# VOS Administrator's Guide for Open StrataLINK

Stratus Technologies
R388-06

# 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.





Manual Name: *VOS Administrator's Guide for Open StrataLINK*

Part Number: R388
Revision Number: 06
VOS Release Number: 16.1.0
Publication Date: December 2006

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

# Contents

# Figures

# Tables

# Preface

The *VOS Administrator's Guide for Open StrataLINK* (R388) documents how to configure and manage Open StrataLINK (OSL) for VOS Release 16.1.0 (and later releases). Open StrataLINK is a networking product that establishes cross-module and cross-system communications using STREAMS TCP/IP (STCP).

This manual is intended for system or network administrators. It may also be useful for systems programmers who are designing distributed applications or networking software with specific configuration and/or administration requirements.

Before using the *VOS Administrator's Guide for Open StrataLINK* (R388), you should be familiar with the following manuals.

- *VOS STREAMS TCP/IP Administrator's Guide* (R419)
- *VOS Commands User's Guide* (R089)
- *VOS Commands Reference Manual* (R098)
- *VOS System Administration: Configuring a System* (R287)

You must follow the configuration procedures for STCP as described in the *VOS STREAMS TCP/IP Administrator's Guide* (R419).

## 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.

This revision introduces the following new sections.

- "OSL Configurations and Fault Tolerance" on page 1-7
- "Configuring Routes in STCP" on page 1-9
- "The OSL Administrative Tool `osl_admin`" on page 1-11

This revision newly documents several `osl_admin` requests:

- "The `adjust_saved_trace` Request" on page 6-28
- "The `display_trace` Request" on page 6-34
- "The `list_saved_traces` Request" on page 6-41
- "The `match` Request" on page 6-42

- "The `merge_trace_buffers` Request" on page 6-44
- "The `sleep` Request" on page 6-66

This revision also newly documents the `osl_daemon` and `osl_overseer` commands.

This revision incorporates information about OSL and SDLMUX groups (for example, Figure 1-5, Figure 1-7, "The Sample System `%admin`" on page 3-4, and Figure 4-1, as well as other sections).

Explanations of various procedures have been updated. Examples of configurations with proprietary StrataLINK have been removed.

## Manual Organization

This manual contains the following chapters.

Chapter 1 describes the design and function of Open StrataLINK with STCP and the configurations supported by Open StrataLINK.

Chapter 2 presents a quick configuration checklist for you to use when you are configuring a single-system or multiple-system Open StrataLINK network.

Chapter 3 describes how to configure modules for single-system Open StrataLINK communications.

Chapter 4 describes how to configure modules for multiple-system Open StrataLINK communications.

Chapter 5 explains how to handle common configuration errors.

Chapter 6 describes the administrative commands that enable you to administer Open StrataLINK.

## Related Manuals

Refer to the following Stratus manuals for related documentation.

- *VOS Commands User's Guide* (R089)
- *VOS Commands Reference Manual* (R098)
- VOS System Administration manuals:

  *VOS System Administration: Administering and Customizing a System* (R281)

  *VOS System Administration: Starting Up and Shutting Down a Module or System* (R282)

  *VOS System Administration: Registration and Security* (R283)

*VOS System Administration: Disk and Tape Administration* (R284)

*VOS System Administration: Backing Up and Restoring Data* (R285)

*VOS System Administration: Administering the Spooler Facility* (R286)

*VOS System Administration: Configuring a System* (R287)

- hardware manuals:

  *Stratus ftServer V 250, V 300, V 500, and V 502 Systems: Site Planning Guide* (R605)

  *Stratus ftServer V 250, V 300, V 500, and V 502 Systems: Operation and Maintenance Guide* (R606)

  *Stratus ftServer: Network I/O Enclosure Guide* (R608)

- *VOS STREAMS TCP/IP Migration Guide* (R418)
- *VOS STREAMS TCP/IP Administrator's Guide* (R419)

## 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.

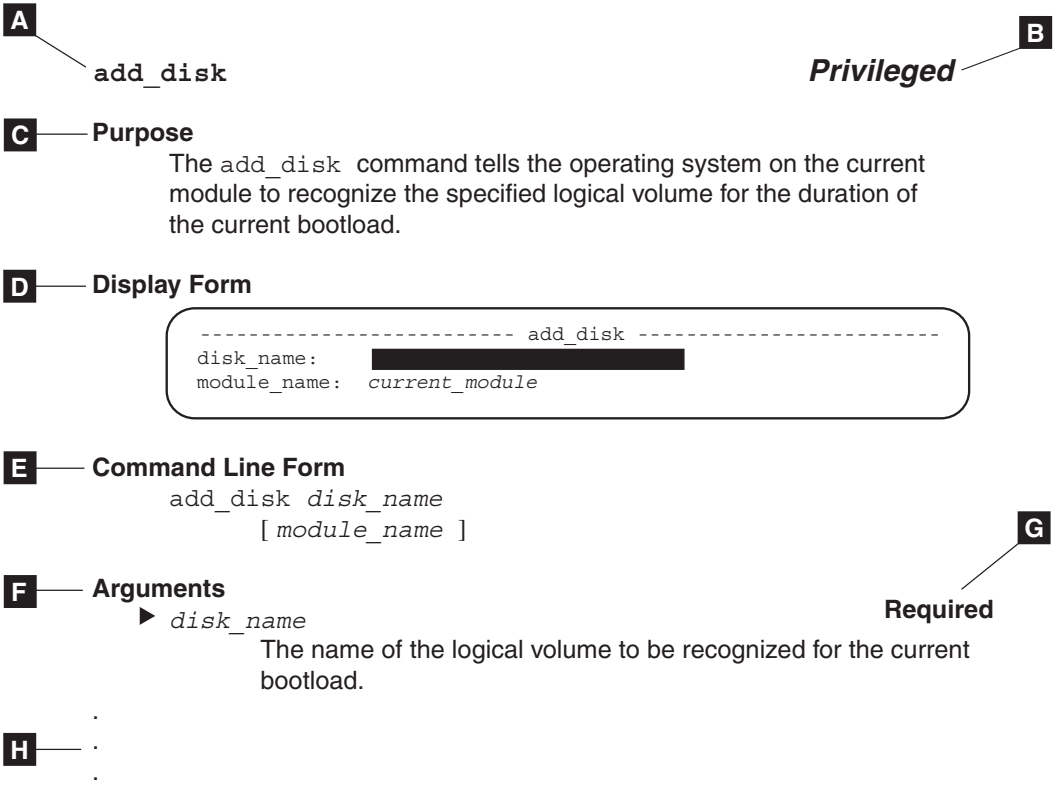
  ```
  list_users -module module_name
  ```

- 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  *.*
  ```

## Format for Commands and Requests

This section describes the format conventions that Stratus manuals use to document commands and requests. (A *request* is typically a command used within a subsystem, such as `analyze_system`.) Note that the command and request descriptions do not necessarily include each of the parts described in this section.




**A name**

The name of the command or request is at the top of the first page of the description.

**B *Privileged***

This notation appears after the name of a command or request that can be issued only from a privileged process.

**C Purpose**

Explains briefly what the command or request does.

**D Display Form**

Shows the form that is displayed when you type the command or request name followed by `-form` or when you press the key that performs the `DISPLAY FORM` function. Each field in the form represents a command or request argument. If an argument has a default value, that value is displayed in the form.

The following table explains the notation used in display forms.

**The Notation Used in Display Forms**

| Notation | Meaning |
|---|---|
| [redacted] | Required field with no default value. |
| [redacted] | The cursor, which indicates the current position on the screen. For example, the cursor may be positioned on the first character of a value, as in [redacted]`ll`. |
| *`current_user`*<br>*`current_module`*<br>*`current_system`*<br>*`current_disk`* | The default value is the current user, module, system, or disk. The actual name is displayed in the display form of the command or request. |

**E Command-Line Form**

Shows the syntax of the command or request with its arguments. You can display an online version of the command-line form of a command or request by typing the command or request name followed by `-usage`.

The following table explains the notation used in command-line forms. In the table, the term *multiple values* refers to explicitly stated separate values, such as two or more object names. Specifying multiple values is **not** the same as specifying a star name. When you specify multiple values, you must separate each value with a space.

**The Notation Used in Command-Line Forms**

| Notation | Meaning |
|---|---|
| *argument_1* | Required argument. |
| *argument_1*... | Required argument for which you can specify multiple values. |
| { *argument_1* \| *argument_2* } | Set of arguments that are mutually exclusive; you must specify one of these arguments. |
| [ *argument_1* ] | Optional argument. |
| [ *argument_1* ]... | Optional argument for which you can specify multiple values. |
| [ *argument_1* \| *argument_2* ] | Set of optional arguments that are mutually exclusive; you can specify only one of these arguments. |
| **Note:** Dots, brackets, and braces are not literal characters; you should **not** type them. Any list or set of arguments can contain more than two elements. Brackets and braces are sometimes nested. | |

**F** **Arguments**

Describes the command or request arguments. The following table explains the notation used in argument descriptions.

**G** **The Notation Used in Argument Descriptions**

| Notation | Meaning |
|---|---|
| CYCLE | There are predefined values for this argument. In the display form, you display these values in sequence by pressing the key that performs the CYCLE function. |
| **Required** | You cannot issue the command or request without specifying a value for this argument.<br><br>If an argument is required but has a default value, it is not labeled **Required** since you do not need to specify it in the command-line form. However, in the display form, a required field must have a value—either the displayed default value or a value that you specify. |
| **(Privileged)** | Only a privileged process can specify a value for this argument. |

**H** The following additional headings may appear in the command or request description: Explanation, Error Messages, Examples, and Related Information.

**Explanation**

Explains how to use the command or request and provides supplementary information.

**Error Messages**

Lists common error messages with a short explanation.

**Examples**

Illustrates uses of the command or request.

**Related Information**

Refers you to related information (in this manual or other manuals), including descriptions of commands, subroutines, and requests that you can use with or in place of this command or request.

## Online Documentation

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

You can access the VOS StrataDOC Web site, at no charge, at `http://stratadoc.stratus.com`. A copy of the VOS StrataDOC CD-ROM is included with this release. You can also order additional copies from Stratus.

This manual is available on the VOS StrataDOC Web site.

For information about ordering the VOS StrataDOC CD-ROM, 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.
- Customers in North America can call the Stratus Customer Assistance Center (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 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 by using the command `comment_on_manual`. To use the `comment_on_manual` command, your system must be connected to the RSN. Alternatively, you can email comments on this manual to `comments@stratus.com`.

The `comment_on_manual` command is documented in the manual *VOS System Administration: Administering and Customizing a System* (R281) and the *VOS Commands Reference Manual* (R098). There are two ways you can use this command to send your comments.

- 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, `R388`, 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 Open StrataLINK Administration

Open StrataLINK is the open-network implementation of StrataLINK and StrataNET, the two former Stratus proprietary networks that establish VOS communications among Stratus modules and systems. Open StrataLINK (OSL) works with STREAMS TCP/IP (STCP) to enable communication between Stratus modules and systems in a standard network environment using Ethernet local area networks (LANs).

NOTES

1. This revision (-06) of the *VOS Administrator's Guide for Open StrataLINK* (R388) documents OSL administration for VOS Release 16.1.0 (and later).
2. Modules running VOS Release 15.0.0 (or later) require OSL to establish VOS communications with other modules in a single-system configuration; Modules running VOS Release 15.0.0 (or later) do not support StrataLINK (specifically, the proprietary StrataLINK hardware).
3. Beginning in VOS Release 15.0.0, you must configure OSL networks with STCP because VOS Release 15.0.0 (and later) does not support OS TCP/IP.
4. Stratus supports OSL compatibility only between consecutive major releases (for example, VOS Releases 16.*x*.*x* and 15.*x*.*x*); therefore, in an OSL network, all modules within a system must be within one major VOS release of one another.

This manual describes how to administer an OSL network. This chapter, which contains the following sections, presents an overview of OSL administration.



# How to Use This Manual

For an overview of OSL administration, read this chapter. The type of network you are configuring determines how to use the other chapters, as the following sections describe.



## Creating a Single-System OSL Network

When you are creating a single-system configuration, read ‘‘Checklist: A Single-System OSL Network’’ on page 2-1 for a quick configuration checklist. You should also read the following chapters that apply to your configuration.

- Read Chapter 3 if the system you are configuring is a single-system OSL network.
- Read Chapter 4 if you are adding a single-system OSL network to a multiple-system OSL network.

## Creating a Multiple-System OSL Network

When you are creating a multiple-system OSL network, read ‘‘Checklist: A Multiple-System OSL Network’’ on page 2-2 for a quick configuration checklist. Read Chapter 3 and Chapter 4 if you are configuring a multiple-system OSL network. Configure each single-system OSL network as described in Chapter 3, then configure the multiple-system OSL network as described in Chapter 4.

## Additional Administrative Information

The `osl_admin` command and its requests enable you to perform certain administrative tasks on an OSL network and to monitor the network. For information, see ‘‘The OSL Administrative Tool `osl_admin`’’ on page 1-11 as well as the `osl_admin` command description.

While you are creating a single-system or multiple-system configuration, see Chapter 5 and Chapter 6 for information about maintaining various aspects of your network configuration. Chapter 5 explains how to handle errors; Chapter 6 describes the administrative commands, include the `osl_admin` command.

To help you understand how the different configuration procedures work together, Chapter 3 and Chapter 4 have a common element, the sample system `%admin`. In Chapter 3, you first configure `%admin` as a three-module system using OSL only. In Chapter 4, you add `%admin` as one of several systems in a sample network configuration using OSL.

# Overview of Open StrataLINK

OSL software enables TCP/IP communications among modules within a **single** system as well as among modules within **multiple** systems, in both LANs and wide area networks (WANs).

When you configure modules running VOS Release 16.1.0 (or later) for OSL, you **must** observe the software-configuration requirements of STCP. Therefore, before configuring modules to support OSL, you **must** be familiar with the configuration procedures for STCP, as described in the *VOS STREAMS TCP/IP Administrator's Guide* (R419). (If you are converting from OS TCP/IP to STCP, you must also follow the procedures in the *VOS STREAMS TCP/IP Migration Guide* (R418).) You must also ensure that the required software and hardware components are available on all of the modules that communicate using OSL.

This revision (-06) of the manual documents the administration of OSL for modules running VOS Release 16.1.0 (or later). The following sections provide additional information about OSL.



## The OSL Architecture

The OSL architecture consists of a driver for OSL and the OSL server processes working with STCP to communicate with VOS user applications through a client/server architecture. A *client/server architecture* comprises a client and a server: the client handles requests from a user application and forwards the requests to the server, which processes the requests and sends responses back to the client. A *driver* is an operating-system program module that controls the input and/or the output of communications.

The driver for OSL that STCP requires is SOSL Net driver (`sosl_net_driver`); it establishes communications through STCP. A communications protocol, SOSL Net driver controls the client and server functions of OSL communications after it is activated with the `configure_comm_protocol` command. The driver for OSL controls client communications within a single system and among multiple systems by establishing communications with the server side of a remote module over a TCP port. SOSL Net driver also executes requests that require kernel processing. When requests require user-level processing, SOSL Net driver forwards them to OSL server processes. (The *OSL server process* (`osl_server`) is a user-level program module.) For information on configuring SOSL Net driver, see ‘‘STCP Requirements’’ on page 1-9.

These software components provide the interface to STCP, which handles all communications with the STCP network interface, the logical interface as well as the physical interface. In any OSL network configuration, the hardware that provides the physical connection between a module and a LAN is sometimes called the TCP/IP *network interface*. The hardware that provides the physical connection is also called the *physical interface*, and is typically a PCI adapter. The *logical interface* is the representation of the physical interface in software. Each STCP logical interface on a module is associated with a host name, and hosts are associated with routes (see ‘‘Configuring Routes in STCP’’ on page 1-9). (A *host* is a module, router/gateway, workstation, or other network device that is accessible in a TCP/IP network environment.)

Figure 1-1 shows the client side of the OSL architecture and Figure 1-2 shows the server side. Both figures show a fault-tolerant configuration in which the module provides two different logical interfaces, each of which is associated with its own physical interface, and each physical interface connects to a different LAN. In other words, the module provides two connections to independent Internet Protocol (IP) networks or subnetworks.

A network with two logical interfaces, two physical interfaces, and two independent IP routes allows communications between modules within one system or across multiple systems to continue if one network interface or LAN fails.

STCP uses the STREAMS Data-Link Multiplexor (SDLMUX) as a communications driver that, for fault tolerance, partners two physical interfaces into one SDLMUX group as the logical interface for the physical interfaces. You can optionally configure the OSL physical interfaces into SDLMUX groups, for an additional level of fault tolerance (see ‘‘OSL Configurations and Fault Tolerance’’ on page 1-7).

Whether each LAN is dedicated only to OSL communications or provides other network resources depends on the size and scope of your entire network configuration. For performance reasons, you may want to dedicate each LAN to OSL communications.

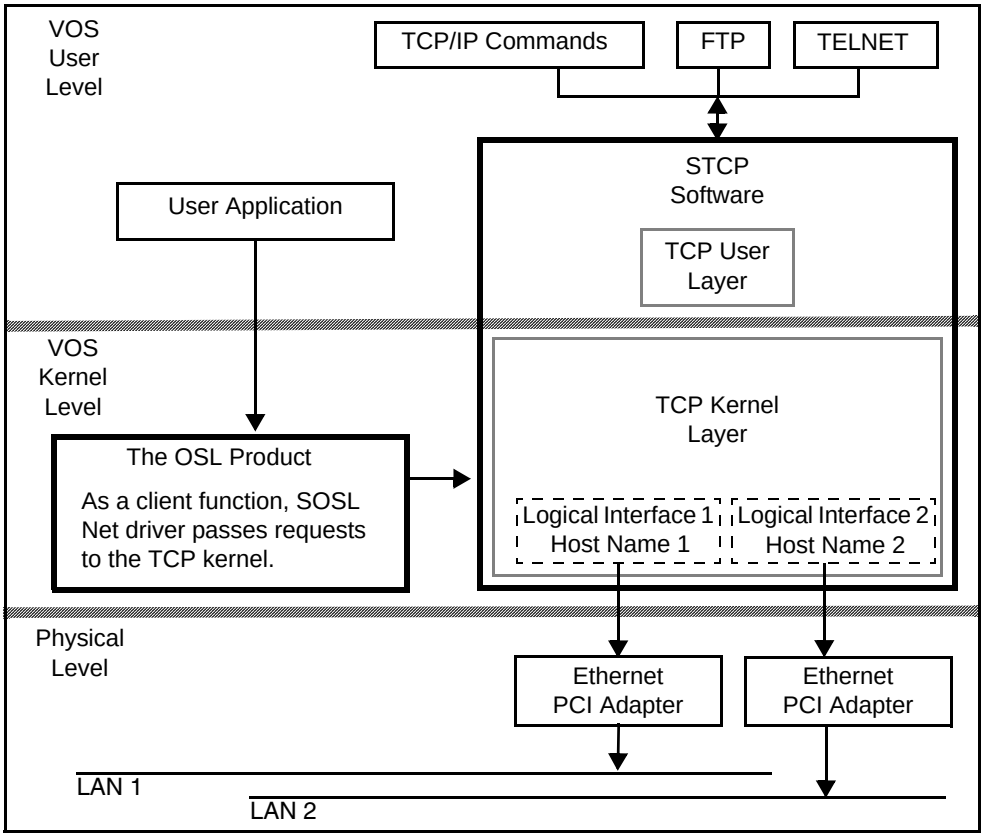



**Figure 1-1. The OSL Architecture: Client Side**

In Figure 1-1, a user application issues a request, which is sent to OSL. In OSL, the client side of the driver for OSL sends the request to the TCP kernel, which then sends the request through one of the logical interfaces, and then one of the physical interfaces to a LAN.

In Figure 1-2, the request passes through one of the physical interfaces on the receiving module, then the logical interface, and through the TCP kernel layer, which the driver controls for OSL. If the driver can execute the request, it does so; otherwise, it forwards the request to OSL server processes at the VOS user level.

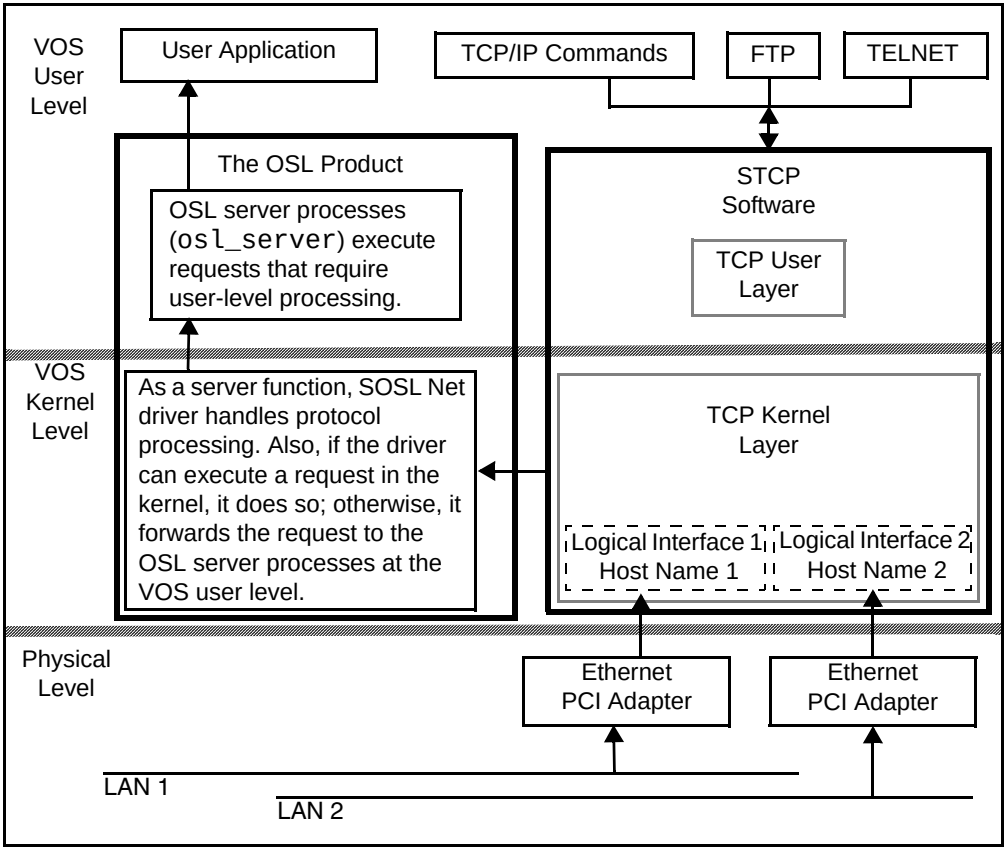



**Figure 1-2. The OSL Architecture: Server Side**

OSL attempts to connect to modules as in a round-robin; that is, OSL attempts to connect to each module individually and sequentially within the network. Initially, OSL attempts to connect to each configured interface on a module using a short time-out value. If these attempts fail, OSL repeats the attempt with a longer time-out value. If OSL still cannot connect to any interface on a module, the module is considered inaccessible and assumed to be offline as long as either monitoring is in use or no other transactions are active.

OSL is included on the VOS release tape. For instructions on how to install OSL separately from a VOS release tape, follow the instructions listed in the *VOS Installation Guide* (R386).

For information on OSL configurations and fault tolerance, see ‘‘OSL Configurations and Fault Tolerance’’ on page 1-7. The basic requirements for single-system OSL

configurations are documented in ‘‘Single-System Configurations’’ on page 1-13 and in Chapter 3; the basic requirements for multiple-system OSL configurations are documented in ‘‘Multiple-System Configurations’’ on page 1-16 and in Chapter 4.

## OSL Configurations and Fault Tolerance

An OSL configuration must be fault-tolerant to ensure continued operation despite the failure of a single component or single path. To achieve fault tolerance, you must configure each module with at least two physical interfaces, where each physical interface connects to a different IP subnet, thus creating two paths between each module. This configuration ensures that the module can transmit and receive data over two different IP subnets. The devices (for example, switches, routers, etc.) that support a subnet must be disjoint from the devices that support the redundant subnet to ensure that the failure of a single cable or device does not affect both paths. You may configure more than two physical interfaces for each module, but you must always configure at least two physical interfaces.

In addition, two STCP routes must exist between all modules in an OSL network, to ensure that STCP has two paths for transmitting and receiving data. If the modules are not on the same subnets, you must explicitly define routes using the STCP `route` command. For information on STCP routes, see ‘‘Configuring Routes in STCP’’ on page 1-9.

OSL detects communication failures by monitoring the connection status of each monitored module (see ‘‘The `set_monitoring` Request’’ on page 6-55). When monitoring is disabled, OSL detects communication failures by establishing maximum time limits for normal (that is, non-monitoring) operations, or by detecting various transmission-related error conditions. When OSL determines that it is unable to communicate over a particular path, it switches to a different path.

Optionally, you may also use SDLMUX groups to provide faster detection and correction of certain types of failures. By design, the time limits that SDLMUX uses to detect a failure (and then to move traffic to the backup interface) are longer than the time limits that OSL uses. However, whereas OSL relies exclusively on time limits to redirect operations, SDLMUX can often quickly detect certain types of hardware errors and immediately redirect operations without OSL involvement. In this case, SDLMUX simply switches to the alternate physical interface and OSL (using TCP/IP) can continue to communicate using the same logical interface, avoiding the need for retransmission and recovery. Thus, you may wish to configure one or more SDLMUX groups to use multiple local hardware components in order to provide for instantaneous message redirection in the case of component failure. Therefore, you should configure SDLMUX groups for OSL connections supporting direct queues.

You must configure at least two SDLMUX groups to ensure fault-tolerant operation of OSL. Since each SDLMUX group requires two physical interfaces, an OSL configuration requires four physical interfaces for two SDLMUX groups.

Note that although each SDLMUX group uses two physical interfaces, the group only provides a single logical interface to one IP subnet because each SDLMUX group has one IP address. And although each SDLMUX group is locally fault-tolerant, each SDLMUX group cannot be considered fault-tolerant for communication across the network.

Figure 1-3 shows the lower levels of the OSL architecture with the physical interfaces configured in SDLMUX groups.

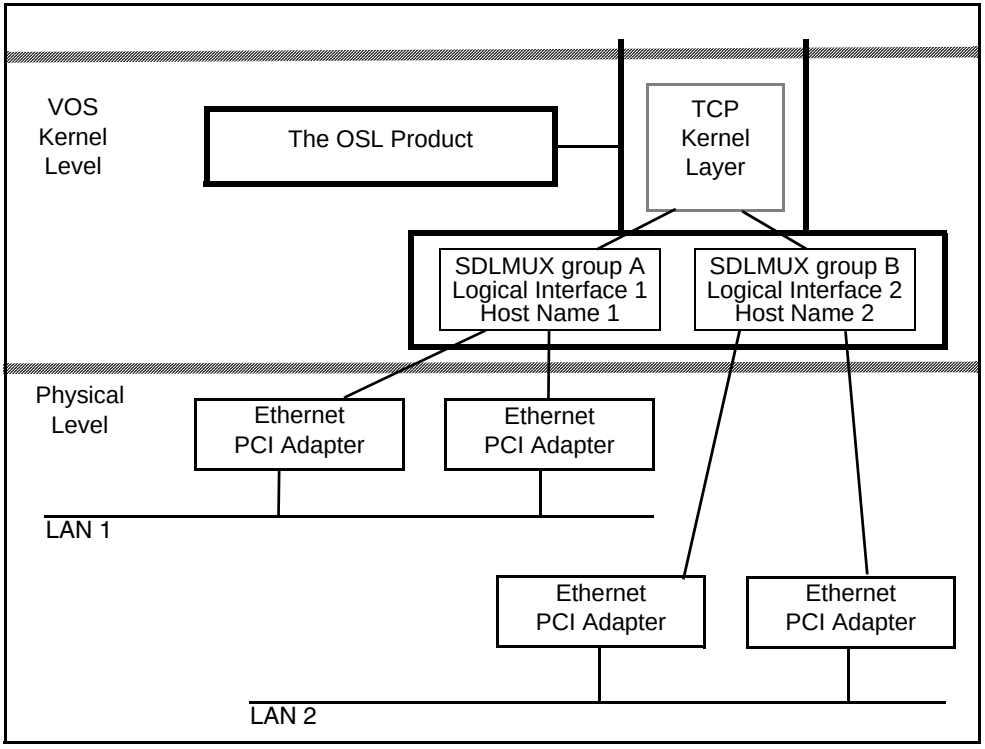



**Figure 1-3. The OSL Architecture (Lower Levels) with SDLMUX Groups**

Regardless of the number of routes between modules, and regardless of the failure and recovery actions it may undertake, OSL ensures that all user-visible network operations are executed exactly once, and are executed in the original order. For example, if five messages are entered into a direct queue in the order A,B,C,D,E, then those five messages are presented to the server in the same order, regardless of whether the queue is local or remote with respect to the requester or server. This property of OSL is a fundamental aspect of its operation and is unaffected by the type or number of routes that you configure for its use.

## STCP Requirements

On modules running VOS Release 16.1.0 (or later), OSL requires STCP. Note that modules running VOS Release 15.0.0 (or later) support only STCP for TCP/IP functionality.

NOTES

1. Because OSL works with STCP, you typically need to work with your network administrator to configure STCP for OSL.
2. Beginning in VOS Release 15.0.0, STCP is included with VOS on the VOS release tape; it does not have its own tape.
3. You must use Ethernet PCI adapters as the STCP physical interface for OSL networks. Modules running VOS Release 15.0.0 (or later) do not support OSL networks configured through any other type of physical interface.
4. If you want to migrate from OS TCP/IP to STCP, see the *VOS STREAMS TCP/IP Migration Guide* (R418).

Before you configure modules for OSL, you **must** configure STCP. Configuring STCP includes creating routes to remote hosts and networks. For information about how to configure STCP, see the *VOS STREAMS TCP/IP Administrator's Guide* (R419).

In **addition** to the standard STCP configuration procedures described in the *VOS STREAMS TCP/IP Administrator's Guide* (R419), you must also add an entry to the `devices.tin` table input file for the OSL multiplexor, activate SOSL Net driver, and start the OSL daemon process. The following sections describe these procedures.


- "Configuring Routes in STCP" on page 1-9
- "Adding the OSL Multiplexor to the `devices.tin` File" on page 1-10
- "Activating SOSL Net Driver" on page 1-10
- "Starting the OSL Daemon Process" on page 1-11


### Configuring Routes in STCP

In STCP networks, a *route* is a path by which data flows between two hosts. Simple networks may have only a single route between hosts. Complex networks may have multiple routes between hosts. In the latter case, some routes may offer lower response times, higher capacity, or lower costs. Some hosts may only be accessible via certain gateways.

Two STCP routes must exist between all modules in an OSL network, to ensure that STCP has two paths to be used for communicating with every other module. If the modules are **not** on the same subnets, you must explicitly define routes using the STCP `route` command. You use the STCP `route` command to add, delete, or change entries in the STCP routing table that control the path used for each outbound packet. Note that only one route can be defined to a given destination.

However, STCP automatically creates a route for the local network/subnetwork, based on the network address and subnet mask of a logical interface, when you configure the interface using the STCP `ifconfig` command. These local routes are **not** displayed by the `route print` command but are part of the routing table. If, for example, you have two modules (modules `m1` and `m2`), where each module has a physical interface (which is associated with a logical interface) connected to network A and a different physical interface (which is associated with a logical interface) connected to network B, STCP automatically creates a route between the `m1` and `m2` logical interfaces to network A, and STCP automatically creates a different route between the `m1` and `m2` logical interfaces to network B.

For information on the STCP `ifconfig` and `route` commands, as well as complete information on STCP routes, see the *VOS STREAMS TCP/IP Administrator's Guide* (R419).

### Adding the OSL Multiplexor to the `devices.tin` File

When you create the appropriate entries in the `devices.tin` table input file for the STCP components and related components (for example, device entries for the physical and logical interfaces), you must also add an entry for the OSL multiplexor. By convention, the name of the OSL multiplexor is `stcp_osl_mux.`*`module_name`*. The following entry is for the OSL multiplexor on module `m1`; therefore, its name is `stcp_osl_mux.m1`.

