimal source-code modification.

When you convert the source code of OS TCP/IP programs to STCP, you need to decide if you want to create POSIX-compliant source code. You can import most POSIX-compliant TCP/IP applications directly to OpenVOS with no modifications and run them correctly using STCP and the OpenVOS POSIX environment. Note, however, that POSIX socket support is an addition to POSIX support, and you must enable POSIX socket support by using POSIX socket mode. (For information on enabling POSIX socket support, see the documentation listed at the end of this section.)

The OpenVOS Standard C compiler, which you invoke using the `cc` command, allows you to create programs that are ANSI-C- and/or POSIX-compliant. Stratus recommends that you use the `cc` command for the OpenVOS Standard C compiler rather than the `c` command for the older OpenVOS C compiler. For information about the `cc` command, see the *OpenVOS Standard C User's Guide* (R364).

The OpenVOS Standard C compiler may generate numerous compiler warnings the first time you use it to compile a TCP/IP program if you had previously used the OpenVOS C compiler to compile the program. A common cause of warnings is the absence of function prototypes. You may need to include additional header files (such as `string.h`, `stdio.h`, and `stdlib.h`) that contain the prototypes for many standard C language functions in order to remove these warnings. For example, if your program uses the `strcpy` function, the header file definition list must include the header file `string.h`. For information about the OpenVOS Standard C header files and the functions that require them, see the *OpenVOS Standard C Reference Manual* (R363).

You can eliminate other compiler compatibility problems by using one or all of the compatibility arguments of the `cc` command: `-compatible_bitfields`, `-compatible_search`, and `-compatible_generics`. For information about these arguments, see the *OpenVOS Standard C User's Guide* (R364).

To create POSIX-compliant source code for OpenVOS, see the following additional documentation:

- *OpenVOS POSIX.1: Conformance Guide* (R217M), which describes how the OpenVOS POSIX implementation adheres to or deviates from the POSIX standard. This document is available on the StrataDOC (OpenVOS Version) DVD and on the OpenVOS StrataDOC Web site.
- *OpenVOS POSIX.1 Reference Guide* (R502), which documents OpenVOS POSIX features, including POSIX socket mode.

Basic Steps to Convert Source Code to STCP

To convert source code to STCP, use the STCP API, which includes the STCP standard libraries. Before you write, compile, and bind your program to conform to the STCP API, see the *OpenVOS STREAMS TCP/IP Programmer's Guide* (R420).

The basic steps to convert source code to STCP are as follows:

1. Modify the program to conform to the STCP API (as described in the *OpenVOS STREAMS TCP/IP Programmer's Guide* (R420)). As you modify your program, you should be familiar with the following information:
   - differences in error-code mappings (see "Error Handling" on page 3-9 and Appendix A)
   - new functions that STCP supports as well as supported and unsupported OS TCP/IP functions ("Using Functions in STCP" on page 3-16)
   - the program must include the STCP header files (an STCP API requirement—see "Including Header Files" on page 3-19)
2. Compile the program (see the *OpenVOS STREAMS TCP/IP Programmer's Guide* (R420)).
3. Bind the program with the STCP object library (see the *OpenVOS STREAMS TCP/IP Programmer's Guide* (R420)).

Using Functions in STCP

The following sections describe differences between STCP and OS TCP/IP functions.

- "New Functions" on page 3-16 lists functions that are in the STCP, OpenVOS Standard C, and OpenVOS STREAMS libraries but are not in the OS TCP/IP library.
- "Supported OS TCP/IP Functions" on page 3-17 lists functions in the STCP, OpenVOS Standard C, or OpenVOS STREAMS libraries as well as in the OS TCP/IP library.
- "Unsupported OS TCP/IP Functions" on page 3-19 lists functions that are not in the STCP and OpenVOS Standard C libraries but are in the OS TCP/IP library.
- "Function Prototype" on page 3-19 describes the `socket.h` and `netdb.h` files.

New Functions

The STCP, OpenVOS Standard C, and OpenVOS STREAMS libraries support several functions that OS TCP/IP does not support. Table 3-4 lists these functions and the library that provides the function.

Table 3-4. Functions for STCP Only

Function	Library
`accept_on`	STCP
`close`	OpenVOS Standard C†
`fcntl`	OpenVOS Standard C†
`ioctl`	OpenVOS STREAMS
`map_stcp_error`	STCP
`read`	OpenVOS Standard C†
`so_recv`	STCP
`stcp_spawn_child_process`	STCP
`write`	OpenVOS Standard C†

† In OS TCP/IP, you can use these functions only for file handling. In STCP, you can use these functions for both file and socket handling.

For descriptions of the STCP functions, see the *OpenVOS STREAMS TCP/IP Programmer's Guide* (R420).