```
/    =name               stcp_osl_mux.m1
     =module_name        m1
     =device_type        streams
     =streams_driver     sosl_net_driver
     =clone_limit        1
```

After you have added the `devices.tin` file entries, issue the `create_table` command to re-create the `devices.table` file in the `>system>configuration` directory, copy the `devices.table` file to the `>system` directory, and issue the `configure_devices` command to activate the `devices.table` file.

### Activating SOSL Net Driver

You must also issue the command that activates SOSL Net driver (`sosl_net_driver`), which is the driver that STCP requires for OSL. You activate SOSL Net driver for the current bootload by issuing the following command; to activate

SOSL Net driver for subsequent bootloads, uncomment this command line in the `module_start_up.cm` file.

```
configure_comm_protocol sosl_net_driver
```

NOTE

You cannot unload STCP after SOSL Net driver is started. If STCP is stopped, you should restart the OSL daemon. If the incoming or outgoing connections are not resolved, contact the CAC for assistance.

### Starting the OSL Daemon Process

You must start the OSL daemon (`osl_daemon`) process for the OSL multiplexor and the STCP protocol driver for the current bootload. To do so, issue the following command at the command line; to start the OSL daemon process for subsequent bootloads, uncomment this command line in the `module_start_up.cm` file, substituting *`MODULE`* with the module name. You can also use this command to restart a daemon that has been terminated.

```
start_process 'osl_daemon #stcp.MODULE #stcp_osl_mux.MODULE'
     -privileged -priority 7
```

NOTES

1. If the OSL daemon process is terminated or is not started, the current module cannot establish incoming or outgoing connections; therefore, other modules cannot communicate with it.
2. If the OSL daemon process is terminated, connections that have already been established are not affected.

## The OSL Administrative Tool `osl_admin`

The `osl_admin` command is an OSL administrative tool that provides requests for different administrative and troubleshooting tasks.

You can use the following requests to perform OSL administrative tasks: `compare_configuration`, `disable_destination`, `enable_destination`, `get`, `reset_port`, and `status`.

You can monitor OSL using the `set_monitoring` request. OSL detects communication failures by monitoring the connection status of each monitored module. When monitoring is disabled, OSL detects communication failures by establishing maximum time limits for normal (that is, non-monitoring) operations, or by detecting

various transmission-related error conditions. When OSL determines that it is unable to communicate over a particular path, it switches to a different path.

You can use tracing requests to monitor the progress of transactions through SOSL Net driver loaded on your module (`sosl_net_driver`). The tracing requests are as follows:

```
adjust_saved_trace          restart_tracing
create_trace_buffer         save_trace_buffer
display_trace               set_default_trace_flags
list_saved_traces           set_default_trace_size
match                       set_trace_flags
merge_trace_buffers         sleep
resize_trace_buffer         test
```

When tracing is enabled, each module in a system is allocated its own trace buffer. Each trace buffer is an array of trace entries 40 bytes long, and each transaction uses a minimum of 12 of these entries. As the load on OSL increases, the driver may queue transactions, or it may need to make multiple calls to TCP, which increases the number of entries used for each transaction.

By default, tracing is enabled and each trace buffer contains 500 entries. Also by default, OSL is set to automatically halt tracing if it marks another module offline. The `restart_tracing` request searches for buffers that have been automatically stopped, saves them to a file, and restarts tracing. You can disable automatic tracing using the `set_trace_flags` or `set_default_trace_flags` request.

**CAUTION**

The tracing capabilities of OSL provide the CAC with a log of recent events when a module is marked offline or a system interruption occurs. Although you can disable tracing or change the size of trace buffers to decrease memory consumption, be aware that doing so affects Stratus's ability to diagnose problems quickly and accurately.

# OSL Configurations

OSL supports the following configurations, which also require STCP.

- a single-system configuration—This configuration consists of two or more modules that use OSL with STCP (and, optionally, SDLMUX) to form a single system and establish TCP/IP connections over two or more IP networks or subnetworks (subnets). All modules in a single system can share resources. From a user's point of view, the modules function as if they were a single module. ''Single-System Configurations'' on page 1-13 presents detailed information about this type of configuration.
- a multiple-system configuration—This configuration consists of two or more systems that use OSL with STCP (and, optionally, SDLMUX) to establish TCP/IP connections over a local or wide area. (The WAN functionality of OSL is sometimes called Open StrataNET.) One module in each system must serve as a bridge module to handle incoming and outgoing requests. ''Multiple-System Configurations'' on page 1-16 presents detailed information on this type of configuration.

## Single-System Configurations

A single-system configuration uses OSL and STCP (and, optionally, SDLMUX) to connect 2 to 32 modules as a single VOS system. TCP/IP connections among the modules in the system let VOS users view the modules as a single module. (If you configure a single-module system, the module can provide TCP/IP connections that enable it to communicate with other systems using OSL and STCP.)

The following sections provide additional information about single-system configurations.



### Configuration Requirements of Single-System Configurations

A single-system configuration has the following requirements.

- Each module in the system must be configured to run STCP. You typically need to work with your network administrator to configure STCP.
- To achieve fault tolerance, you must configure each module with at least two interfaces to ensure that the module can transmit and receive data over two different IP subnets. For information, see ''OSL Configurations and Fault Tolerance'' on page 1-7.
- You must create routes to remote hosts and networks using the STCP `route` command, to ensure that STCP has two paths for transmitting and receiving data. However, you do not need to create routes for a local network/subnetwork because

STCP automatically creates them. For information, see “Configuring Routes in STCP” on page 1-9.

- The dual IP routes may contain third-party LAN interconnect hardware as long as the hardware does not introduce a single point of failure (that is, no one cable segment, bridge, or router/gateway common to the routes should be able to stop communications between modules).
- On each module, `osl_server` processes must be running, and SOSL Net driver must be activated. The VOS module startup file, `module_start_up.cm`, starts the `osl_server` processes and activates SOSL Net driver (`sosl_net_driver`).
- All modules in the system must contain a new modules configuration-table file (`new_modules.table`). The file must contain an entry with STCP-related information for each module. The STCP-related information includes the complete IP addresses of the logical interfaces and a base TCP port number (OSL does not support fully qualified host names).

NOTE

Even a single-module system that uses OSL for cross-system communications requires the `new_modules.table` file, since VOS uses the entry for the current module when processing any type of OSL information. When OSL handles cross-system communications on this system’s single bridge module, the entry for the bridge module in the `new_modules.table` file must define the availability of OSL.

You can monitor OSL using the `set_monitoring` request (see “The `set_monitoring` Request” on page 6-55) of the `osl_admin` command.

You can optionally also use SDLMUX groups to provide faster detection and correction of certain types of failures. See “OSL Configurations and Fault Tolerance” on page 1-7.

### A Sample Single-System Configuration

Figure 1-4 illustrates a sample single-system configuration, where each of the four modules in the system `%oplink` connects to two IP network or subnets. Each module has one physical interface for each network/subnet, for a total of two physical interfaces on each module.

In all examples, module names have the prefix `#m` and system names have the prefix `%`.

NOTE

To provide fault tolerance, all modules communicating with OSL need at least two independent IP routes to one another. Although the physical layout of each LAN is transparent to OSL (for example, the use of bridges, routers, and/or gateways to connect LAN segments and multiple LANs), you should carefully evaluate the use of additional hardware and aim to protect modules from a single point of failure along the routes.

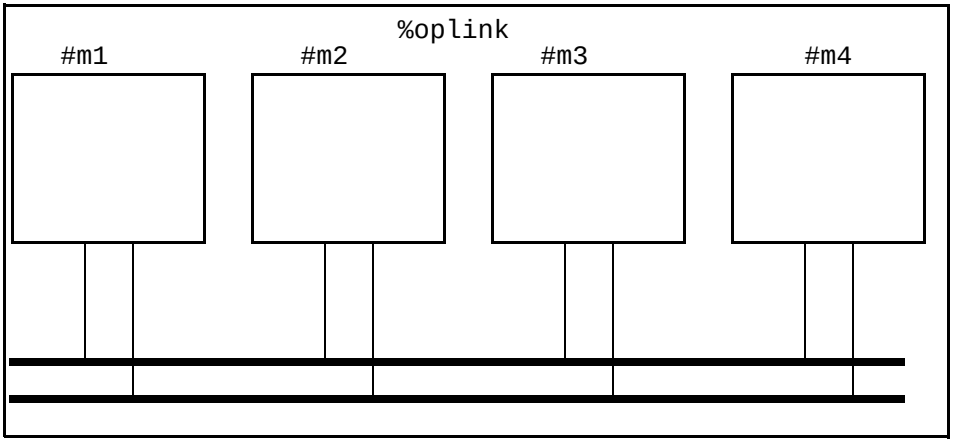



**Figure 1-4. A Single-System Configuration**

Figure 1-5 illustrates a sample single-system configuration, where each module has two physical interfaces, configured in one SDLMUX group, for each network/subnet, for a total of four physical interfaces on each module. Each of the four modules in the system `%oplink` connects to two IP network or subnets.

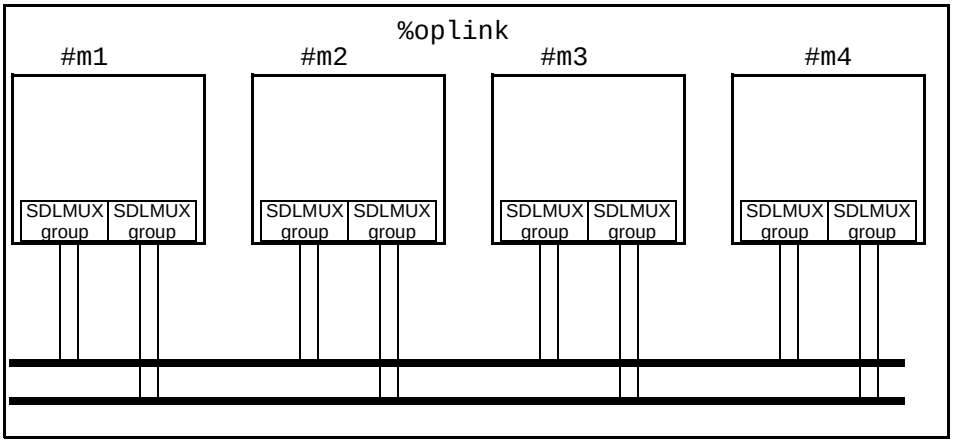



**Figure 1-5. A Single-System Configuration with SDLMUX Groups**

Chapter 3 describes in detail how to create and start a single-system configuration.

## Multiple-System Configurations

A multiple-system configuration consists of 2 to 255 systems. In this type of configuration, you can define single modules and/or groups of modules as separate systems, and use OSL with STCP to enable the systems to communicate.

A multiple-system configuration offers several advantages over a single-system configuration. For example, management of computer resources is easier when the module(s) used by each department or group within an organization can be defined as separate systems. In addition, security is improved with a network access configuration-table file (`network_access.table`), which controls access to the network on a system-by-system basis. This configuration-table file lets you control which systems and users in the network can access a given system. Moreover, network traffic can be controlled by dividing users among modules and systems.

The following sections provide additional information about multiple-system configurations.


- ‘‘Configuration Requirements of Multiple-System Configurations’’ on page 1-17
- ‘‘A Sample Multiple-System Configuration’’ on page 1-18

### Configuration Requirements of Multiple-System Configurations

A multiple-system configuration has the following requirements.

- Each system must have a module that is configured to run STCP.
- To achieve fault tolerance on the bridge module, you must configure it with at least two interfaces to ensure that the module can transmit and receive data over two different IP subnets. For information, see ‘‘OSL Configurations and Fault Tolerance’’ on page 1-7.
- You must create routes to remote hosts and networks using the STCP `route` command, to ensure that STCP has two paths for transmitting and receiving data. However, you do not need to create routes for a local network/subnetwork because STCP automatically creates them. For information, see ‘‘Configuring Routes in STCP’’ on page 1-9.
- The dual IP routes may contain third-party LAN interconnect hardware as long as the hardware does not introduce a single point of failure (that is, no one cable segment, bridge, or router/gateway common to the routes should be able to stop communications between modules).
- You must designate one module in each system as a bridge module; this module must run the OSL server (`osl_server`) processes and must activate SOSL Net driver (`sosl_net_driver`) to handle intersystem requests. All cross-system communications must occur through the bridge module. The remaining modules are called *nonbridge modules*. The bridge module forwards requests for its nonbridge modules.

  A bridge module typically has outbound/inbound OSL servers. *Outbound/inbound OSL servers* forward outbound requests from nonbridge modules on the local system to other bridge modules on remote systems. They also receive inbound requests from other bridge modules on remote systems for nonbridge modules on the local system. You create outbound/inbound OSL servers by specifying the `-super` argument of the `osl_server` command when you start the OSL server processes. When you specify the `-super` argument for one `osl_server` command, you must specify it for **all** `osl_server` commands that you issue on the same module. All OSL servers on one module must be of the same type.

- The bridge module of each system communicating using OSL and STCP must have the new backbone systems configuration-table file (`new_backbone_systems.table`), which defines how a bridge module accesses each remote system using OSL and STCP.
- Each system’s nonbridge modules must have the new systems configuration-table file (`new_systems.table`), which specifies the information needed to establish communications with other systems using OSL through the local-system bridge module.

On the bridge module, you can monitor OSL using the `set_monitoring` request (see "The `set_monitoring` Request" on page 6-55) of the `osl_admin` command.

### A Sample Multiple-System Configuration

Figure 1-6 illustrates a sample multiple-system configuration in which three systems (`%os1`, `%os2`, and `%os3`) are connected using OSL and STCP over two LANs (networks or subnets). In this configuration, the modules within each system also communicate using OSL, so the cross-module and cross-system communications can exist over the same LANs.

Each bridge module in the configuration (`%os1#m2`, `%os2#m1`, and `%os3#m10`) handles all incoming and outgoing traffic for its respective system. These bridge modules start the outbound/inbound `osl_server` processes and activate SOSL Net driver (`sosl_net_driver`). Note that while the single-module system `%os3` is shown as accessible over a local area, it could just as easily reside at a remote site and be accessible over a wide area. The wide-area connection, of course, requires the use of additional third-party hardware.

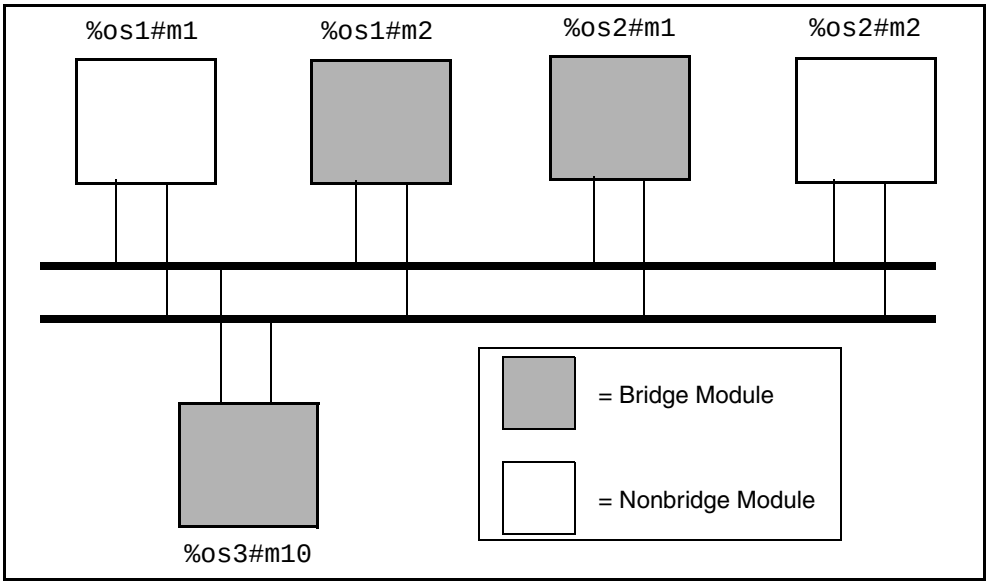



**Figure 1-6. A Multiple-System Configuration**

Figure 1-7 illustrates the sample multiple-system configuration of Figure 1-6, where each module has two physical interfaces, configured in an SDLMUX group.

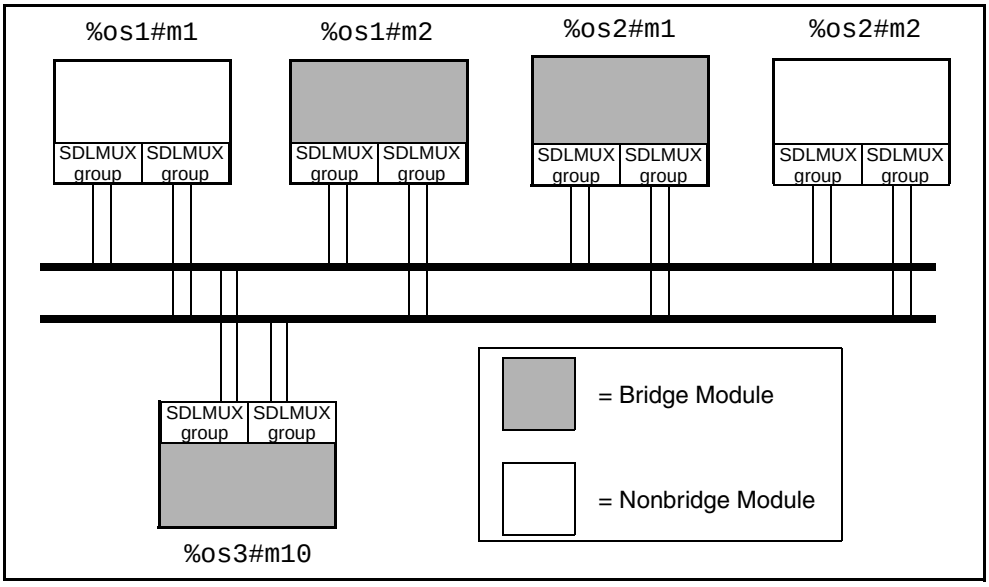



**Figure 1-7. A Multiple-System Configuration with SDLMUX Groups**

# Standard VOS System Administration Procedures

The following sections describe some standard VOS system administration procedures, as they apply to OSL.


- "Working with Configuration-Table Files" on page 1-19
- "Issuing Commands from the Module Startup File" on page 1-24


## Working with Configuration-Table Files

A *configuration-table file* is a structured record file that VOS uses to recognize the components of a system or network. A configuration-table file contains a record for each component that it configures.

Table 1-1 lists and briefly describes many of the configuration-table files referenced in this manual. The table also identifies where the files are documented.

**Table 1-1. Configuration-Table Files**

| Configuration-Table File | Description | Documentation |
|---|---|---|
| `devices.table`, the device configuration-table file | Defines each device in a system. | Chapter 3 and *VOS System Administration: Configuring a System* (R287) |
| `disks.table`, the disks configuration-table file | Defines each disk in a system. | Chapter 3 and *VOS System Administration: Configuring a System* (R287) |
| `network_access.table`, the network access configuration-table file | Defines the conditions under which users on remote systems can access files and other resources on the current OSL system. | Chapter 4 |
| `new_backbone_systems.table`, the new backbone systems configuration-table file | Defines how the bridge module of each system (or X.25 gateway module) accesses remote systems in a multiple-system OSL configuration. | Chapter 4 |
| `new_modules.table`, the new modules configuration-table file | Defines each module in an OSL configuration. | Chapter 3 |
| `new_systems.table`, the new systems configuration-table file | Defines how nonbridge modules access remote systems through their local-system bridge module in a multiple-system OSL configuration. | Chapter 4 |

The following sections provide additional information on working with configuration-table files.


- "Creating a Configuration-Table File" on page 1-20
- "Installing a Configuration-Table File" on page 1-21
- "Activating a Configuration-Table File" on page 1-22
- "Summary of Configuration Steps" on page 1-22


## Creating a Configuration-Table File

You create a configuration-table file (*`table_name`*`.table` file) by issuing the `create_table` command, which uses the following files as input.

- A data-description file, *`table_name`*`.dd`, is a format file that declares the template of the records in a configuration-table file. **Never** modify this file. However, if Stratus modifies this file for any reason, you must re-create, reinstall, and reactivate all *`table_name`*`.table` files that were based on the old format file. For example, if a

new release of VOS alters a data description file, the Software Release Bulletin describes the change and provides instructions for re-creating, reinstalling, and reactivating all relevant files.

- A table-input file, *table_name*.tin, is a text file that contains a record (that is, an entry) for each component that has been defined. To add, change, or delete a component's definition, begin by editing this file. The descriptions of configuration-table files in this manual focus on how to create entries for various components (for example, modules, devices, and systems) in the table input file.

When this manual states that a configuration-table file has been *created,* it means that the *table_name*.table file has been compiled from the *table_name*.dd file and the corresponding *table_name*.tin file. The create_table command creates the *table_name*.table file, using the record format specified in the *table_name*.dd file and the values specified in the *table_name*.tin file. Each record in the *table_name*.table file contains a complete set of values for defining a component, including the values specified in the *table_name*.tin file and default values for fields that are unspecified in the *table_name*.tin file.

The following example shows how to create the device configuration-table file (devices.table). The table-input file (devices.tin) and the data-description file (devices.dd) are in the current directory, which should be (master_disk)>system>configuration. The following command causes VOS to create the devices.table file in the (master_disk)>system>configuration directory.

```
create_table devices
```

For a detailed description of the create_table command, see the manual *VOS System Administration: Configuring a System* (R287).

### Installing a Configuration-Table File

When this manual states that a file has been *installed* on a module, it means that the file has been placed in the appropriate configuration directory. Data-description files (*table_name*.dd) and the corresponding table-input files (*table_name*.tin) are installed in the (master_disk)>system>configuration directory. Configuration-table files (*table_name*.table) are typically installed in the (master_disk)>system directory.

Some configuration-table files must be identical on all modules in the same system or network. In this case, you can create the configuration-table file on one module, called the *master module*, and then install it on all relevant modules. In a multimodule system, you should create a principal master disk, as described in the manual *VOS System Administration: Administering and Customizing a System* (R281), with VOS links to the (master_disk)>system>configuration directory from the other modules. Keep one copy of a configuration-table file in the (master_disk)>system directory and

another copy in the `(master_disk)>system>configuration` directory on the principal master disk.

### Activating a Configuration-Table File

When this manual states that a component (for example, a device) has been *configured* on a module, it means that the component is defined in a table-input file, the corresponding configuration-table file has been compiled or recompiled by the `create_table` command, and the configuration-table file has been activated by the appropriate configuration command. Each configuration-table file has a corresponding configuration command for configuring the components defined in it. For example, the `configure_devices` command configures the devices defined in the `devices.table` file. You must issue the configuration command on **each** module in a system or network.

You must reconfigure the components of a system or network each time the module is rebooted, which means that you typically issue the configuration commands from the `module_start_up.cm` file. (See "Issuing Commands from the Module Startup File" on page 1-24 for more information.) If you make changes to a configuration-table file, you can usually make additions (but not other changes) effective for the current bootload by creating a new configuration-table file and reissuing the corresponding configuration command.

During the configuration process, you may need to issue configuration commands that are not associated with configuration-table files. Other configuration commands, such as `add_module` and `add_system`, let you add a component to a system or network for the duration of the current bootload only (that is, temporarily, if the component is not defined in the appropriate configuration-table file). For a complete description of the `configure_comm_protocol` command, see the manual *VOS System Administration: Configuring a System* (R287); for a description of the `add_module` and `add_system` commands, see Chapter 6.

### Summary of Configuration Steps

The steps in this section summarize the procedures for creating, installing, and activating a configuration-table file. Both here and throughout the manual, some procedural steps instruct you to create a configuration-table file on the master module and issue the `broadcast_file` command to install the configuration-table file on all modules in the same system. Unless otherwise specified, create the initial configuration-table file in the `(master_disk)>system>configuration` directory. Then, store the configuration-table file on a principal master disk and install it in the `(master_disk)>system` directory on each module in the system, including the current module. To activate the configuration-table file, issue the appropriate configuration command from the `(master_disk)>system` directory on each module. Note that you must have logged in as privileged to issue any configuration command.

To create, install, and activate a configuration-table file, perform the following steps.

1. On the master module, in the `(master_disk)>system>configuration` directory, edit the table-input file, *`table_name`*`.tin`; then, issue the `create_table` command to create the configuration-table file, *`table_name`*`.table`.
2. Install the configuration-table file on all relevant modules in the system. Most configuration-table files are relevant to all modules in a multimodule system. Some, however, are relevant only to modules that perform a specific function, such as a bridge module. To install a configuration-table file, issue the `broadcast_file` command to copy the configuration-table file to the appropriate installation directory on each module in the system, which is usually `(master_disk)>system`. Note that you can also use the `copy_file` command to copy files (for example, to modules in other systems). The `broadcast_file` command copies files only to the `(master_disk)` directory of the specified modules. (The `broadcast_file` command is documented in the manual *VOS System Administration: Configuring a System* (R287); the `copy_file` command is documented in the *VOS Commands Reference Manual* (R098).)

   NOTE

   The `broadcast_file` or `copy_file` command will not perform the intended operation successfully unless a network connection exists either between the modules in a system or between the systems in a multiple-system configuration. For more information about establishing network connections, see Chapter 3 and Chapter 4.

3. Issue the configuration command (for example, `configure_modules`, `configure_devices`, `configure_systems`, or `configure_disks`) that activates the configuration-table file. In most cases, when you issue a configuration command for the current bootload, only **additions** take effect immediately; deletions and changes do not take effect until the next bootload. The following configuration commands have arguments for making changes and deletions effective for the current bootload.

   ```
   configure_devices -flush
   configure_systems -reset
   configure_modules -reset
   ```

   **CAUTION**

   Using the `-reset` argument with the commands `configure_systems` and `configure_modules` may

interrupt cross-system and cross-module communications.

For more information about the configuration commands `configure_modules` and `configure_systems`, see Chapter 6. For more information about the configuration commands `configure_devices` and `configure_disks`, see the manual *VOS System Administration: Configuring a System* (R287). That manual describes the different methods you can use to issue the configuration commands on each module, including creating a command macro with `start_process` commands and creating a subprocess on each module from the master module.

NOTE

The `network_access.table` file, which defines access to systems in multiple-system network configurations, takes effect as soon as it is broadcast. Therefore, it is not necessary to issue a configuration command to activate this configuration file.

4. Edit the `module_start_up.cm` file to issue the configuration command automatically each time the module is rebooted. (For additional information about the `module_start_up.cm` file, see "Issuing Commands from the Module Startup File" on page 1-24.)

See the manual *VOS System Administration: Configuring a System* (R287) for more detailed information about configuring a system's components, including descriptions of the commands and specific information about what to do if a configuration-table file or one of its input files is lost.

## Issuing Commands from the Module Startup File

VOS contains a module startup file (`module_start_up.cm`) that typically includes as comments all of the commands required to start a module. Comment lines in a command macro are preceded by an ampersand (`&`) and a space. To enable VOS to execute these commands at each reboot, you uncomment the command lines by deleting the ampersand and the space preceding each command. Note that in some command lines, you must also specify values for some arguments. Before working with the `module_start_up.cm` file, see the manual *VOS System Administration: Starting Up and Shutting Down a Module or System* (R282).

Many of the commands described in this manual must appear in the `module_start_up.cm` file. The following sections provide additional information about issuing commands from the module startup file to start OSL.


- "STCP Commands" on page 1-25
- "OSL Commands" on page 1-25

### STCP Commands

For STCP, the `module_start_up.cm` file includes command lines to perform the following actions.

- Add the library paths of the STCP command library
- Define the STCP host name for the module by issuing a `hostname` command
- Invoke the `start_stcp.cm` command macro

For complete information about the commands in the `module_start_up.cm` file necessary to start STCP, see the *VOS STREAMS TCP/IP Administrator's Guide* (R419). In order to run OSL with STCP, the `module_start_up.cm` file must also include the commands described in "STCP Requirements" on page 1-9.

### OSL Commands

To start the `osl_server` processes, the `module_start_up.cm` file must contain the `osl_server` command within a `start_process` command line. If you need to start additional `osl_server` processes on the module, set a value for the `ADDITIONAL_OSL_SERVERS` variable in the `module_start_up.cm` file. The value you set for the `ADDITIONAL_OSL_SERVERS` variable should equal the number of additional servers that you need.

In the `module_start_up.cm` file, you must also uncomment the following command, which enables the `module_start_up.cm` file to activate SOSL Net driver (`sosl_net_driver`). To uncomment a command, delete the ampersand and the space preceding it.

```
configure_comm_protocol sosl_net_driver
```

If you need to add other commands to the `module_start_up.cm` file, you can create one or more command macros that include these commands and call these command macros from the `module_start_up.cm` file. Using this method, you can edit the command macros when you need to change network configuration commands. This method is safer and more convenient than editing the `module_start_up.cm` file.

For additional information, see the following:

- "Commands That Start the OSL Server Processes" on page 3-17 and "Commands That Start the OSL Server Processes" on page 4-30
- the *VOS Commands Reference Manual* (R098) for information about the `start_process` command
- the manual *VOS System Administration: Configuring a System* (R287) for a complete description of the `configure_comm_protocol` command
- the *VOS Commands User's Guide* (R089) for information on developing command macros

# Chapter 2
# Quick Configuration Checklists

This chapter provides checklists for you to use as you configure a single-system or multiple-system OSL network.



To perform the procedures in the checklists, you must be a privileged user and have modify access to the `(master_disk)>system` directory. (The manual *VOS System Administration: Registration and Security* (R283) provides information on establishing user access rights.)

## Checklist: A Single-System OSL Network

Use this checklist as you configure a single-system OSL network. Each step provides references in Chapter 3 for more detailed information.

1. Plan your configuration. Typically, you need to work with your network administrator to plan this configuration because each module that will communicate using OSL must have the software and hardware required to run STCP. (For complete information on this step, see "Planning a Single-System Configuration" on page 3-1.)

   Step complete: ☐

2. Examine and, if necessary, update each module's master disk label. (For complete information on this step, see "Updating the Master Disk Label" on page 3-5.)

   Step complete: ☐

3. In the `new_modules.tin` file, define modules that will communicate using OSL, and create a `new_modules.table` file. (For complete information on this step, see "Using the New Modules Configuration-Table File" on page 3-6.)

   Step complete: ☐

4. In the `disks.tin` file, define the disks associated with all modules in the system, and create a new `disks.table` file. (For complete information on this step, see "Using the Disks Configuration-Table File" on page 3-12.)

   Step complete: ☐

5. In the `devices.tin` file, define the devices associated with all modules in the system, and create a new `devices.table` file. (For complete information on this step, see "Using the Device Configuration-Table File" on page 3-13.)

   Step complete: ☐

6. Issue VOS commands from the `module_start_up.cm` file that enable VOS on each module to permanently recognize all system components. (For complete information on this step, see "Modifying the `module_start_up.cm` File" on page 3-15.)

   Step complete: ☐

7. Register users on the system. (For complete information on this step, see "Registering Users on the System" on page 3-20.)

   Step complete: ☐

8. Reboot the system **or** issue the VOS commands that enable the modules to start communicating during the current bootload. (Since many of the commands you issue from command level are the same ones you issue from the `module_start_up.cm` file, see "Modifying the `module_start_up.cm` File" on page 3-15 for information on this step. "Configuring and Starting a New OSL System" on page 3-20 also provides information.)

   Step complete: ☐

## Checklist: A Multiple-System OSL Network

Use this checklist as you configure a multiple-system OSL network. Each step provides references in Chapter 4 for more detailed information.

1. Plan your configuration. Within each system, you must designate one module as a bridge module, which manages all inbound and outbound traffic for its local system (all other modules in that system are nonbridge modules). Typically, you need to work with your network administrator to plan this configuration because each module that communicates using OSL must have the software and hardware required to run STCP. (For complete information about this step, see "Planning a Multiple-System Configuration" on page 4-2.)

   Step complete: ☐

2. In the `new_systems.tin` file of each system, define all systems in the multiple-system configuration and identify the bridge module of the system being configured. Create a `new_systems.table` file and install it on all nonbridge modules of each system. (For complete information on this step, see "Using the New Systems Configuration-Table File" on page 4-6.)

   Step complete: ☐

3. In the `new_backbone_systems.tin` file of each system, define how all systems access each other using their respective bridge module. Create a `new_backbone_systems.table` file and install it on each system's bridge module. (For complete information on this step, see "Using the New Backbone Systems Configuration-Table File" on page 4-12.)

   Step complete: ☐

4. For each system, create a new network access configuration-table file (`network_access.table`) that establishes access rights to each system. (For complete information on this step, see "Using the Network Access Configuration-Table File" on page 4-21.)

   Step complete: ☐

5. Issue VOS commands from the `module_start_up.cm` file that enable VOS to permanently recognize certain components of the bridge and nonbridge modules (these commands include the `configure_systems` command). (For complete information on this step, see "Modifying the `module_start_up.cm` File" on page 4-29.)

   Step complete: ☐

6. Reboot modules in the configuration **or** issue the VOS commands that let you start multiple-system communications during the current bootload. (Since many of the commands you issue from command level are the same ones you issue from the `module_start_up.cm` file, see "Modifying the `module_start_up.cm` File" on page 4-29 for information on this step. "Starting Multiple-System OSL Communications" on page 4-32 also provides information.)

   Step complete: ☐

# Chapter 3
# Configuring a Single-System OSL Network

This chapter, which contains the following sections, describes the configuration procedures to create, start, and modify a single system of Stratus modules in an OSL network.



‘‘Checklist: A Single-System OSL Network’’ on page 2-1 provides a checklist for creating a single-system configuration. ‘‘Standard VOS System Administration Procedures’’ on page 1-19 provides an overview of how to work with configuration-table files and the `module_start_up.cm` file.

## Planning a Single-System Configuration

When planning a system that uses OSL, you must decide which modules will be in the system and how they will access each other. The system can support up to 32 modules and use OSL over STCP. Every module in a system should be able to communicate with every other module in the system over two independent networks or subnets, if the configuration is to be fault-tolerant. For information on OSL and fault tolerance, see ‘‘OSL Configurations and Fault Tolerance’’ on page 1-7.

NOTE

Because OSL works with STCP and uses a standard network environment such as Ethernet, you typically need

to work with your network administrator to plan the system.

As you plan the system, make one module the master module. You typically update various configuration-table files on the master module and install a copy of these files on the other modules in the system (that is, you use the `broadcast_file` command to *broadcast*, or copy, the files from the master module to the other modules).

In addition, you must provide a unique module name, module number, and station number for each module in the system. When you specify a unique module name, module number, and station number, the module can be uniquely identified within the system. In any OSL configuration, you must supply this information in each module's master disk label **and** in the new modules configuration-table file (`new_modules.table`), both of which are described in ''Updating the Master Disk Label'' on page 3-5 and ''Using the New Modules Configuration-Table File'' on page 3-6. The module name, module number, and station number you supply for a module in the new modules configuration-table file **must** match the name and numbers you supply in the module's master disk label.

In a system that uses only OSL to establish module communications, you must configure STCP and OSL on each module in the system, as the following sections describe.



## Configuring STCP

To create a single-system OSL configuration, you must first determine how the multimodule system will fit into your overall STCP network configuration. For example, decide whether the configuration warrants dedicated or public networks and determine how the physical layout accommodates dual routes of communications among the modules. When you are ready to configure each module for STCP, you must follow the hardware and software requirements defined in the *VOS STREAMS TCP/IP Administrator's Guide* (R419). Additional requirements specific to running OSL with STCP are described in ''STCP Requirements'' on page 1-9. You **must** follow the procedures described in that section in addition to the standard STCP configuration procedures in order to run OSL with STCP. (For information on migrating from OS TCP/IP to STCP, see the *VOS STREAMS TCP/IP Migration Guide* (R418).)

Typically, you need to work with your network administrator to configure the STCP portion of the OSL network. STCP configuration requirements described in the *VOS STREAMS TCP/IP Administrator's Guide* (R419) include the following procedures.

- loading the different portions of the STCP software
- defining STCP devices (such as the logical and physical interfaces on each module)
- editing the STCP database files to provide information such as the IP addresses
- issuing STCP commands to establish the logical interfaces and the routes

See also ''STCP Requirements'' on page 1-9 for information on the STCP requirements.

If you configure the minimum of two routes and the routes contain third-party LAN interconnect hardware, the hardware should not introduce a single point of failure. No one cable segment, bridge, or router/gateway common to the routes should be able to stop communications among the modules if it fails.

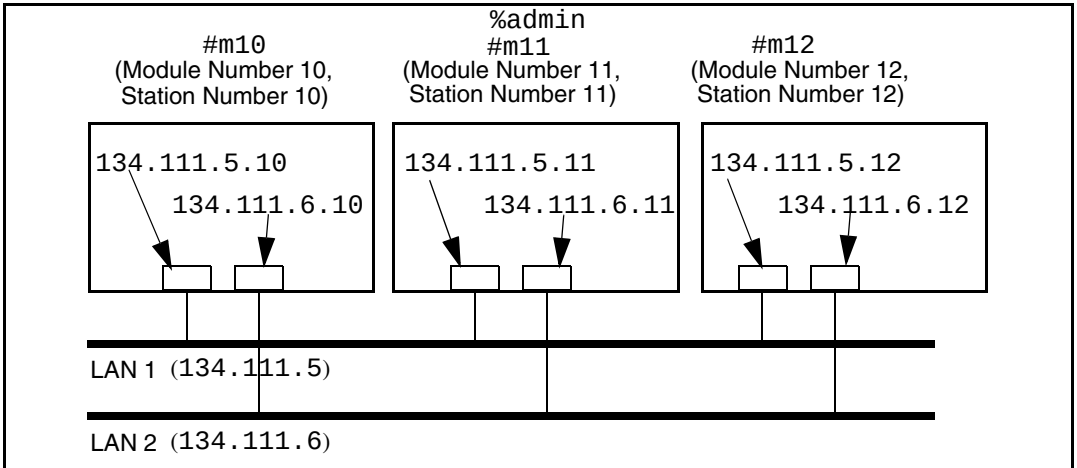
## Configuring OSL

After performing the STCP configuration procedures, you must perform the VOS configuration procedures that establish OSL on each module. These procedures require you to perform the following actions.

- Start the OSL server processes (`osl_server`). See ''Commands That Start the OSL Server Processes'' on page 3-17.
- Activate SOSL Net driver (`sosl_net_driver`). See ''The Command That Activates SOSL Net Driver'' on page 3-18.
- Start the network-watchdog process. See ''Commands That Start the Network-Watchdog Process'' on page 3-19.
- Edit the following `.tin` files, in order to create VOS configuration-table files.
  - `new_modules.tin`, in which you create module entries that specify the information required by OSL and STCP (for example, the IP address of a module's logical interface)—You should create one version of the file and use the `broadcast_file` command to broadcast it throughout the system. A sample `new_modules.tin` file based on Figure 3-1 appears in ''Sample `new_modules.tin` Files'' on page 3-10.
  - `disks.tin`—See ''Using the Disks Configuration-Table File'' on page 3-12.
  - `devices.tin`—See ''Using the Device Configuration-Table File'' on page 3-13.

## The Sample System %admin

Figure 3-1 illustrates the sample system %admin, which is a single-system OSL configuration. Figure 3-1 includes a list of the software components of the configuration. The system %admin consists of three modules: #m10, #m11, and #m12. Each module connects to two different LANs: subnetworks 134.111.5 and 134.111.6. Each module could contain either two Ethernet PCI adapters or four Ethernet PCI adapters configured into two SDLMUX groups (each SDLMUX group has one IP address). The two separate STCP connections enable fault-tolerant communication. Note that the IP addresses on your system will differ from the ones in Figure 3-1.




Each module in the system %admin includes:

- VOS software
- STCP software, including edited database files
- two physical interfaces to two IP networks/subnets or four physical interfaces configured in two SDLMUX groups to two IP network/subnets (each SDLMUX group has one IP address)
- two or more osl_server processes
- an osl_overseer process
- an osl_daemon process
- the communications driver, SOSL Net driver
- a network_watchdog process
- a new_modules.table file
- a disks.table file
- a devices.table file (with STCP device entries)

**Figure 3-1. A Single-System Configuration: Software Components**

# Updating the Master Disk Label

The label of a module's master disk contains specific information about the module. Each time the module is rebooted, VOS reads the master disk label for information about the current module. VOS uses the appropriate modules configuration-table file (for OSL, the `new_modules.table` file) on the current module for information about **other** modules in the system. However, because the same `new_modules.table` file resides on all modules in the system, the information you specify for a module in the `new_modules.table` file must match the information you specify in that module's master disk label. A discrepancy prevents other modules in the network from communicating with that module.

You can issue the `display_disk_label` command to check each module's master disk label. If you want to update the master disk label to change the module name, module number, or station number associated with a module (or the system name or system number used to identify the system to which the module belongs), follow the procedures in the manual *VOS System Administration: Disk and Tape Administration* (R284). (Note that the system name and system number must be the same for **all** modules within a single system.) These procedures include using the privileged `update_disk_label` command, which writes updated information to the label of the specified master disk.

The changes you make to the master disk label by issuing the `update_disk_label` command take effect the next time the module is booted. If you change a module's name, module number, or station number in the module's master disk label, you must also update the information in the `new_modules.table` file.

NOTE

Each module that is part of a single system must have a unique module name, module number, and station number.

For descriptions of the `display_disk_label` and `update_disk_label` commands, see the manual *VOS System Administration: Disk and Tape Administration* (R284).

# Using the New Modules Configuration-Table File

For OSL, a new modules configuration-table file (`new_modules.table`) enables VOS to recognize the modules in a system. In any system of VOS modules using OSL with STCP, each module in the system must have an entry in this file.

When creating the `new_modules.table` file, note the following:

- If you plan to designate one module in the system as a bridge module in order to handle multiple-system OSL communications (as described in "Planning a Multiple-System Configuration" on page 4-2), the module entry for the bridge module must include the information required for OSL.
- Even a single-module system requires the `new_modules.table` file. VOS uses the entry for the current module when processing any type of OSL information (for example, if OSL is used to handle multiple-system communications, as described in "Planning a Multiple-System Configuration" on page 4-2).
- Entries in the `new_modules.tin` file must **not** contain the `version` field. It is for Stratus-internal use only.

Like all other configuration-table files, you create the `new_modules.table` file by issuing the `create_table` command, which uses the following files as input.

- The `new_modules.dd` file is a data-description or format file that declares the template of records in the `new_modules.table` file. **Never** modify this file; you only display it to see which fields you must use in a record (that is, an entry) for a module.
- The `new_modules.tin` file is a table-input file that you edit to create an entry for each module in the system.

The `new_modules.table`, `new_modules.dd`, and `new_modules.tin` files reside in each module's `(master_disk)>system>configuration` directory. You typically edit the `new_modules.tin` file that resides in the directory `(master_disk)>system>configuration` of the designated master module and then re-create the `new_modules.table` file by issuing the `create_table` command, as described in "Creating, Installing, and Activating the `new_modules.table` File" on page 3-11.

The following sections provide additional information about the new modules configuration-table file.

## The new_modules.dd File

You use the fields defined in the new_modules.dd file when you create an entry for a module in the new_modules.tin file. You can display, but **never** modify, the new_modules.dd file.

NOTE

The version field indicates the version number, which is 1. Do **not** place the version field in your module entries. It is for Stratus-internal use only.

The new_modules.dd file contains the following information.

```
fields:   version             fixed bin (15),
          module_name         char (32) varying,
          module_number       fixed bin (15),
          station_number      fixed bin (15),
          stratalink_hw       fixed bin (15), /*default is 1*/
          open_socket_number  fixed bin (15),
          max_open_servers    fixed bin (15),
          base_port           fixed bin (15),
          hostname1           char (64) varying,
          hostname2           char (64) varying,
          hostname3           char (64) varying,
          hostname4           char (64) varying,
          hostname5           char (64) varying,
          hostname6           char (64) varying,
          hostname7           char (64) varying,
          hostname8           char (64) varying;
end;
```

## Creating Entries in the new_modules.tin File

To create an entry for a module in the new_modules.tin file, you specify information in the appropriate fields, as they appear in the new_modules.dd file.

You must include all required fields when creating an entry for a module in the new_modules.tin file, as identified in the following field descriptions. Use these descriptions to help you specify information in the fields.

► module_name **Required**

In this field, specify the name of a module in the system. Each module in a system must have a unique module name. The module name you specify must match the module name defined in the module's master disk label.

- `module_number` **Required**

  In this field, specify a number for the module identified in the `module_name` field. The value can be an integer from 1 to 32. You should assign module numbers sequentially, starting at 1. Each module in a system must have a unique module number. The module number you specify must match the module number defined in the module's master disk label.

- `station_number` **Required**

  In this field, specify a network station number for the module specified in the `module_name` field. A *station number* is an addressing mechanism that uniquely identifies each module in a single system. In a multimodule system, the station number of each module must be unique within the system. The station number you specify must match the station number defined in the master disk label of that module. Typically, the station number is the same as the module number. Although the value can be an integer from 1 to 127, a single system can contain no more than 32 modules.

  NOTE

  For communications across multiple systems, a station number identifies each bridge module that handles the incoming and outgoing requests associated with its local system. (See Chapter 4 for more information about configuring modules for multiple-system OSL communications.)

- `stratalink_hw` **Required**

  In this field, specify the value `0`, which indicates that the module does not have proprietary StrataLINK hardware.

- `open_socket_number` **Required**

  In this field, specify a nonzero value. If you have a `new_modules.tin` file from a previous VOS release, you can retain the `open_socket_number` values specified in that file.

- `max_open_servers` **Required**

  In this field, specify a value that determines the number of TCP ports available to the server side of SOSL Net driver. The TCP ports and SOSL Net driver are located on the module specified in the `module_name` field. The server side of SOSL Net driver listens on these TCP ports for client requests from SOSL Net driver on a remote module. The number of TCP ports is twice the value specified in the `max_open_servers` field. For example, the value 2 enables the server side of SOSL Net driver to listen on four TCP ports. Specify a value from `1` through `32`.

  For typical configurations, you specify the same value for the `max_open_servers` field in the `new_modules.tin` and

new_backbone_systems.tin files. If, however, you need to specify different values, the value for the max_open_servers field in the new_modules.tin file should be greater than (or equal to) the value specified for the new_backbone_systems.tin file.

On bridge modules, you typically start a number of osl_server processes that equals four times the value you specify for the max_open_servers field of the new_modules.tin file. On a non-bridge module, you typically start a number of osl_server processes that equals twice the value you specify for the max_open_servers field of the new_modules.tin file. "Commands That Start the OSL Server Processes" on page 3-17 provides information on starting osl_server processes.

► base_port **Required**

In this field, specify the lowest-numbered TCP port where the server side of SOSL Net driver listens for client requests. The TCP port and SOSL Net driver are located on the module specified in the module_name field. Stratus recommends that you specify the value 3000; you must specify a value in the range 3000 to 20,000.

NOTE

The value that you specify for the base_port field in the new_modules.tin file **must not** be in the range of port numbers determined by the value that you specify for the base_port field in the new_backbone_systems.tin file.

SOSL Net driver on the specified module uses this port number and the value specified in the max_open_servers field to establish a range of port numbers on which the server side of SOSL Net driver listens for client requests. The highest number in the range equals the following value.

base_port value + (max_open_servers value * 2) - 1

For example, when the max_open_servers field specifies the value 2 and the base_port field specifies the value 3000, SOSL Net driver on the specified module establishes TCP ports 3000 through 3003 for its server side to listen for client requests. With a max_open_servers value of 10, the range is 3000 to 3019. through

► `hostname1` through `hostname8` **Required**

In these fields, specify the IP addresses of the STCP interfaces that handle OSL communications on the specified module, as follows:

- If you do not configure the interfaces into SDLMUX groups, specify the IP address of each interface.
- If you configure two physical interfaces into one SDLMUX group, specify the IP address of the SDLMUX group.

You specify an IP address using standard dot notation, such as 134.111.5.10. To support dual communications routes among the modules, you must specify two IP addresses. The IP addresses represent two interfaces to two different networks or subnets. For example, `#m10` can provide two interfaces with the IP addresses 134.111.5.10 and 134.111.6.10.

## Sample `new_modules.tin` Files

The following example illustrates a sample `new_modules.tin` file for the three-module system `%admin`, which is illustrated in Figure 3-1. Each module in the system is configured to communicate using only OSL and STCP and to use common client/server values. These common values allow the modules to use the same `new_modules.tin` file. Note that each entry defines two IP addresses, since each module provides two physical interfaces and, therefore, supports two routes of communications.

```
/     =module_name         m10
      =module_number       10
      =station_number      10
      =stratalink_hw       0
      =open_socket_number  1
      =max_open_servers    2  /* m10 listens on 4 ports */
      =base_port           3000
      =hostname1           134.111.5.10
      =hostname2           134.111.6.10

/     =module_name         m11
      =module_number       11
      =station_number      11
      =stratalink_hw       0
      =open_socket_number  1
      =max_open_servers    2  /* m11 listens on 4 ports */
      =base_port           3000
      =hostname1           134.111.5.11
      =hostname2           134.111.6.11
```

*(Continued on next page)*