For descriptions of the OpenVOS Standard C functions, see the *OpenVOS Standard C Reference Manual* (R363).

For descriptions of the OpenVOS STREAMS functions, see the *OpenVOS Communications Software: STREAMS Programmer's Guide* (R306).

Supported OS TCP/IP Functions

The STCP standard library provides many of the same or similar functions as OS TCP/IP. When the STCP function has the same name as the OS TCP/IP function, you can often include the appropriate header files and recompile the program. When the STCP function has a different name, you need to replace the OS TCP/IP function with the corresponding STCP function, include the appropriate header files, and recompile the program.

Table 3-5 lists OS TCP/IP functions that the STCP standard library supports, and describes differences between the STCP and OS TCP/IP implementations.

Table 3-5. STCP Support for OS TCP/IP Functions *(Page 1 of 2)*

Function	Description of STCP Support
`bind`	Supported by the OpenVOS Standard C and POSIX libraries.
`getpeername`	Supported by the OpenVOS Standard C and POSIX libraries.
`getsockname`	Supported by the OpenVOS Standard C and POSIX libraries.
`getsockopt`	Supported by the OpenVOS Standard C and POSIX libraries. However, the STCP function supports a limited set of options, as described in ‘‘Unsupported OS TCP/IP Socket Options’’ on page 3-10.
`get_socket_event`	Supported by the STCP standard library for compatibility with OS TCP/IP.
`inet_addr`	Supported by the STCP standard library. However, the STCP `inet_addr` function returns `-1` if unsuccessful. (The OS TCP/IP function returns `INADDR_NONE`.)
`listen`	Supported by the OpenVOS Standard C and POSIX libraries.
`net_close`	Replace with the STCP standard library `close` function.
`net_ioctl`	Replace with the OpenVOS STREAMS library `fcntl` function.
`net_read`	Replace with the OpenVOS Standard C library `read` function.
`net_write`	Replace with the OpenVOS Standard C library `write` function.
`receive_socket`	Supported by the OpenVOS Standard C and POSIX libraries.
`recv`	Supported by the OpenVOS Standard C and POSIX libraries.
`recvfrom`	Supported by the OpenVOS Standard C and POSIX libraries.
`recvmsg`	Supported by the OpenVOS Standard C and POSIX libraries.
`select`	Supported by the OpenVOS Standard C and POSIX libraries.
`select_with_events`	Supported by the OpenVOS Standard C and POSIX libraries.
`send`	Supported by the OpenVOS Standard C and POSIX libraries.
`sendmsg`	Supported by the OpenVOS Standard C and POSIX libraries.
`sendto`	Supported by the OpenVOS Standard C and POSIX libraries.
`sethostname`	Supported by the STCP standard library.
`setsockopt`	Supported by the OpenVOS Standard C and POSIX libraries.
`shutdown`	Supported by the OpenVOS Standard C and POSIX libraries.

Table 3-5. STCP Support for OS TCP/IP Functions *(Page 2 of 2)*

Function	Description of STCP Support
`socket`	Supported by the OpenVOS Standard C and POSIX libraries.
`transfer_socket`	Supported by the OpenVOS Standard C and POSIX libraries.
`writev`	Supported by the STCP standard library.

Unsupported OS TCP/IP Functions

The STCP standard library does not support the following OS TCP/IP functions.

```
get_device
gethostid
getsockbase
readv
set_device
sethostid
```

If you compile an OS TCP/IP program that contains these functions with the standard libraries, the program produces compiler errors. To ensure that the program compiles properly, remove or replace the unsupported functions and recompile the program.

Function Prototype

The `socket.h` file, in the `(master_disk)>system>include_library>sys` directory, contains prototypes for socket-handling functions. The `netdb.h` file, in the `(master_disk)>system>include_library` directory, contains prototypes for functions that map host, network, protocol, and service names to numeric values.

Including Header Files

STCP requires the header files that reside in the directory `(master_disk)>system>include_library` and its subdirectories. (OS TCP/IP requires the header files in the directory `(master_disk)>system>tcp_os>include_library`.) For descriptions of the STCP header files, see the *OpenVOS STREAMS TCP/IP Programmer's Guide* (R420).

Be sure to include the header file `sys/types.h` in your STCP program because this file defines the data types used in other header files.

Since the `(master_disk)>system>include_library` directory is in the default include library search paths, you do not need to take any action to reference the STCP header files in your source program.

To avoid search-path ambiguities, delete existing OS TCP/IP include-library paths. The `delete_library_path` command allows you to delete a single library path from the list of directories that define the specified library.

See the *OpenVOS Commands Reference Manual* (R098) for more information about the `delete_library_path` command.