```
/     =module_name         m12
      =module_number       12
      =station_number      12
      =stratalink_hw       0
      =open_socket_number  1
      =max_open_servers    2  /* m12 listens on 4 ports */
      =base_port           3000
      =hostname1           134.111.5.12
      =hostname2           134.111.6.12
```

## Creating, Installing, and Activating the new_modules.table File

After you edit the new_modules.tin file, you issue the create_table command to create the new_modules.table file. This command uses the new_modules.dd and new_modules.tin files as input.

Each module in the system must have a new_modules.table file. All modules can have a copy of the same file, as long as the same client/server values apply to each module. You can create and install identical new_modules.table files on each module in the system, or you can modify the new_modules.table file on the master module and then issue the broadcast_file command to install this file on the other modules in the system. (You can also use the copy_file command.)

NOTE

You cannot successfully execute the broadcast_file or copy_file command unless network connections are operating among the modules in the system. When network connections are not operating among the modules in the system, you can transfer files using the ftp command (see the *VOS STREAMS TCP/IP User's Guide* (R421)).

To enable VOS to recognize the new new_modules.table file at each subsequent bootload, issue the configure_modules command from each module's module_start_up.cm file. To enable VOS to recognize the new new_modules.table file during the **current** bootload, issue the configure_modules command from command level on each module in the system. Note that the configure_modules command has been modified to accept either the new_modules.table file or the modules.table file (for compatibility). If you do not specify the table's path name, the command searches for the file (master_disk)>system>new_modules.table first, then it searches for the file (master_disk)>system>modules.table.

For a description of the configure_modules command, see Chapter 6.

The following sections provide task-specific information.

- For information about issuing commands such as `configure_modules` from the `module_start_up.cm` file, see "Modifying the `module_start_up.cm` File" on page 3-15.
- For a step-by-step description of how to add OSL to an operational system, see "Adding a Module to an OSL System" on page 3-24. Some of the steps involve creating, installing, and activating the `new_modules.table` file, as well as adding a single module to an existing system.
- For information about creating intermodule connectivity on a new system, see "Configuring and Starting a New OSL System" on page 3-20.
- For information about deleting a module, see "Deleting a Module from the System" on page 3-27.

For an overview of the steps involved in working with a configuration-table file, see "Standard VOS System Administration Procedures" on page 1-19. For detailed information about the procedures for creating and installing a configuration-table file and for descriptions of the `create_table` and `broadcast_file` commands, see the manual *VOS System Administration: Configuring a System* (R287). For information about the `copy_file` command, see the *VOS Commands Reference Manual* (R098).

## Using the Disks Configuration-Table File

The disks configuration-table file (`disks.table`) enables VOS to recognize the various disks in a system. Each disk (including removable disks) in a single system must be defined in the `disks.table` file.

Like all other configuration-table files, you edit the `disks.tin` file and then create the `disks.table` file by issuing the `create_table` command, which uses the following files as input.

- The `disks.dd` file is a data-description or format file that declares the template of records in the `disks.table` file. **Never** modify this file; you only display it to see which fields you must use in a record (that is, an entry) for a disk.
- The `disks.tin` file is a table-input file that you edit to create an entry for each disk in the system.

The `disks.table`, `disks.dd`, and `disks.tin` files reside in each module's `(master_disk)>system>configuration` directory. You typically edit the `disks.tin` file that resides in the `(master_disk)>system>configuration` directory of the designated master module. Since disk options can change with each VOS release, refer to the manual *VOS System Administration: Configuring a System* (R287) for information about creating entries for disks in the `disks.tin` file.

To create an entry for a disk, you specify information in the fields defined in the `disks.dd` file.

After you edit the `disks.tin` file, you can issue the `create_table` command to create a new `disks.table` file. The `create_table` command uses the `disks.dd` and `disks.tin` files as input.

Each module in the system must have a copy of the `disks.table` file. You can create and install identical `disks.table` files on each module in the system, or you can modify the `disks.table` file on the master module and then issue the `broadcast_file` command to install this file on the other modules in the system. (You can also use the `copy_file` command.)

NOTE

You cannot successfully execute the `broadcast_file` or `copy_file` command unless network connections are operating among the modules in the system. When network connections are not operating among the modules in the system, you can transfer files using the `ftp` command (see the *VOS STREAMS TCP/IP User's Guide* (R421)).

To enable VOS to permanently recognize the new `disks.table` file at each subsequent bootload, you must issue the `configure_disks` command from each module's `module_start_up.cm` file. To enable VOS to recognize the new `disks.table` file during the **current** bootload, you must issue the `configure_disks` command from command level on each module in the system.

For descriptions of the `configure_disks` and `broadcast_file` commands, see the manual *VOS System Administration: Configuring a System* (R287). For information about issuing commands from the `module_start_up.cm` file, see ''Modifying the `module_start_up.cm` File'' on page 3-15. For information about the `copy_file` command, see the *VOS Commands Reference Manual* (R098).

## Using the Device Configuration-Table File

The device configuration-table file (`devices.table`) enables VOS to recognize the devices in a system. Each device in a single system of Stratus modules, including the STCP devices required to run OSL, must be defined in the `devices.table` file.

The `devices.table` file recognizes a device as either a hardware or software entity that can be referenced and accessed by the system or its users. In a single-system configuration using OSL with STCP, the relevant devices are the STCP devices, which include the physical interfaces and logical interfaces that establish the STCP connections to the LANs. For more information about creating device entries for STCP

devices, see the *VOS STREAMS TCP/IP Administrator's Guide* (R419). In **addition** to the standard STCP devices, you must also configure devices unique to running OSL with STCP, as described in ''STCP Requirements'' on page 1-9. Terminals and printers that are directly connected to the modules are also defined as devices in the `devices.table` file.

Like all other configuration-table files, you edit the `devices.tin` file and then create the `devices.table` file by issuing the `create_table` command, which uses the following files as input.

- The `devices.dd` file is a data-description or format file that declares the template of records in the `devices.table` file. **Never** modify this file; you only display it to see which fields you must use in a record (that is, an entry) for a device (for example, the STCP physical interfaces). For a complete list of the fields in the `devices.dd` file, see the manual *VOS System Administration: Configuring a System* (R287).
- The `devices.tin` file is a table-input file that you edit to create an entry for each device on each module in the system.

The `devices.table`, `devices.dd`, and `devices.tin` files reside in each module's `(master_disk)>system>configuration` directory. You typically edit the `devices.tin` file that resides in the `(master_disk)>system>configuration` directory of the designated master module and then create a new `devices.table` file by issuing the `create_table` command.

After you edit the `devices.tin` file, you can issue the `create_table` command to create a new `devices.table` file. The `create_table` command uses the `devices.dd` and `devices.tin` files as input.

Each module in the system must have a copy of the `devices.table` file. You can create and install identical `devices.table` files on each module in the system, or you can modify the `devices.table` file on the master module and then issue the `broadcast_file` command to install this file on the other modules in the system. (You can also use the `copy_file` command.)

NOTE

You cannot successfully execute the `broadcast_file` or `copy_file` command unless network connections are operating among the modules in the system. When network connections are not operating among the modules in the system, you can transfer files using the `ftp` command (see the *VOS STREAMS TCP/IP User's Guide* (R421)).

To enable VOS to permanently recognize the new `devices.table` file at each subsequent bootload, you must issue the `configure_devices` command from each module's `module_start_up.cm` file.To enable VOS to recognize the new `devices.table` file during the **current** bootload, you must issue the `configure_devices` command from command level on each module in the system. (See the manual *VOS System Administration: Configuring a System* (R287) for a description of the `configure_devices` command.)

The following sections provide task-specific information.

- For information about issuing commands such as `configure_devices` from the `module_start_up.cm` file, see "Modifying the `module_start_up.cm` File" on page 3-15.
- For a step-by-step description of how to add OSL to an operational system, see "Adding a Module to an OSL System" on page 3-24. Some of the steps involve creating, installing, and activating the `devices.table` file, as well as adding a single module to an existing system.

For an overview of the steps involved in working with a configuration-table file, see "Standard VOS System Administration Procedures" on page 1-19. For detailed information about the procedures for creating and installing a configuration-table file, see the manual *VOS System Administration: Configuring a System* (R287).

## Modifying the `module_start_up.cm` File

To automatically configure the modules in a single-system configuration that uses OSL at each subsequent bootload, you must issue the following commands from each module's `module_start_up.cm` file.

- configuration commands (for example, `configure_devices` and `configure_modules`), as described in "Configuration Commands" on page 3-16
- commands that start the `osl_server` processes, as described in "Commands That Start the OSL Server Processes" on page 3-17
- the command that activates SOSL Net driver, as described in "The Command That Activates SOSL Net Driver" on page 3-18
- the command that starts the OSL daemon process, as described in "The Command That Starts the OSL Daemon Process" on page 3-18
- the command that starts the OSL overseer, as described in "The Command That Starts the OSL Overseer" on page 3-18
- commands that start the network-watchdog process, as described in "Commands That Start the Network-Watchdog Process" on page 3-19

- commands that configure the different components of STCP, as described in the following sections and manual:
  - ''Commands That Start STCP'' on page 3-19
  - ''STCP Requirements'' on page 1-9
  - *VOS STREAMS TCP/IP Administrator's Guide* (R419)

The preceding commands exist as comments in the `module_start_up.cm` file shipped with the release; that is, each command is preceded by an ampersand (&) and a space. To issue these commands from the `module_start_up.cm` file, you delete the ampersand and the space preceding each command and specify the arguments required by the command. Note that you must issue some commands multiple times to meet the configuration's needs. Note also that you must issue all of these commands from command level to start a single-system configuration during the current bootload. (When you enter a command at command level and the command line extends beyond the screen width for the terminal, the command wraps to the next line and the second line begins with a continuation character, the plus (+) sign.)

NOTE

Since OSL relies on STCP, the `module_start_up.cm` file must execute the `start_stcp` command macro, with its series of STCP commands, **before** it issues commands to configure OSL. For a description of the STCP configuration procedures, see ''STCP Requirements'' on page 1-9. For information on the `start_stcp` command macro, see the *VOS STREAMS TCP/IP Administrator's Guide* (R419).

## Configuration Commands

For a module permanently configured for OSL, the `module_start_up.cm` file must execute the following configuration commands to enable VOS to use the configuration-table files and communications protocols, such as protocols associated with STCP (for example, TELNET). If you want VOS to recognize the configuration-table files and the appropriate communications protocols during the current bootload, you must issue these commands from command level.

- `configure_modules`
- `configure_disks`
- `configure_devices`

NOTE

In order for the configure_devices command to configure the OSL multiplexor, you must add an entry for it to the devices.tin file before the module_start_up.cm file executes. For information on the OSL multiplexor entry, see "Adding the OSL Multiplexor to the devices.tin File" on page 1-10.

- configure_comm_protocol

The configure_modules, configure_disks, and configure_devices commands activate the new_modules.table, disks.table, and devices.table files, respectively. These configuration commands have no required arguments; therefore, you can simply delete the ampersand and the space preceding each command. For information about the configuration commands, see the manual *VOS System Administration: Configuring a System* (R287). For information about the communications protocols associated with STCP components, see the *VOS STREAMS TCP/IP Administrator's Guide* (R419).

## Commands That Start the OSL Server Processes

The module_start_up.cm file contains commands that establish the conditions for proper operation of the osl_server processes and then start the processes. The commands include osl_admin requests that determine if the module is a bridge module (and, thus, the osl_server command requires the -super argument) as well as the number of required osl_server processes.

In a basic configuration of a non-bridge module, the module_start_up.cm file typically starts a number of osl_server processes that equals twice the value of the max_open_servers field in the new_modules.tin file. On bridge modules, the module_start_up.cm file typically starts a number of osl_server processes that equals four times the value of the max_open_servers field in the new_modules.tin file. For complete information on the new_modules.tin file, see "Creating Entries in the new_modules.tin File" on page 3-7.

See the sample_module_start_up.*new_release_no* file in the >system>*release* directory on your module for the command lines that start OSL, including OSL server processes. To locate these command lines, search for OSL.

To enable the appropriate commands, delete the ampersand and the space preceding each set of commands in the module_start_up.cm file. Note that the osl_server processes are actually **started** within start_process command lines. See the *VOS Commands Reference Manual* (R098) for a description of the start_process command and the commands that establish the conditions for proper operation of a process (for example, the set_implicit_locking command).

If you need to start additional osl_server processes on the module, set a value for the ADDITIONAL_OSL_SERVERS variable in the module_start_up.cm file. The value you set for the ADDITIONAL_OSL_SERVERS variable should equal the number of additional servers that you need.

To start osl_server processes during the current bootload, issue osl_server commands from command level. For a complete description of the osl_server command, see Chapter 6.

## The Command That Activates SOSL Net Driver

You must uncomment the following command, which enables the module_start_up.cm file to activate SOSL Net driver (sosl_net_driver). To uncomment a command, delete the ampersand and the space preceding it.

```
configure_comm_protocol sosl_net_driver
```

For a complete description of the configure_comm_protocol command, see the manual *VOS System Administration: Configuring a System* (R287).

## The Command That Starts the OSL Daemon Process

The module_start_up.cm file must start the OSL daemon process. Uncomment the following command line in the module_start_up.cm file, substituting ***MODULE*** with the module name.

```
start_process 'osl_daemon #stcp.MODULE #stcp_osl_mux.MODULE'
     -privileged -priority 7
```

For additional information, see ‘‘Starting the OSL Daemon Process’’ on page 1-11.

## The Command That Starts the OSL Overseer

The module_start_up.cm file must start the OSL overseer process (osl_overseer). Uncomment the following command lines in the module_start_up.cm file:

```
!delete_file osl_overseer.out.old -brief
!rename osl_overseer.out osl_overseer.out.old
!create_file osl_overseer.out
!set_implicit_locking osl_overseer.out
!start_process -privileged 'osl_overseer' -process_name
     OSL_Overseer
```

## Commands That Start the Network-Watchdog Process

The *network watchdog* is a system process that periodically performs the following functions.

- It determines if other modules in the system have either lost or re-established communications with the module on which the network watchdog is executing.
- It closes connections left open when other, previously attached modules have lost communications with the module on which the network watchdog is executing.

You start the network watchdog by specifying the `network_watchdog` command within a `start_process` command line in the `module_start_up.cm` file. A network-watchdog process must be executing on each module in the system in order for the system to operate correctly. For a detailed description of the `network_watchdog` command, see the manual *VOS System Administration: Administering and Customizing a System* (R281). See the *VOS Commands Reference Manual* (R098) for a description of the `start_process` command and the commands that establish the conditions for proper operation of the network-watchdog process (for example, the `set_implicit_locking` command).

The `module_start_up.cm` file should already contain one set of commands that establishes the conditions for proper operation of a network-watchdog process and then starts the process. To execute the command lines, delete the ampersand and the space preceding each command line in the `module_start_up.cm` file.

The following example illustrates how this set of commands initially appears in the `module_start_up.cm` file.

```
& delete_file network_watchdog.out.old -brief
& rename network_watchdog.out network_watchdog.out.old
& create_file network_watchdog.out
& set_implicit_locking network_watchdog.out
& start_process network_watchdog -priority 7 -privileged &+
          -process_name NetworkWatchdog
```

## Commands That Start STCP

For your module to run STCP, the `module_start_up.cm` file must include command lines to perform the following actions, which are part of the standard STCP startup procedures.

- Add the library paths of the STCP command library.
- Define the STCP host name for the module by issuing a `hostname` command.
- Invoke the `start_stcp.cm` command macro.

For complete information about the commands in the `module_start_up.cm` file needed to start STCP, see the *VOS STREAMS TCP/IP Administrator's Guide* (R419).

In addition to the standard STCP startup procedures, the `module_start_up.cm` file must also include the commands described in ''STCP Requirements'' on page 1-9.

## Registering Users on the System

As the final step in creating a single-system configuration using OSL, you must create the system's registration databases. In these databases, you add information about each system user. The registration database files, `user_registration.sysdb` and `change_password.sysdb`, contain information about the users who can log in to the system. Issue the `create_user_sysdbs` command to create these files and their VOS links. You must issue this command once for each module on the system, designating which module will serve as the master module. The master module saves the master copies of the registration database files in the directory `(master_disk)>system>configuration` and is responsible for collecting and broadcasting password changes. After you create the registration databases, issue the `registration_admin` command to add information about each system user, including the user's person name, alias, password, home directory, and process priorities.

To restrict user access to a module, use the `login_admin` command. For detailed descriptions of this command and the `create_user_sysdbs` command, see the manual *VOS System Administration: Registration and Security* (R283).

## Configuring and Starting a New OSL System

This section describes procedures that let you configure modules in a new system using OSL and STCP and to start OSL communications. These procedures assume that the modules currently are not connected. You can start OSL communications between the modules in a new system in two ways.

- You can reboot each module immediately to enable communications with the other modules in the system. The changes that establish permanent operation of the network take effect immediately.
- You can issue the appropriate VOS commands to start the single-system configuration during the current bootload, then make the changes that will establish permanent operation of the network whenever you reboot the system (that is, the `module_start_up.cm` file must also issue the commands).

The second method lets you start the system without interrupting a module's current operation, thus minimizing the number of times a module is rebooted. This method also lets you issue the `broadcast_file` command **after** you establish active network connections between the modules in the system. (You cannot execute this command

successfully until network connections are operating between the modules.) Using the `broadcast_file` command lets you maintain a single version of each configuration-table file on one module's master disk (the principal master disk) and copy each configuration-table file to the other modules in the system. For information on the principal master disk, see the manual *VOS System Administration: Administering and Customizing a System* (R281).

This section describes how to start a new system using OSL and STCP during the current bootload. Follow these procedures if you want to avoid rebooting the modules in the system. Note that these startup procedures encompass the configuration procedures described earlier in this chapter, such as the creation of configuration-table files and the definition of commands that establish various network components. As you perform the startup procedures, assume that a system using OSL is not already operating. To perform these procedures, you must be a privileged user with modify access to the `(master_disk)>system` directory.

If you need to reboot the modules, follow the procedures described in the manual *VOS System Administration: Starting Up and Shutting Down a Module or System* (R282).

To start a new system that uses OSL during the current bootload, perform the following steps.

1. Configure each module in the system for STCP. Each module must have the hardware and software required to run STCP, and each module must provide two physical interfaces to different networks or subnets to enable dual routes of communications among the modules. Follow the procedures described in "STCP Requirements" on page 1-9, as well as the standard STCP configuration procedures.
2. Issue the `display_disk_label` command to check that the system name and system number recorded on the master disks of each module match, and that no two master disks have the same module name, module number, or station number.
3. When each module was booted, the `module_start_up.cm` file must have issued all of the pertinent configuration-table file commands as well as the commands to configure STCP. If these commands were not issued, you must issue them now manually.
4. Designate one module to be the master module. The master copies of the system's configuration-table files are located on this module's master disk; this disk is the principal master disk.
5. On the master module, add an entry to the `new_modules.tin` file for each of the other modules in the system. Also, add an entry to the `disks.tin` file for the master disk of each of the other modules.
6. On all other modules in the system, add an entry to the `new_modules.tin` file for the master module. Also, add an entry to the `disks.tin` file for the master module's master disk.

7. On all modules, issue the `create_table` command to re-create the `new_modules.table` and `disks.table` files, and copy these files to the directory `(master_disk)>system`.
8. On all modules, issue the `configure_modules` and `configure_disks` commands.
9. On all modules, issue the commands to start the OSL server processes and to activate SOSL Net driver. To start the server processes, issue `start_process` commands such as the following:

   ```
   start_process 'osl_server -syserr -server_suffix 1' -priority 9 -privileged
   ```

   On a non-bridge module, the number of `osl_server` processes typically should equal twice the value you specify for the `max_open_servers` field of the `new_modules.tin` file. On bridge modules, the number of `osl_server` processes typically should equal four times the value you specify for the `max_open_servers` field of the `new_modules.tin` file. "Using the New Modules Configuration-Table File" on page 3-6 provides information on the `max_open_servers` field.

   You must also activate SOSL Net driver (`sosl_net_driver`) by issuing the following command.

   ```
   configure_comm_protocol sosl_net_driver
   ```

   The `module_start_up.cm` file shipped with each module contains the text of these commands. Note that the `start_process` command lines are part of a larger set of commands that establishes the conditions for the proper operation of the server and network-watchdog processes; you must issue all of the commands in the set to establish the proper networking environment. At this point, the master module should be able to communicate with each of the other modules in the system, but the other modules cannot yet communicate with each other.

   You must also add the OSL multiplexor to the `devices.tin` file, as described in "Adding the OSL Multiplexor to the `devices.tin` File" on page 1-10, and start the OSL daemon process. To start the OSL daemon process, issue the following command.

   ```
   start_process 'osl_daemon #stcp.MODULE #stcp_osl_mux.MODULE'
        -privileged -priority 7
   ```

The following command line starts the OSL daemon process on module `m1`; therefore, the name of the STCP protocol driver is `stcp.m1` and the name of the OSL multiplexor is `stcp_osl_mux.m1`.

```
start_process 'osl_daemon #stcp.m1 #stcp_osl_mux.m1'
     -privileged -priority 7
```

10. On the master module, add entries to the `devices.tin` file for each device defined on the other modules in the system. You should be able to access each module's `devices.tin` file directly from the master module, which will facilitate adding these entries.
11. If the other modules in the system have disks other than their master disks, add entries for these additional disks to the `disks.tin` file on the master module.
12. On the master module, issue the `create_table` command to re-create the `devices.table` and `disks.table` files, and copy the newly created files to the `(master_disk)>system` directory.
13. From the `(master_disk)>system` directory of the master module, issue the `broadcast_file` command to broadcast the `new_modules.table`, `devices.table`, and `disks.table` files to all other modules in the system.
14. On all modules, issue the commands `configure_modules`, `configure_disks`, and `configure_devices` to establish full communications between all modules.
15. Issue the `create_user_sysdbs` command to create the registration databases, which contain all of the information about the system users. Then, add each system user to the system by issuing the `registration_admin` command.
16. You may choose to maintain a common copy of the `module_start_up.cm` file for all modules in the system to use. If so, edit the `module_start_up.cm` file on the master module as necessary to incorporate the other modules' startup requirements. For example, you need to add the proper commands to configure the physical interfaces on the other modules and supply them with their proper IP addresses.

For information about executing the `configure_devices` and `configure_disks` commands, see the manual *VOS System Administration: Configuring a System* (R287). To verify that the commands worked correctly, issue commands such as `list_modules` and `display_disk_info`. These commands are documented in the *VOS Commands Reference Manual* (R098).

# Modifying Existing Systems

The following sections describe how to modify existing systems in certain ways.



## Adding a Module to an OSL System

The following sections describe how to add a module to an OSL system.



Note that these procedures assume that all of the existing modules in the system are already running OSL.

### Adding a Module Permanently

To add a new module permanently to a single-system configuration using OSL with STCP and to have VOS recognize the module during the current bootload, perform the following steps.

1. Issue the `display_disk_label` command to check that the system name and system number recorded on the new module’s disk match their respective values on the disk of the existing system’s master module, and that the module name, module number, and station number recorded on the new module’s disk do not duplicate those of any other module in the existing system.
2. When the new module was booted, its `module_start_up.cm` file must have issued all of the pertinent configuration-table file commands as well as the commands to configure STCP. If these commands were not issued, you must issue them now, manually. For information about how to configure STCP, see the *VOS STREAMS TCP/IP Administrator’s Guide* (R419), as well as ‘‘STCP Requirements’’ on page 1-9.
3. On the existing system’s master module, add an entry to the `new_modules.tin` file for the new module. Also, add an entry to the `disks.tin` file for the master disk of the new module.
4. On the new module, add an entry to the `new_modules.tin` file for the existing system’s master module. Also, add an entry to the `disks.tin` file for the master disk of the existing system’s master module.
5. On both the master module and the new module, issue the `create_table` command to re-create the `new_modules.table` and `disks.table` files. Copy these files to the `(master_disk)>system` directory on both modules.
6. On both modules, issue the `configure_modules` and `configure_disks` commands.

7. If OSL server processes are not yet started and SOSL Net driver has not yet been activated on the new module, you must issue commands to start the server processes and to activate the driver for the current bootload. To start the server processes, issue `start_process` commands such as the following:

   ```
   start_process 'osl_server -syserr -server_suffix 1' -priority 9 -privileged
   ```

   In general, the number of `osl_server` processes to start on a non-bridge module should equal twice the value you specify for the `max_open_servers` field of the `new_modules.tin` file. On a bridge module, the number of `osl_server` processes should be four times the value you specify for the `max_open_servers` field of the `new_modules.tin` file. "Using the New Modules Configuration-Table File" on page 3-6 provides information on the `max_open_servers` field.

   You must also activate SOSL Net driver (`sosl_net_driver`) by issuing the following command.

   ```
   configure_comm_protocol sosl_net_driver
   ```

   You must also add the OSL multiplexor to the `devices.tin` file, as described in "Adding the OSL Multiplexor to the `devices.tin` File" on page 1-10, and start the OSL daemon process. To start the OSL daemon process, issue the following command.

   ```
   start_process 'osl_daemon #stcp.MODULE #stcp_osl_mux.MODULE'
        -privileged -priority 7
   ```

   The following command line starts the OSL daemon process on module `m1`; therefore, the name of the STCP protocol driver is `stcp.m1`, and the name of the OSL multiplexor is `stcp_osl_mux.m1`.

   ```
   start_process 'osl_daemon #stcp.m1 #stcp_osl_mux.m1'
        -privileged -priority 7
   ```

   The `module_start_up.cm` file shipped with the new module contains the text of these commands. Note that the `start_process` command lines are part of a larger set of commands that establishes the conditions for the proper operation of the server and network-watchdog processes; you must issue all commands in the set to establish the proper networking environment. At this point, networking should become operational between these two modules **only**.

8. On the master module, add entries to the `devices.tin` file for each device defined on the new module. You should be able to access the new module's `devices.tin` file directly from the master module, which will facilitate adding these entries.
9. If the new module has disks other than its master disk, add entries for these additional disks to the `disks.tin` file on the master module.

10. On the master module, issue the `create_table` command to re-create the `devices.table` and `disks.table` files, and copy the newly created files to the `(master_disk)>system` directory.
11. From the `(master_disk)>system` directory of the master module, issue the `broadcast_file` command to broadcast the `new_modules.table`, `devices.table`, and `disks.table` files to all other modules in the system, including the new module.
12. On all modules, including the master module and the new module, issue the commands `configure_modules`, `configure_disks`, and `configure_devices`. This establishes full communications among all modules.
13. If your site maintains a common copy of `module_start_up.cm` to be used by all modules in the system, edit the `module_start_up.cm` file on the master module as necessary to incorporate the new module's startup requirements. For example, you need to add the proper commands to configure the STCP physical interfaces on the new module and supply them with their proper IP addresses.

### Adding a Module Temporarily

There are two situations in which you will want changes to take effect during the current bootload: after you install new configuration-table files, and when you want to make **temporary** changes that are not reflected in the configuration-table files or the `module_start_up.cm` file. Two commands let you add a module for the duration of the current bootload only.

- `add_module`—If you want VOS on the current module to recognize an additional module for the duration of the current bootload only, issue the `add_module` command. This command enables VOS to recognize a module that does not have an entry in the `new_modules.table` file. You can also use this command to restore a particular module to service.

  NOTE

  The `add_module` command also lets you change the current module's configuration during the current bootload. For a description of the `add_module` command, see Chapter 6.

- `configure_modules`—This command configures all modules defined in the `new_modules.table` file (or in the older `modules.table` file) that have not previously been defined. The `module_start_up.cm` file contains this command in order to allow VOS to access the appropriate modules configuration-table file automatically at each bootload.

## Deleting a Module from the System

The following sections describe how to delete a module from a system.


- "Deleting a Module Permanently" on page 3-27
- "Deleting a Module Temporarily" on page 3-28


Note that these procedures assume that the existing modules in the system are already running OSL.

### Deleting a Module Permanently

To delete a module permanently from a single-system configuration using OSL with STCP, perform the following steps.

1. On the master module, delete the module's entry in the `new_modules.tin` file.
2. On the master module, issue the `create_table` command to create the new `new_modules.table` file. Then, copy the file to the `(master_disk)>system` directory.
3. On the master module, update the `disks.tin` and `devices.tin` files so that they no longer reflect the components in the module being deleted. Create new `disks.table` and `devices.table` files with the `create_table` command. Then, copy the files to the `(master_disk)>system` directory. Issue the commands `configure_disks` and `configure_devices -flush` on all modules to enable VOS to recognize immediately the new `disks.table` and `devices.table` files.
4. Issue the command `osl_admin -disable_destination` *`module_name`* to disable the module from the OSL network.
5. On the master module, issue either the `delete_module` command or the `configure_modules -reset` command so that the master module no longer recognizes the module as part of its system.

   **CAUTION**

   Using the `-reset` argument with the commands `configure_systems` and `configure_modules` may interrupt cross-system and cross-module communications.

6. On the master module, issue the `broadcast_file` command to copy the new `new_modules.table` file to each module in the system.
7. On each module in the system, issue the `configure_modules -reset` command to enable VOS to recognize the updated `new_modules.table` file.
8. In the `module_start_up.cm` file of the module to be deleted, comment out all of the commands that start the `osl_server` processes and to activate

`osl_net_driver`. To comment out these commands, insert an ampersand and a space at the beginning of each of these command lines. "Commands That Start the OSL Server Processes" on page 3-17 shows command lines as comments (that is, the commands are commented out). If you are using STCP, you also need to delete the commands required to run OSL with STCP, as described in "STCP Requirements" on page 1-9.

9. On the module to be deleted, issue `stop_process` commands from command level to stop the server processes. Also, stop the network-watchdog process.

### Deleting a Module Temporarily

To delete a module for the duration of the current bootload only (that is, temporarily), issue the `delete_module` command. After you issue the `delete_module` command, VOS no longer recognizes the specified module.

**CAUTION**

Before issuing the `delete_module` command, you must notify users who are currently using that module.

For a description of the `delete_module` command, see Chapter 6.

Your site may have other administrative or configuration issues regarding the deletion of a new module. If so, you should address these issues now.

The following manuals describe how to make other configuration changes.

- For an explanation of how to **rename a module** in a single system, see the discussion about changing a module or system identifier (or site ID) in the manual *VOS System Administration: Administering and Customizing a System* (R281).
- For an explanation of how to **change a module number or station number** in a single system, see the discussion about changing a module or system identifier (or site ID) in the manual *VOS System Administration: Administering and Customizing a System* (R281).
- For an explanation of how to **change the disk configuration**, see the discussion about configuring disks in the manual *VOS System Administration: Configuring a System* (R287).

# Chapter 4
# Configuring a Multiple-System OSL Network

This chapter describes the software configuration procedures to create, start, and modify a multiple-system OSL network. In this type of configuration, multiple systems use OSL and STCP to communicate with each other over a local area (using Ethernet) or a wide area. (The WAN functionality of OSL is sometimes called Open StrataNET.) Multiple-system OSL configurations over a local area do not require additional hardware to handle cross-system communications.

This chapter contains the following sections.



‘‘Checklist: A Multiple-System OSL Network’’ on page 2-2 provides a checklist for creating a multiple-system configuration. ‘‘Standard VOS System Administration Procedures’’ on page 1-19 provides an overview of how to work with configuration-table files and the `module_start_up.cm` file.

# Planning a Multiple-System Configuration

After presenting general information on planning a multiple-system configuration, this section provides the following additional sections, which contain information related to planning this configuration:



You can begin to plan a multiple-system configuration by deciding which systems will be in the configuration. The systems you connect can be composed of either a single module or multiple modules. When planning the configuration, consider the following:

- Decide how OSL cross-system communications fit into your overall STCP network configuration. Typically, you need to work with your network administrator to determine this configuration. For STCP configuration, see the *VOS STREAMS TCP/IP Administrator’s Guide* (R419). Additional requirements specific to running OSL with STCP are described in ‘‘STCP Requirements’’ on page 1-9. You must follow the procedures described in this section in addition to the standard STCP configuration procedures in order to run OSL with STCP. (For information about migrating from OS TCP/IP to STCP, see the *VOS STREAMS TCP/IP Migration Guide* (R418).)
- Decide on unique system names and system numbers for each system (if the systems you plan to connect do not already have unique names and numbers). You will use this information in two systems configuration-table files, which are described in ‘‘Using the New Systems Configuration-Table File’’ on page 4-6 and ‘‘Using the New Backbone Systems Configuration-Table File’’ on page 4-12. If you have not already done so, you also need to assign module numbers and station numbers in the new modules configuration-table file (`new_modules.table`), as described in Chapter 3.
- Decide which **one** module in each system will serve as the bridge module. (In the case of a single-module system, that single module is the bridge module.) As with other modules in a multiple-system OSL configuration, the bridge module must be configured for multiple-system OSL and STCP. A bridge module also has the following special requirements.
  - A bridge module typically has outbound/inbound OSL servers. Outbound/inbound OSL servers forward outbound requests from nonbridge modules on the local system to other bridge modules on remote systems. They also receive inbound requests from other bridge modules on remote systems for nonbridge modules on the local system. You create outbound/inbound OSL servers by specifying the `-super` argument of the `osl_server` command when you start the OSL server processes. If you specify the `-super` argument for one `osl_server` command, you must specify it for **all** `osl_server` commands that you issue on the same module. All OSL servers on one module must be of the same type.

– If possible, assign each bridge module a unique station number; that is, the station-number value you define for a bridge module in the `station_number` field of the `new_modules.table` file ideally should not be defined for any other bridge module. (The station-number value defined for a bridge module must always match the value defined in that bridge module's master disk label.) The bridge modules require the file `new_backbone_systems.table`, as described in "Using the New Backbone Systems Configuration-Table File" on page 4-12. (Nonbridge modules require an additional configuration-table file, the `new_systems.table` file.)

NOTE

Each system can have only **one** bridge module.

You may want to draw diagrams of your planned configuration and refer to them as you plan and perform the configuration procedures.

## A Multiple-System Configuration

Multiple-system OSL enables a system to communicate, using STCP, with up to 254 other systems over a local area or a wide area. In a configuration using only multiple-system OSL for all cross-system communications, each module in each system requires an additional configuration-table file, the `network_access.table` file.

Each system must also have a bridge module that is configured for STCP and multiple-system OSL. The bridge module handles all incoming and outgoing requests for its local system. (All other modules in each system are nonbridge modules.) For OSL, each system's bridge module requires the following:

- SOSL Net driver (`sosl_net_driver`)
- the network-watchdog process (`network_watchdog`)
- outbound/inbound OSL servers (`osl_server -super`)—Note that all OSL servers on one module must be of the same type. When you create one outbound/inbound OSL server on a module, all of the OSL servers on the module must be outbound/inbound.
- the configuration-table file, `new_backbone_systems.table` (nonbridge modules must have a `new_systems.table` file)

To create a multiple-system OSL configuration, you must determine how the systems fit into your overall STCP network configuration and plan your routes accordingly (for information on STCP routes, see "Configuring Routes in STCP" on page 1-9). Typically, you need to work with your network administrator to plan this configuration. If a system is already configured for single-system OSL and STCP, the modules in each system should already provide two STCP logical interfaces and have two routes

of communications. You can use the same logical interfaces and routes for cross-system communications, since OSL does not require the creation of separate backbone connections on each bridge module. Your STCP network configuration may feature a corporate backbone network that connects multiple systems residing on different subnets, but this type of configuration is optional and might not require two additional logical interfaces on the bridge module.

To ensure that your multiple-system configuration provides fault tolerance, you must create dual routes of communications between the systems. For information on OSL and fault tolerance, see ‘‘OSL Configurations and Fault Tolerance’’ on page 1-7.

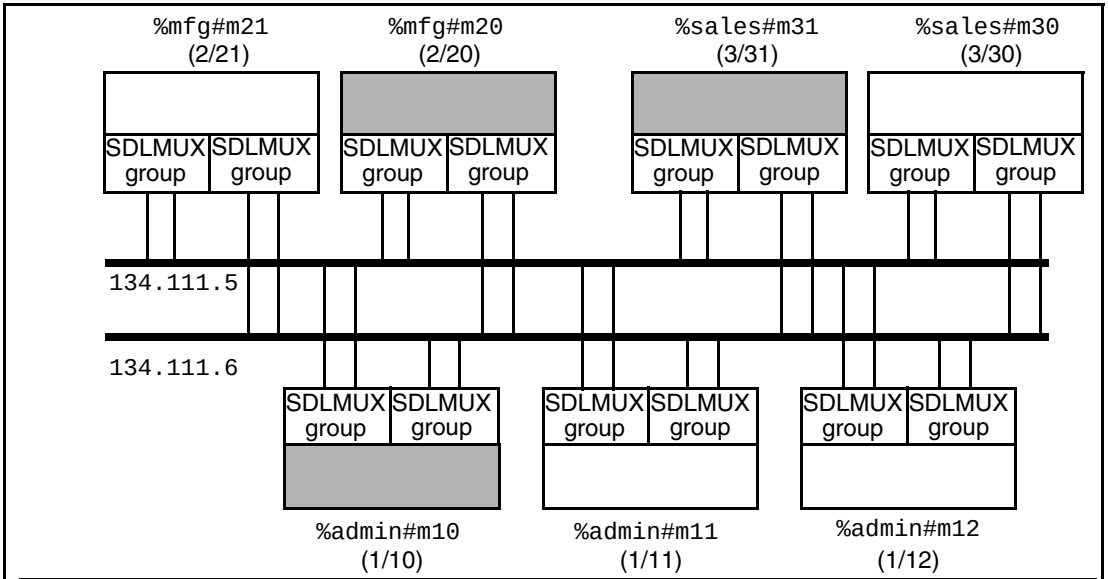
If you already have two STCP logical interfaces but want to establish additional STCP logical interfaces and routes to handle cross-systems communications, you must issue STCP commands such as `route` and ensure that the STCP database files contain the appropriate information. You must then define the complete IP addresses associated with the interfaces on each bridge module in the `new_backbone_systems.tin` file (see ‘‘Using the New Backbone Systems Configuration-Table File’’ on page 4-12).

NOTES

1. OSL does not support fully qualified host names.
2. You can configure up to eight logical interfaces on a module to handle OSL communications with STCP. For a description of the STCP configuration procedures, see the *VOS STREAMS TCP/IP Administrator’s Guide* (R419).
3. A bridge module handling multiple-system OSL **must** be defined for OSL in the `new_modules.table` file. To do this, you must specify the availability of OSL in the bridge module’s entry in the `new_modules.tin` file (see ‘‘Using the New Modules Configuration-Table File’’ on page 3-6).

Figure 4-1 shows a sample configuration using multiple-system OSL as the cross-system communications scheme. This configuration consists of three systems (`%admin`, `%mfg`, and `%sales`), each of which also uses OSL to handle cross-module communications within the system. (Note that Figure 3-1 illustrates the configuration of the system `%admin` for OSL only.) In this configuration, the interfaces configured for cross-module communications also handle the cross-system communications (that is, all traffic goes over the 134.111.5 and 134.111.6 subnets).

Figure 4-1 lists the software components that you need to provide on the bridge and nonbridge modules in each system in the network. The *`number/number`* format (for example, 2/20) represents the system number and the station number associated with a module.




Each module in each system has the following:

- STCP software, including edited database files
- four STCP physical interfaces configured into two SDLMUX groups for connections among modules/systems
- SOSL Net driver
- `osl_server` processes (on bridge modules, all are `osl_server -super`)
- a `network_watchdog` process
- a `new_systems.table` file (nonbridge modules only)
- a `new_backbone_systems.table` file (bridge modules only)
- a `network_access.table` file, a `new_modules.table` file, a `disks.table` file, and a `devices.table` file

= Bridge Module = Nonbridge Module

(*`number/number`*) = System Number/Station Number

**Figure 4-1. A Multiple-System Configuration: Software Components**

For information on the network access configuration-table file (`network_access.table`), see "Using the Network Access Configuration-Table File" on page 4-21. For information on the disks configuration-table file (`disks.table`), see "Using the Disks Configuration-Table File" on page 3-12.

### Driver and Process Functions on a Bridge Module

To handle the requests sent to and received from the other modules in its own system, a bridge module supporting single-system OSL requires SOSL Net driver and the OSL server processes. The appropriate single-system processes should already be configured on the bridge module, as described in Chapter 3.

To handle the requests sent to and received from other systems, a bridge module supporting multiple-system OSL requires SOSL Net driver and the OSL server processes. The following driver and the types of processes run on a bridge module:

- `osl_server -super`—This process executes inbound and outbound requests that require user-level processing.
- `sosl_net_driver`— SOSL Net driver sends outbound requests to, and receives inbound requests from, the bridge modules of remote systems with which it communicates using STCP.

## Using the New Systems Configuration-Table File

After introducing the new systems configuration-table file, this section provides the following additional sections, which provide information related to this file.


- "The `new_systems.dd` File" on page 4-7
- "Creating Entries in `new_systems.tin` Files" on page 4-8
- "Sample `new_systems.tin` Files" on page 4-10
- "Creating, Installing, and Activating the `new_systems.table` Files" on page 4-11


For a description of the `new_backbone_systems.table` file, see "Using the New Backbone Systems Configuration-Table File" on page 4-12.

The new systems configuration-table file (`new_systems.table`) and the new backbone systems configuration-table file (`new_backbone_systems.table`) enable VOS to recognize a multiple-system configuration that relies on OSL for all or some of the cross-system communications. Each multimodule system using multiple-system OSL must be defined in the `new_systems.table` and `new_backbone_systems.table` files.

The `new_systems.table` file must reside on each system's **nonbridge** modules and is unique to each system. The `new_systems.table` file can also reside on the system's designated bridge module, although the bridge module accesses the `new_systems.table` file only if it cannot find a `new_backbone_systems.table` (or the older `backbone_systems.table`) file. (For information on the `new_backbone_systems.table` file, see "Using the New Backbone Systems Configuration-Table File" on page 4-12.)

NOTE

Each system can have only **one** bridge module.

The nonbridge modules of each system use the `new_systems.table` file. This file defines all systems that can be accessed through the local bridge module using multiple-system OSL.

The `new_systems.table` file for a particular system can reside on all modules within that system; each module has an identical copy of this file. Note that a single-module system does **not** typically have a `new_systems.table` file, since the system's single module is the bridge module and typically uses a `new_backbone_systems.table` file instead of a `new_systems.table` file.

Like all other configuration-table files, you create the `new_systems.table` file by issuing the `create_table` command, which uses the following files as input.

- The `new_systems.dd` file is a data-description or format file that declares the template of records in the `new_systems.table` file. You **never** modify this file; you only display it to see which fields you must use in a record (that is, an entry) for a system.
- The `new_systems.tin` file is a table-input file that you edit to create an entry for each system. You must edit the `new_systems.tin` file on each system to include an entry for each system. In each entry, you must identify the bridge module of the system being configured (the current system).

The `new_systems.table`, `new_systems.dd`, and `new_systems.tin` files reside in the `(master_disk)>system>configuration` directory. You typically edit the `new_systems.tin` file that resides in the `(master_disk)>system>configuration` directory on the master module of each system and then create a new `new_systems.table` file by issuing the `create_table` command, as described in "Creating, Installing, and Activating the `new_systems.table` Files" on page 4-11.

## The `new_systems.dd` File

You use the fields defined in the `new_systems.dd` file when you create an entry for a system in the `new_systems.tin` file. You can display, but **never** modify, the `new_systems.dd` file.

NOTES

1. Although the `new_systems.dd` file and the `new_backbone_systems.dd` file define the same fields, you use only a small subset of the fields when you create an entry in the `new_systems.tin` file

(unlike the new_backbone_systems.tin file, which uses a larger subset of the fields).

2. The version field indicates the current version number, which is 1. Do **not** place the version field in your system entries.

The new_systems.dd file defines the following fields.

```
fields:  version           fixed bin (15), /* default is 1 */
         system_name       char (32) varying,
         system_number     fixed bin (15),
         station_number    fixed bin (15),
         socket_number     fixed bin (15),
         funnel_name       char (32) varying,
         max_open_servers  fixed bin (15), /* default is 0 */
         base_port         fixed bin (15), /* default is 0 */
         hostname1         char (64) varying,
         hostname2         char (64) varying,
         hostname3         char (64) varying,
         hostname4         char (64) varying,
         hostname5         char (64) varying,
         hostname6         char (64) varying,
         hostname7         char (64) varying,
         hostname8         char (64) varying;
end;
```

## Creating Entries in new_systems.tin Files

To create an entry for a system in each system's new_systems.tin file, you specify information in a small subset of the fields defined in the new_systems.dd file. Note that the system_name, system_number, station_number, and socket_number fields are the relevant fields in the new_systems.tin file and are the only fields described in this section. Although the older systems.tin file may include the funnel_name field, this field is not necessary in the new_systems.tin file. For a description of all of the fields, see "Creating Entries in the new_backbone_systems.tin File" on page 4-15.

Within each system entry, the values for two of the fields (system_name and system_number) must be the same in all new_systems.tin files in the multiple-system configuration. Typically, the values for the other fields are customized for each system. For example, to enable the system you are configuring (the current system) to communicate with the other systems, the station_number field in each system entry must identify the current system's bridge module.

The current system requires its own entry in the new_systems.table file created from the new_systems.tin file; therefore, you **mus**t include an entry for the current

system to order for the system to configure its own system information. The values you specify for system names and system numbers in the `new_systems.tin` file must match the corresponding values in the master disk label of each module in the system. To change values in the master disk label of a module, issue the `update_disk_label` command from a privileged process, as described in ‘‘Updating the Master Disk Label’’ on page 3-5.

To help you specify information in the fields, use the following descriptions.

- `system_name` **Required**
  In this field, specify the name of a system in the multiple-system configuration. Each system must have a unique system name (and a unique `new_systems.table` file), and each system must have an entry in each `new_systems.table` file. When specifying the system name, do not include the percent sign (%) prefix.

- `system_number` **Required**
  In this field, specify the number of the system specified in the `system_name` field. The number must be an integer from 1 through 255; Stratus recommends assigning system numbers sequentially starting with 1. Each system number must be unique within the network.

- `station_number` **Required**
  In this field, specify the station number of the bridge module that resides in the current system. This bridge module provides network access from the current system to the system specified in the `system_name` field. You specify the station number of the bridge module in the current system because outbound requests are always routed through the current system’s bridge module. The station number you specify must be defined in the `station_number` field of the bridge module’s `new_modules.table` file (as described in ‘‘Using the New Modules Configuration-Table File’’ on page 3-6). If possible, each bridge module in a multiple-system OSL configuration should have a unique station number. The value for this field must be an integer from 1 through 127.

NOTE

Entries in the `new_systems.table` file identify the station number of the bridge module in the current system, regardless of the system specified in the `system_name` field. However, the `new_backbone_systems.table` file identifies the station number based on the cross-system communications scheme. For information on the `station_number` field of the `new_backbone_systems.table` file, see ‘‘Creating

Entries in the new_backbone_systems.tin File" on page 4-15.

- `socket_number` **Required**
  In this field, specify a nonzero value of 1 through 100. Note that OSL does not use the value.

## Sample `new_systems.tin` Files

Using the multiple-system OSL configuration illustrated in Figure 4-1, the following example shows `new_systems.tin` files for three systems, beginning with the system `%admin`. Each system's `new_systems.tin` file resides on the modules in that system. (Note that in the `system_name` field, you do not include the percent sign (%) prefix. Note also that each file has comments that identify the module of the system on which the file resides.)