Chapter 4
Differences in the FTP User Interface

This chapter discusses the differences between the FTP user interfaces of OS TCP/IP and STCP. Table 4-1 describes these differences.

Table 4-1. Differences between the FTP User Interfaces of OS TCP/IP and STCP

OS TCP/IP User Interface	STCP User Interface
The explanation of CR-stripping mode (in FTP) in the manual *VOS Communications Software: OS TCP/IP User's Guide* (R222) is incorrect.	The explanation of CR-stripping mode (in FTP) has been corrected in the *VOS STREAMS TCP/IP User's Guide* (R421).
The OS TCP/IP version of FTP supports the `account`, `device`, and `reset` subcommands as described in the *VOS Communications Software: OS TCP/IP User's Guide* (R222).	STCP does not support the `account`, `device`, and `reset` subcommands. In addition, `account` and `device` are not listed in the FTP `help` subcommand output.
The OS TCP/IP version of FTP does not support the `literal` and `sequential` subcommands. (Note, however, that OS TCP/IP does support the `quote` subcommand, which performs the same function as the `literal` subcommand. Also, the default mode for ASCII is sequential.)	STCP supports the `literal` and `sequential` subcommands.
The `trace` subcommand is listed in the FTP `help` subcommand but is unimplemented.	The `trace` subcommand is not listed in the FTP `help` command and is unimplemented.

For more information about the STCP user interface of FTP and other commands, see the *VOS STREAMS TCP/IP User's Guide* (R421).

Appendix A
Error Code Mappings

The STCP compatibility library converts STCP error codes to OS TCP/IP error codes, and returns the OS TCP/IP error codes to an application. Table A-1 shows the mapping of STCP error codes to OS TCP/IP error codes.

Table A-1. Error Code Mapping *(Page 1 of 3)*

STCP Error Code	OS TCP/IP Error Code
EACCESS	e$insufficient_access
EADDRINUSE	e$tcp_address_in_use
EADDRNOTAVAIL	e$tcp_address_unavailable
EAFNOSUPPORT	e$tcp_invalid_address_family
EAGAIN	e$caller_must_wait
EALREADY	e$action_in_progress
EBADF	e$invalid_socket
EBADMSG	e$invalid_message
EBUSY	e$device_in_use
ECONNABORTED	e$tcp_connection_aborted
ECONNREFUSED	e$tcp_connection_refused
ECONNRESET	e$tcp_connection_reset
EDESTADDRREQ	e$tcp_destination_required
EEXIST	e$tcp_entry_already_exists
EFAULT	e$invalid_arg
EFBIG	e$max_file_exceeded
EHOSTDOWN	e$tcp_host_down
EHOSTUNREACH	e$tcp_host_unreachable

Table A-1. Error Code Mapping *(Page 2 of 3)*

STCP Error Code	OS TCP/IP Error Code
EINPROGRESS	e$tcp_op_in_progress
EINTR	e$tcp_interrupted_operation
EINVAL	e$invalid_arg
EIO	e$internal_error
EISCONN	e$tcp_socket_connected
EMFILE	e$no_more_sockets
EMSGSIZE	e$tcp_message_too_long
ENETDOWN	e$tcp_network_is_down
ENETRESET	e$tcp_network_reset
ENETUNREACH	e$tcp_network_unreachable
ENFILE	e$tcp_ft_overflow
ENOBUFS	e$tcp_no_buffer_space
ENODEV	e$not_a_device
ENOMEM	e$no_alloc_user_heap
ENOPROTOOPT	e$tcp_protocol_unavailable
ENOSPC	e$tcp_no_space
ENOSR	e$tr_no_resrc
ENOSTR	e$invalid_socket
ENOTCONN	e$tcp_socket_not_connected
ENOTSOCK	e$tcp_non_socket
ENOTTY	e$invalid_socket
ENXIO	e$no_more_sockets
EOPNOTSUPP	e$tcp_invalid_operation
EPERM	e$insufficient_access
EPFNOSUPPORT	e$tcp_invalid_protocol_family
EPROTONOSUPPORT	e$tcp_protocol_not_supported
EPROTOTYPE	e$tcp_wrong_protocol_type

Table A-1. Error Code Mapping *(Page 3 of 3)*

STCP Error Code	OS TCP/IP Error Code
ERANGE	e$tcp_message_too_long
ESHUTDOWN	e$tcp_shutdown
ESOCKTNOSUPPORT	e$tcp_invalid_socket_type
ESPIPE	e$tcp_broken_pipe
ESRCH	e$tcp_entry_not_found
ETIMEOUT	e$timeout
ETOOMANYREFS	e$tcp_too_many_refs
EWOULDBLOCK	e$caller_must_wait

Index

D

E

F

P

R

S

T

U

V

W