```
/* new_systems.tin file for %admin, installed on #m11 and #m12 */

   /    =system_name        admin
        =system_number      1
        =station_number     10  /* Bridge module for %admin */
        =socket_number      31

   /    =system_name        mfg
        =system_number      2
        =station_number     10
        =socket_number      31

   /    =system_name        sales
        =system_number      3
        =station_number     10
        =socket_number      31

   /* new_systems.tin file for %mfg, installed on #m21 */

   /    =system_name        admin
        =system_number      1
        =station_number     20   /* Bridge module for %mfg */
        =socket_number      31
```

*(Continued on next page)*

```
/    =system_name        mfg
     =system_number      2
     =station_number     20
     =socket_number      31

/    =system_name        sales
     =system_number      3
     =station_number     20
     =socket_number      31

/* new_systems.tin file for %sales, installed on #m31 */

/    =system_name        admin
     =system_number      1
     =station_number     30 /* Bridge module for %sales */
     =socket_number      31

/    =system_name        mfg
     =system_number      2
     =station_number     30
     =socket_number      31

/    =system_name        sales
     =system_number      3
     =station_number     30
     =socket_number      31
```

## Creating, Installing, and Activating the `new_systems.table` Files

After you edit the `new_systems.tin` file for each system, issue the `create_table` command to create a new `new_systems.table` file for each system. This command uses the `new_systems.dd` and `new_systems.tin` files as input.

Each nonbridge module in a system must have an identical copy of the `new_systems.table` file; the bridge module can also have a copy of the file, although it is used only if VOS cannot find a `new_backbone_systems.table` (or the older `backbone_systems.table`) file. Issue the `broadcast_file` command to install the updated table on the modules in each system, including the current module. This step assumes that you have already configured each of your systems as described in Chapter 3, and that connections are operating between the modules in the current system. Note that the `new_systems.table` file for each system is unique to that system.

To enable VOS to recognize the `new_systems.table` file at each subsequent bootload, issue the `configure_systems` command from the

module_start_up.cm file of each nonbridge module in each system. To enable VOS to recognize the new_systems.table file during the **current** bootload, issue the configure_systems command from command level on each nonbridge module in each system. (See Chapter 6 for a description of the configure_systems command.)

You can test the installation of a new_systems.table file by issuing the list_systems command, which is documented in the *VOS Commands Reference Manual* (R098). This command displays the names of all of the systems defined in the new_systems.table file and indicates, by the designations online or offline, whether the current module can access each system. If you specify the argument -long, the command displays additional information about those systems designated as offline.

For task-specific information, see the following sections.

- "Modifying the module_start_up.cm File" on page 4-29 describes how to issue commands such as configure_systems from the module_start_up.cm file. (See Chapter 6 for a description of the configure_systems command.)
- "Starting Multiple-System OSL Communications" on page 4-32 provides a step-by-step description of how to start cross-system OSL communications. Some of the steps involve creating, installing, and activating the new_systems.table file.
- "Adding a System to a Multiple-System Configuration" on page 4-35 and "Deleting a System from a Multiple-System Configuration" on page 4-37 describe how to add or delete a system.

For an overview of the steps involved in working with a configuration-table file, see "Standard VOS System Administration Procedures" on page 1-19. For detailed information about the procedures for creating and installing a configuration-table file and for descriptions of the create_table and broadcast_file commands, see the manual *VOS System Administration: Configuring a System* (R287).

## Using the New Backbone Systems Configuration-Table File

After introducing the new backbone systems configuration-table file, this section provides the following additional sections that contain information related to this file.

- "The new_backbone_systems.dd File" on page 4-14
- "Creating Entries in the new_backbone_systems.tin File" on page 4-15
- "Sample new_backbone_systems.tin Files" on page 4-17
- "Creating, Installing, and Activating the new_backbone_systems.table File" on page 4-20

For systems that communicate using multiple-system OSL, the new backbone systems configuration-table file (`new_backbone_systems.table`) defines the systems from the perspective of the local system's bridge module. In essence, it defines how a system's bridge module communicates with other remote systems using STCP. Therefore, it is unique to that system.

The `new_backbone_systems.table` file must reside on each system's **bridge** module and is also unique to each system if it contains entries for OSL. The new file applies to any bridge module that handles cross-system communications for its local system (for example, an OSL bridge module).

NOTE

The erroneous existence of the `new_backbone_systems.table` file on a **nonbridge** module causes the `configure_systems` command to generate the error `e$bridge_config_file` (5151) when it tries to process the system information. See ''Configuration Error Messages'' on page 5-3 for more information about this error.

Like all other configuration-table files, you create the `new_backbone_systems.table` file by issuing the `create_table` command, which uses the following files as input.

- The `new_backbone_systems.dd` file is a data-description or format file that declares the template of records in the `new_backbone_systems.table` file. You **never** modify this file; you only display it to see which fields you must use in a record (that is, an entry) for a system.
- The `new_backbone_systems.tin` file is a table-input file that you edit to create an entry for each system. You must edit the `new_backbone_systems.tin` file to include an entry for each system in the multiple-system configuration. For a system that is accessed using multiple-system OSL, the system entry must identify the remote system, the local system's bridge module, and the TCP/IP information needed to communicate with the remote system.

The files `new_backbone_systems.table`, `new_backbone_systems.dd`, and `new_backbone_systems.tin` reside in the `(master_disk)>system>configuration` directory. You edit the `new_backbone_systems.tin` file that resides in the `(master_disk)>system>configuration` directory of each system's bridge module and then create a new `new_backbone_systems.table` file on each system by issuing the `create_table` command, as described in ''Creating, Installing, and Activating the `new_backbone_systems.table` File'' on page 4-20.

## The new_backbone_systems.dd File

You use the fields defined in the new_backbone_systems.dd file when you create an entry for a system in the new_backbone_systems.tin file. You can display, but **never** modify, the new_backbone_systems.dd file.

NOTES

1. Although the new_backbone_systems.dd and new_systems.dd files define the same fields, you use a larger subset of the fields when you create an entry in the new_backbone_systems.tin file.
2. The version field indicates the current version number, which is 1. Do **not** place the version field in your system entries.

The new_backbone_systems.dd file defines the following fields.

```
fields:   version           fixed bin (15),
          system_name       char (32) varying,
          system_number     fixed bin (15),
          station_number    fixed bin (15),
          socket_number     fixed bin (15),
          funnel_name       char (32) varying,
          max_open_servers  fixed bin (15), /* default is 0 */
          base_port         fixed bin (15),
          hostname1         char (64) varying,
          hostname2         char (64) varying,
          hostname3         char (64) varying,
          hostname4         char (64) varying,
          hostname5         char (64) varying,
          hostname6         char (64) varying,
          hostname7         char (64) varying,
          hostname8         char (64) varying;
end;
```

## Creating Entries in the new_backbone_systems.tin File

To create an entry for a system in the new_backbone_systems.tin file, you specify information using the fields defined in the new_backbone_systems.dd file. Note that multiple-system OSL requires the max_open_servers, base_port, and hostname***n*** fields. All entries require the system_name, system_number, and station_number fields. The funnel_name field is optional.

To help you specify information in the fields, use the following descriptions.

- system_name **Required**
  In this field, specify the name of a system in the multiple-system configuration. Each system must have a unique system name and an entry in the new_backbone_systems.table file. When specifying the system name, do not include the percent sign (%) prefix.

- system_number **Required**
  In this field, specify the number of the system identified in the system_name field. The number must be an integer from 1 through 255; Stratus recommends assigning system numbers sequentially starting with 1. Each system number must be unique within the network.

- station_number **Required**
  In this field, specify the station number of the bridge module that provides network access to the system specified in the system_name field. The station_number value must be an integer from 1 through 127. For OSL, specify the station number of the local system's bridge module (which should be the current module). The station number you specify must be defined in the station_number field of the appropriate new_modules.table file (as described in "Using the New Modules Configuration-Table File" on page 3-6). For bridge modules communicating using multiple-system OSL, unique station numbers are recommended but not required.

- socket_number **Required**
  In this field, specify a nonzero value of 1 through 100. Note that OSL does not actually use the value.

- funnel_name
  If the system specified in the system_name field must be accessed indirectly through another system, use this field to specify the system name of the intermediary (funnel) system. For OSL, this field applies to **any** system that must rely on an intermediary system to reach a remote system. If you specify a value for the funnel_name field in a system entry, do not specify a value for any hostname***n*** field; the funnel_name field and the hostname***n*** fields are mutually exclusive.

- `max_open_servers` **Required**

  In this field, specify a value that determines the number of TCP ports available to the server side of SOSL Net driver. The TCP ports and SOSL Net driver are located on the bridge module of the system specified in the `system_name` field. The server side of SOSL Net driver listens on these TCP ports; the client side of SOSL Net driver on the bridge module specified in the `station_number` field connects to these TCP ports. Specify a value from `1` through `32`.

  For typical configurations, you specify the same value for the `max_open_servers` field in the `new_modules.tin` and `new_backbone_systems.tin` files. If, however, you need to specify different values, the value for the `max_open_servers` field in the `new_modules.tin` file should be greater than (or equal to) the value specified for the `new_backbone_systems.tin` file.

  On a bridge module, you typically start a number of `osl_server` processes that equals four times the value you specify for the `max_open_servers` field in the `new_backbone_systems.tin` file. (For information on specifying values for the `new_modules.tin` file, see "Creating Entries in the `new_modules.tin` File" on page 3-7.)

  NOTE

  To start an `osl_server` process on a bridge module, you must issue the `osl_server` command with the `-super` argument.

  The number of TCP ports specified in `max_open_servers`, when used in a multiple-system OSL network that uses only RPC ports, will not double the number of TCP ports. However, a single-system OSL network allows for both RPC and Direct Queue Ports (or Fast Queue Ports (FQP)); therefore, the number of ports specified are doubled.

- `base_port` **Required**

  In this field, specify the lowest-numbered TCP port where the server side of SOSL Net driver provides wide-area communications. (The WAN functionality of OSL is sometimes called Open StrataNET.) The TCP ports and SOSL Net driver are located on the bridge module of the system specified in the `system_name` field. Stratus recommends that you specify the value `4000`; you must specify a value in the range 3000 to 20,000.

  NOTE

  The value that you specify for the `base_port` field in the `new_backbone_systems.tin` file **must not** be in the range of port numbers determined by the value that you

specify for the `base_port` field in the `new_modules.tin` file.

SOSL Net driver on the specified system's bridge module uses this port number and the value specified in the `max_open_servers` field to establish a range of port numbers on which to listen for requests. The highest number in the range equals the following value.

`base_port` value + `max_open_servers` value - 1

For example, if the `max_open_servers` field specifies the value 2, the `base_port` value `4000` tells SOSL Net driver on the specified system's bridge module that the server side listens for requests on TCP ports 4000 and 4001; the client side of SOSL Net driver on the bridge module specified in the `station_number` field can connect to these ports. With a `max_open_servers` value of `10`, the range is 4000 to 4009.

- `hostname1` through `hostname8` **Required**

  In these fields, specify the IP addresses of the STCP logical interfaces that handle multiple-system OSL communications on the specified system's bridge module. Specify the IP address of each logical interface. You specify an IP address using standard dot notation, such as 134.111.5.20. To support dual communications routes among the systems, you must specify at least two IP addresses. The IP addresses represent two logical interfaces to two different networks or subnets. For example, if the current system is `%admin`, the entry for `%mfg` might specify that the bridge module for `%mfg` (`%mfg#m20`) provides two logical interfaces with the IP addresses 134.111.5.20 and 134.111.6.20.

  NOTE

  OSL does not support fully qualified host names.

## Sample `new_backbone_systems.tin` Files

Using the configuration illustrated in Figure 4-1, the example in this section shows three `new_backbone_systems.tin` files (one for each system). Note that the comments in each file identify the bridge module of the current system. Note also that in the `system_name` field, you do not include the percent sign (%) prefix.

```
/* new_backbone_systems.tin file on the bridge module in %admin */

   /    =system_name        admin
        =system_number      1
        =station_number     10           /* %admin bridge */
        =socket_number      31
        =max_open_servers   2
        =base_port          4000
        =hostname1          134.111.5.10
        =hostname2          134.111.6.10

   /    =system_name        mfg
        =system_number      2
        =station_number     10
        =socket_number      31
        =max_open_servers   2
        =base_port          4000
        =hostname1          134.111.5.20 /* interface on %mfg */
        =hostname2          134.111.6.20

   /    =system_name        sales
        =system_number      3
        =station_number     10
        =socket_number      31
        =max_open_servers   2
        =base_port          4000
        =hostname1          134.111.5.30 /* interface on %sales */
        =hostname2          134.111.6.30

/* new_backbone_systems.tin file on the bridge module in %mfg */

   /    =system_name        admin
        =system_number      1
        =station_number     20              /* %mfg bridge */
        =socket_number      31
        =max_open_servers   2
        =base_port          4000
        =hostname1          134.111.5.10 /* interface on %admin */
        =hostname2          134.111.6.10
```

*(Continued on next page)*

```
   /    =system_name        mfg
        =system_number      2
        =station_number     20
        =socket_number      31
        =max_open_servers   2
        =base_port          4000
        =hostname1          134.111.5.20
        =hostname2          134.111.6.20

   /    =system_name        sales
        =system_number      3
        =station_number     20
        =socket_number      31
        =max_open_servers   2
        =base_port          4000
        =hostname1          134.111.5.30 /* interface on %sales */
        =hostname2          134.111.6.30

/* new_backbone_systems.tin file on the bridge module in %sales */

   /    =system_name        admin
        =system_number      1
        =station_number     30            /* %sales bridge */
        =socket_number      31
        =max_open_servers   2
        =base_port          4000
        =hostname1          134.111.5.10 /* interface on %admin */
        =hostname2          134.111.6.10

   /    =system_name        mfg
        =system_number      2
        =station_number     30
        =socket_number      31
        =max_open_servers   2
        =base_port          4000
        =hostname1          134.111.5.20 /* interface on %mfg */
        =hostname2          134.111.6.20
```

*(Continued on next page)*

```
/     =system_name         sales
      =system_number       3
      =station_number      30
      =socket_number       31
      =max_open_servers    2
      =base_port           4000
      =hostname1           134.111.5.30
      =hostname2           134.111.6.30
```

## Creating, Installing, and Activating the new_backbone_systems.table File

After you edit the new_backbone_systems.tin file to define the systems in the configuration, issue the create_table command to create a new new_backbone_systems.table file for each bridge module. For a configuration that includes multiple-system OSL, each bridge module requires its own unique new_backbone_systems.table file. The create_table command uses the new_backbone_systems.dd and new_backbone_systems.tin files as input. After you create the table, you can use the copy_file (or broadcast_file) command to copy it to the (master_disk)>system directory.

To enable VOS to recognize the new_backbone_systems.table file at each subsequent bootload, issue the configure_systems command from the module_start_up.cm file of each bridge module. To enable VOS to recognize the new_backbone_systems.table file during the **current** bootload, issue the configure_systems command from command level on each bridge module. (See Chapter 6 for a description of the configure_systems command.)

For task-specific information, see the following sections.

- "Modifying the module_start_up.cm File" on page 4-29 provides information about issuing commands such as configure_systems from the module_start_up.cm file. (See Chapter 6 for a description of the configure_systems command.)
- "Starting Multiple-System OSL Communications" on page 4-32 provides a step-by-step description of how to start cross-system OSL communications. Some of the steps involve creating, installing, and activating a new_backbone_systems.table file.
- "Adding a System to a Multiple-System Configuration" on page 4-35 and "Deleting a System from a Multiple-System Configuration" on page 4-37 provide information about adding or deleting a system.

For an overview of the steps involved in working with a configuration-table file, see "Standard VOS System Administration Procedures" on page 1-19. For detailed information about the procedures for creating and installing a configuration-table file

and for descriptions of the `create_table` and `broadcast_file` commands, see the manual *VOS System Administration: Configuring a System* (R287).

## Using the Network Access Configuration-Table File

After introducing the network access configuration-table file, this section provides the following additional sections, which contain related information.



In the course of configuring your systems, you must establish network access among the systems. To establish network access, you must create a network access configuration-table file (`network_access.table`) for each system and install the table on each module in the system. This configuration-table file lets you define the access rights of other systems in the network configuration to the current system.

You must provide the following access-rights information in a `network_access.table` file.

- client systems—Define the remote systems that can gain access to the files and other resources on the current system; these systems are *client systems* of the current system. Only systems defined in the `network_access.table` file on the current system are client systems. Therefore, if a system is **not** defined in the `network_access.table` file on the current system, users on that remote system cannot gain access to the resources on the current system. (The current system should also be defined as a client system of the remote systems, thereby making all systems in the network clients of each other.)
- registration requirements—Define the registration requirements for users of the client systems who want to gain access to the resources on the current system. If you require registration, you must register the users of the client systems as *client users* in the current system's registration database. For information about the registration database, see the manual *VOS System Administration: Registration and Security* (R283).
- password requirements—Define the password requirements for client users who want to gain access to the resources on the current system. If you require a

password, the client users logged in to a client system must provide a password for the current system before gaining access to the current system's resources.

Like all other configuration-table files, you create the `network_access.table` file by issuing the `create_table` command, which uses the following files as input.

- The `network_access.dd` file is a data-description or format file that declares the template of records in the `network_access.table` file. You **never** modify this file; you only display it to see which fields you must use in a record (that is, an entry) for a system.
- The `network_access.tin` file is a table-input file that you edit to create an entry for each system. To define access to the current system, you must edit the current system's `network_access.tin` file to include an entry for each system that will be a client system of the current system.

The `network_access.table`, `network_access.dd`, and `network_access.tin` files reside in the `(master_disk)>system>configuration` directory. You typically edit the `network_access.tin` file that resides in the `(master_disk)>system>configuration` directory of the master module of each system and then create a new `network_access.table` file by issuing the `create_table` command. Then, you install the newly created table file on all modules in the system. The procedures for creating and installing the `network_access.table` file are described in "Creating, Installing, and Activating the `network_access.table` File" on page 4-25.

## The `network_access.dd` File

You use the fields defined in the `network_access.dd` file when you create an entry for a system in the `network_client.tin` file. You can display, but **never** modify, the `network_access.dd` file.

The `network_access.dd` file contains the following information.

```
organization:  sequential;
index: system no_duplicates;

fields: system            char (32),
        registration_req  fixed bin (15),
        password_req      fixed bin (15);

end;
```

## Creating Entries in the network_access.tin File

To create an entry for a system in a system's network_access.tin file, you specify information in the fields defined in the network_access.dd file. Note that the system field is required; you cannot omit a required field when creating an entry.

To help you specify information in the fields, use the following descriptions.

- system **Required**
  In this field, specify the name of a system in the network that is a client of the current system. The specified system must be defined in the new_backbone_systems.table and new_systems.table files on the current system. When specifying the system name, do not include the percent sign (%) prefix.

- registration_req
  In this field, specify a value that defines the registration requirements for a user of the client system. The value 1 indicates that, to access the current system, a user of the client system must be registered in the current system's registration database. The default value 0 indicates that registration is not required.

- password_req
  In this field, specify a value that defines the password requirements for a user of the client system. The value 1 indicates that, to access the current system, a user of a client system must provide a password. This requirement is ignored if the user's entry in the current system's registration database does not contain a password. The default value 0 indicates that a password is not required.

NOTE

If the registration_req value is 0, the password_req value must be 0.

## Levels of Access Control

The values you specify in the registration_req and password_req fields produce one of three levels of access control in a client system entry. Table 4-1 shows the three levels of access control and the fields used to define it in a client system entry. Level 1 is the least restrictive control; level 3 is the most restrictive. Note that the value 0 in a field indicates that registration and/or passwords are not required; the value 1 in a field indicates that registration and/or passwords **are** required.

**Table 4-1. Levels of Access Control for Remote Client Users**

| Fields | Control Levels and Corresponding Field Values | | |
|---|---|---|---|
| | **Level 1:** Registration not required; password not required | **Level 2:** Registration required; password not required | **Level 3:** Registration and password required |
| `system`<br>`registration_req`<br>`password_req` | ***name***<br>`0`<br>`0` | ***name***<br>`1`<br>`0` | ***name***<br>`1`<br>`1` |

The following list describes each level of access control.

- Level 1—Users registered on the remote client system specified in the `system` field can access the current system without being registered in the current system's registration database and without specifying passwords.
- Level 2—Users registered on the remote client system specified in the `system` field must also be registered in the current system's registration database in order to access the current system. However, the client users are not required to specify passwords.
- Level 3—Users registered on the remote client system specified in the `system` field must also be registered in the current system's registration database and must specify passwords (if the user entries in the current system's registration database have password values). If the entries do not have password values, the current system will not require the users to supply passwords.

  Before accessing the current system for the first time, a user who is already logged in to the remote client system and who has level-3 access to the current system **must** issue the `verify_system_access` command. The `verify_system_access` command, which is documented in the *VOS Commands Reference Manual* (R098), checks the user's access privileges, such as whether the user can start a privileged process. If the user does not specify a password when issuing the command (and the registration database has a password value), VOS prompts the user to specify one. Once system access has been verified, the user's access remains in effect until the user logs out of the remote client system.

## A Sample network_access.tin File

The network_access.tin file in the following example illustrates two levels of access control. If the network_access.table file generated from this file is installed on all modules in the three systems shown in Figure 4-1, each system will be defined as a client of the other systems in the network, with level-1 control for users on the system %admin and level-3 control for users on the systems %mfg and %sales. This means that users on the system %admin require neither registration nor a password to access %mfg and %sales, but users on the systems %mfg and %sales require both registration and a password to access %admin and each other. Note that in the system field, you do not include the percent sign (%) prefix.

```
/* Access control for client systems */

/    =system                  admin
     =registration_req        0
     =password_req            0

/    =system                  mfg
     =registration_req        1
     =password_req            1

/    =system                  sales
     =registration_req        1
     =password_req            1
```

## Creating, Installing, and Activating the network_access.table File

After creating entries for client systems in the network_access.tin file, issue the create_table command to create a new network_access.table file. Then, issue the copy_file (or broadcast_file) command to install the configuration-table file in the (master_disk)>system directory of all modules in the system. Each module in the system must have a network_access.table file. This configuration-table file can vary from system to system, depending on the level of control you want to assign to each system, or it can be identical on all systems. Until you establish network access between the systems, you cannot access files or other resources across the network. The network_access.table file takes effect immediately upon installation; therefore, no configuration command is required.

To determine how the creation of the network_access.table file fits into the startup procedures, see ‘‘Starting Multiple-System OSL Communications’’ on page 4-32. For an overview of the steps involved in working with a configuration-table file, see ‘‘Standard VOS System Administration Procedures’’ on page 1-19. For detailed information about the procedures for creating and installing a configuration-table file and for descriptions of the create_table and broadcast_file commands, see the manual *VOS System Administration: Configuring a System* (R287).

## Registering Client Users

As stated earlier, a user who gains access to the resources on the current system while logged in to a client system is a client user of the current system. If the `network_access.table` file specifies either level-2 or level-3 access control for a client system, you must register the users of the client system as client users in the current system's registration database. If the `network_access.table` file specifies level-3 access control for the client system, each user entry in the current system's registration database has a password value. If a user entry does not have a password value, the current system will not require the client user to provide a password.

If the current system's `network_access.table` file specifies level-2 or level-3 access control for the client system and if security logging is enabled for a client system in the network, a user on that client system who does not have the corresponding access privileges and who issues the `list_systems` command will cause a message such as the following to be written to the `security_log` file on the remote system.

```
1: 95-03-23 02:25:59 edt  PreLogin.System %mfg
   Target: verify_system_access
   Text: You are not a registered user of the target system.
```

On the user's screen, the output from the `list_systems` command will show the current system as offline. For more information about security logging and registering users in the registration database, see the manual *VOS System Administration: Registration and Security* (R283). For a description of the `list_systems` command, see the *VOS Commands Reference Manual* (R098).

## Logging In Remotely

The following sections provide information about logging into a remote system.

- "Direct Login" on page 4-26 describes how to directly log in to a remote system from a prelogin process on the current system.
- "Indirect Login" on page 4-28 describes how to indirectly log in to a remote system after logging in to the current system.

### Direct Login

*Direct login* enables a user on the current system to log in directly to the remote system without first logging in to the current system. A user issues the `login` command with the `-module` argument from the prelogin process on the current system. The user must specify the full names of the remote system and the module. To display a list of modules before logging in, the user can issue the `list_modules` command with either the *`module_name`* argument or the `-all` argument. The *`module_name`* argument can specify the name of a system and produce a list of modules within that system; the `-all` argument lists the modules for **all** systems that the user can access

from the current system. For a detailed description of the `list_modules` command, see the *VOS Commands Reference Manual* (R098).

The level of access control defined in the remote system's `network_access.table` file determines the client user's registration and password requirements on the current system and the remote system. For example, suppose that two users, `Smith` and `Clark`, are connected to the system `%sales` (their current system) and want to perform a direct login to `%mfg`. Both users are registered on both systems (which are illustrated in the sample configuration in Figure 4-1). The following access-control elements apply to these users.

- The system entry in the `network_access.tin` file on `%mfg` defines the access-control level for the system `%sales` users when they log in to the system `%mfg`.
- The registration requirements are specified for the users `Smith` and `Clark` in the registration databases on both systems.
- The `login` commands and password prompts appear on the screen when each user logs in directly from a terminal on the system `%sales` to the system `%mfg`.

The system entry for the system `%sales` in the `network_access.tin` file on the system `%mfg` defines level-3 access control for `%sales`, as the following example illustrates.

```
/    =system            sales
     =registration_req   1
     =password_req       1
```

Table 4-2 illustrates sample registration data specified for the users `Smith` and `Clark` on the systems `%sales` and `%mfg`.

**Table 4-2. Sample Registration Data**

| Registration Database Fields | System `%sales` | System `%mfg` |
|---|---|---|
| `Name`<br>`Groups`<br>`Password` | `Smith`<br>`sales`<br>`myword` | `Smith`<br>`mfg`<br>`myword` |
| `Name`<br>`Groups`<br>`Password` | `Clark`<br>`sales`<br>`pword_1` | `Clark`<br>`mfg`<br>`pword_2` |

The following example illustrates the `login` command and passwords that the user `Smith` specifies for direct login from the current system, `%sales`, to module `#m20` on

the remote system, %mfg. Here, the user Smith's password is the same on both systems (myword).

```
login  Smith  -module %mfg#m20
Password? myword
Local Password? myword
```

In an actual session, the passwords shown here would not appear on the screen.

The following example illustrates the login command and passwords that the user Clark specifies for direct login from the current system, %sales, to module #m20 on the remote system, %mfg. Here, the user Clark's password is unique on each system. In response to the Password? prompt, the user Clark specifies the password for the remote system (pword_2); in response to the Local Password? prompt, the user Clark specifies the password for the current (local) system (pword_1).

```
login  Clark  -module %mfg#m20
Password? pword_2
Local Password? pword_1
```

In an actual session, the passwords shown here would not appear on the screen. For a description of the login command, see the *VOS Commands Reference Manual* (R098).

### Indirect Login

*Indirect login* enables a user who has already logged in to the current system to log in to the remote system. After logging in to the current system, the user issues the login command with the -module argument from command level.

The level of access control defined in the remote system's network_access.table file determines whether the user must issue the verify_system_access command before issuing the login command. (The remote system should also be defined in the current system's network_access.table file.) If the remote system requires a password, the verify_system_access command prompts the user to specify a password. Once the remote system has verified the user's access, that access remains in effect until the user logs off the current system. Note that if a special session is in progress on the remote system, a user must specify a special-session password. (This is an uncommon occurrence.) For information about special-session passwords, see the manual *VOS System Administration: Registration and Security* (R283).

After logging in to the remote system indirectly, the user's initial current directory remains on the local system. This method can be very inefficient, because VOS searches the current directory each time the user issues a command. To improve response time, the user should either change the current directory to a directory on the remote system or at least avoid using a large current directory on the local system.

For more information about the commands `login`, `list_modules`, and `verify_system_access`, see the *VOS Commands Reference Manual* (R098).

## Modifying the `module_start_up.cm` File

To automatically configure the systems in your configuration for multiple-system OSL and STCP at each subsequent bootload, you must issue the following commands from the `module_start_up.cm` file of each system's bridge module and/or nonbridge modules.

- the `configure_systems` command (on each bridge and nonbridge module), as described in "The `configure_systems` Command" on page 4-30, in addition to the configuration commands required on modules in single-system configurations (see "Configuration Commands" on page 3-16)
- the commands that start the `osl_server` processes, as described in "Commands That Start the OSL Server Processes" on page 4-30
- the command that activates SOSL Net driver, as described in "The Command That Activates SOSL Net Driver" on page 4-31
- the command that starts the OSL daemon process, as described in "The Command That Starts the OSL Daemon Process" on page 4-32
- the command that starts the OSL overseer, as described in "The Command That Starts the OSL Overseer" on page 4-32
- the STCP commands—If the system is already configured with STCP, you may need to configure additional interfaces to handle cross-system communications on each bridge module. If the system is not configured with STCP, you must configure STCP on each bridge module that will communicate using multiple-system OSL by issuing several commands from the `module_start_up.cm` file. For information about the STCP commands, see "STCP Requirements" on page 1-9, "Commands That Start STCP" on page 3-19, and the *VOS STREAMS TCP/IP Administrator's Guide* (R419).

The preceding commands exist as comments in the `module_start_up.cm` file shipped with the release; that is, each command is preceded by an ampersand (`&`) and a space. To issue these commands from the `module_start_up.cm` file, you delete the ampersand and the space preceding each command and specify the arguments required by the command. Note that you must issue some commands multiple times to meet the configuration's needs. Note also that you can issue all of these commands from command level to start a multiple-system configuration during the current bootload. For more information, see "Starting Multiple-System OSL Communications" on page 4-32.

## The configure_systems Command

The bridge modules and nonbridge modules in a multiple-system configuration using OSL and STCP must execute the configure_systems command during module startup to enable VOS to use the appropriate systems file (the new_backbone_systems.table file or the new_systems.table file). Bridge modules require the new_backbone_systems.table file; nonbridge modules require the new_systems.table file.

The *system_file* argument of the configure_systems command lets you specify the appropriate systems configuration-table file. If you do not specify a value for the *system_file* argument, VOS automatically searches for the new_backbone_systems.table file in the (master_disk)>system directory. If the file exists, VOS defines it. If the file does not exist, VOS searches first for the backbone_systems.table, then for the new_systems.table file, and finally for the systems.table file.

NOTE

If the new_backbone_systems.table file exists on a nonbridge module, the configure_systems command generates the error e$bridge_config_file (5151), which states that this file belongs **only** on the bridge module. The command then abandons the current file and searches for the next configuration-table file.

The configure_systems command should already be uncommented and should execute automatically as part of the module_start_up.cm file.

## Commands That Start the OSL Server Processes

The module_start_up.cm file contains commands that let you establish the conditions for the proper operation of the osl_server processes and then start the processes. The commands include delete_file, create_file, set_implicit_locking, start_process, and osl_server. To start osl_server processes during the current bootload, you issue osl_server commands from command level.

On a bridge module, you typically specify the -super argument of the osl_server command to create outbound/inbound OSL servers. (Outbound/inbound OSL servers forward outbound requests from nonbridge modules on the local system to other bridge modules on remote systems. They also receive inbound requests from other bridge modules on remote systems for nonbridge modules on the local system.)

NOTE

All OSL servers on a module must be of the same type. If you specify the `-super` argument for one `osl_server` command, you must specify it for all `osl_server` commands issued on the module so that the module has only outbound/inbound OSL servers.

For additional information on the commands used to start `osl_server` processes, see ''Commands That Start the OSL Server Processes'' on page 3-17. For a complete description of the `osl_server` command, see Chapter 6.

On a non-bridge module, you typically start a number of `osl_server` processes that equals twice the value you specify for the `max_open_servers` field of the `new_modules.tin` file. On a bridge module, you typically start a number of `osl_server` processes that equals four times the value you specify for the `max_open_servers` field in the `new_backbone_systems.tin` file. (For information on specifying values for the `new_modules.tin` file, see ''Creating Entries in the `new_modules.tin` File'' on page 3-7.)

NOTE

To start an `osl_server` process on a bridge module, you must issue the `osl_server` command with the `-super` argument.

## The Command That Activates SOSL Net Driver

You must uncomment the following command to activate SOSL Net driver (`sosl_net_driver`). To uncomment a command, delete the ampersand (&) and the space preceding it.

```
configure_comm_protocol sosl_net_driver
```

You must also add the OSL multiplexor to the `devices.tin` file, as described in ''Adding the OSL Multiplexor to the `devices.tin` File'' on page 1-10, and start the OSL daemon process, as described in ''Starting the OSL Daemon Process'' on page 1-11.

For a complete description of the `configure_comm_protocol` command, see the manual *VOS System Administration: Configuring a System* (R287).

### The Command That Starts the OSL Daemon Process

The `module_start_up.cm` file must start the OSL daemon (`osl_daemon`) process. Uncomment the following command line in the `module_start_up.cm` file, substituting *`MODULE`* with the module name.

```
start_process 'osl_daemon #stcp.MODULE #stcp_osl_mux.MODULE'
     -privileged -priority 7
```

For additional information, see "Starting the OSL Daemon Process" on page 1-11.

### The Command That Starts the OSL Overseer

The `module_start_up.cm` file must start the OSL overseer process (`osl_overseer`). Uncomment the following command lines in the `module_start_up.cm` file:

```
!delete_file osl_overseer.out.old -brief
!rename osl_overseer.out osl_overseer.out.old
!create_file osl_overseer.out
!set_implicit_locking osl_overseer.out
!start_process -privileged 'osl_overseer' -process_name
     OSL_Overseer
```

## Starting Multiple-System OSL Communications

Once you configure the bridge and nonbridge modules of each system that will communicate using multiple-system OSL, you can start multiple-system OSL communications among the systems, using either of the following methods.

- You can reboot each bridge and nonbridge module immediately. The changes that establish permanent operation of the cross-system communications take effect immediately. You can reboot only the bridge modules and avoid rebooting the nonbridge modules simply by issuing the `configure_systems` command with the `-reset` argument from command level on each nonbridge module. This enables each system to recognize its `new_systems.table` file.

  **CAUTION**

  Using the `-reset` argument with the commands `configure_systems` and `configure_modules` may interrupt cross-system and cross-module communications.

- You can issue all of the appropriate VOS commands to start multiple-system OSL communications during the current bootload, then make the changes that will

establish permanent network operation whenever you reboot the system (that is, the `module_start_up.cm` file must also issue the commands).

The latter method lets you start cross-system communications without interrupting any module's current operation, thus minimizing the number of times a module is rebooted.

This section describes the procedures that start multiple-system OSL communications during the current bootload. Follow these procedures if you want to avoid rebooting the modules in each system. Note that these startup procedures encompass the configuration procedures described earlier in this chapter, such as the creation of configuration-table files and the execution of commands that establish various configuration components. These procedures also assume the following:

- You have configured each system (for example, using the single-system OSL configuration procedures described in Chapter 3).
- STCP is running on each bridge module that communicates using multiple-system OSL. These procedures include those described in "STCP Requirements" on page 1-9.
- You have configured and installed any additional physical interfaces (if you want to add dedicated cross-system routes).

To perform the startup procedures, you must be a privileged user with modify access to the `(master_disk)>system` directory.

If you need to review the procedures for establishing OSL communications within a system, see Chapter 3. If you need to reboot the modules, follow the procedures described in the manual *VOS System Administration: Starting Up and Shutting Down a Module or System* (R282).

The following procedures start multiple-system OSL communications during the current bootload.

1. Check that the bridge module of each system that uses multiple-system OSL is configured for STCP and has the hardware and software required to establish STCP connections.
2. If you have not already done so, edit the `new_backbone_systems.tin` file on each bridge module to include an entry for each system. Issue the `create_table` command to create the new `new_backbone_systems.table` file. Issue the `configure_systems` command on each bridge module to enable VOS to immediately recognize all newly added systems defined in each bridge module's `new_backbone_systems.table` file. Do **not** place the `new_backbone_systems.table` file on any nonbridge modules.
3. On the bridge module of each system that will communicate using multiple-system OSL, issue the `start_process` command to start the `osl_server` processes and the `configure_comm_protocol` command to activate SOSL Net driver

(sosl_net_driver). On a bridge module, you typically specify the -super argument of the osl_server command to create outbound/inbound OSL servers. If the network-watchdog process is not already running, issue a start_process command to start the process on each bridge module. Note that the start_process command lines are part of command sets that establish the conditions for the proper operation of the server and watchdog processes.

**NOTE**

All OSL servers on one module must be of the same type. If you specify the -super argument for one osl_server command, you must specify it for all osl_server commands issued on the module so that the module has only outbound/inbound OSL servers.

4. On each system's master module, edit the new_systems.tin file to include an entry for each system. Issue the create_table command to create the new new_systems.table file, and issue the broadcast_file command to install the new table on each module in the system. (If you do not want the bridge module to have the new_systems.table file, you can issue the broadcast_file command and exclude the bridge module by specifying it in the -exclude argument.) Then, issue the configure_systems command with the -reset argument, which you can do in one of two ways.
   - Issue the configure_systems command directly from each module in the system.
   - From the module you designate as the master module, create a subprocess on each module and then execute the configure_systems command.

**CAUTION**

Using the -reset argument with the commands configure_systems and configure_modules may interrupt cross-system and cross-module communications.

For a description of the configure_systems command, see Chapter 6. For a description of how to execute the configuration commands, see the manual *VOS System Administration: Configuring a System* (R287).

5. If you have not already done so, edit the network_access.tin file on each system's master module to include an entry for each system. Each system entry indicates whether a user from that system must be registered in the current system's registration database and whether that user must provide a password when accessing the current system. Issue the create_table command to create a new network_access.table file for each bridge module, and issue the broadcast_file command to install the new table on each module in the

system. VOS recognizes this configuration-table file immediately upon installation; therefore, no configuration command is necessary. Issue the `list_systems` command to verify that the current module recognizes the modules on other systems in the network. The *VOS Commands Reference Manual* (R098) documents the `list_systems` command.

# Modifying Existing Systems

The following sections describe how to modify existing systems.


- ‘‘Adding a System to a Multiple-System Configuration’’ on page 4-35
- ‘‘Deleting a System from a Multiple-System Configuration’’ on page 4-37


## Adding a System to a Multiple-System Configuration

The following sections describe how to add a system to a multiple-system configuraiton.


- ‘‘Adding a System Permanently’’ on page 4-35
- ‘‘Adding a System Temporarily’’ on page 4-37


### Adding a System Permanently

To add a system that uses multiple-system OSL for cross-system communications permanently and to have VOS recognize it during the current bootload, perform the following steps.

1. On each module in the new system, check that the `module_start_up.cm` file issues all pertinent configuration commands, such as the commands that establish the different aspects of STCP and the commands that start the appropriate single-system networking processes and the network-watchdog process. If the new system uses OSL for single-system communications, you should configure it using the procedures described in Chapter 3. For STCP, include the commands described in ‘‘STCP Requirements’’ on page 1-9.
2. Designate one module in the new system as the bridge module. Check that this module is equipped with the hardware and software needed to establish STCP communications.
3. Create a `network_access.table` file with the `create_table` command and install it in the `(master_disk)>system` directory of each module in the new system.
4. Add an entry for the new system to the `network_access.tin` file of every remote system to which the new system needs access. Create a `network_access.table` file with the `create_table` command, and install it in the `(master_disk)>system` directory of each module in the remote system.

5. On the new system's bridge module, edit the `new_backbone_systems.tin` file to create an entry for each system in the configuration, including the current system. Create the `new_backbone_systems.table` file with the `create_table` command, and install it in the `(master_disk)>system` directory of the bridge module.
6. On all other bridge modules in the configuration, add an entry for the new system to the `new_backbone_systems.tin` file. On each bridge module, re-create the configuration-table file using the `create_table` command.
7. If you want the updated `new_backbone_systems.table` file to take effect during the current bootload, issue the `configure_systems` command from a privileged process on each bridge module. The `configure_systems` command enables VOS to immediately recognize all additions to the `new_backbone_systems.table` file.
8. Issue the appropriate number of `start_process` commands to start the `osl_server` processes. On a bridge module, you typically specify the `-super` argument of the `osl_server` command to create outbound/inbound OSL servers. Issue the `configure_comm_protocol` command to activate SOSL Net driver (`sosl_net_driver`). If the network-watchdog process has not already been defined, issue a `start_process` command to start this process on each bridge module. Note that the `start_process` command lines are part of command sets that establish the conditions for the proper operation of the server and watchdog processes.

   NOTE

   All OSL servers on one module must be of the same type. If you specify the `-super` argument for one `osl_server` command, you must specify it for all `osl_server` commands issued on the module so that the module has only outbound/inbound OSL servers.

9. Add an entry for the new system to the `new_systems.tin` file on the master module in the current system (starting with the new system). Create the `new_systems.table` file with the `create_table` command, and install it in the master module's `(master_disk)>system` directory.
10. Copy the `new_systems.table` file to every module in the system. If you do not want the bridge module to have the `new_systems.table` file, you can issue the `broadcast_file` command and exclude the bridge module by specifying it in the `-exclude` argument.
11. If you want the updated `new_systems.table` file to take effect immediately, issue the `configure_systems` command from a privileged process on each module in the current system.
12. Repeat steps 9 through 11 for all other systems in the configuration.

### Adding a System Temporarily

There are two situations in which you will want changes to take effect for the current bootload: after you have installed new configuration-table files, and when you want to make **temporary** changes that are not reflected in either the configuration-table files or the `module_start_up.cm` file. This section describes how to add a system for the duration of the current bootload only.

If you want VOS to recognize an additional system for the duration of the current bootload only, use the `add_system` command. If you are issuing this command from a bridge module, include the `-backbone` argument. The `add_system` command enables VOS to recognize a system that does not have an entry in a `new_systems.table` or `new_backbone_systems.table` file. You can also use this command to restore a particular system to service.

NOTE

The `add_system` command lets you change the current system's configuration during the current bootload. For more information, see the description of the `add_system` command in Chapter 6.

## Deleting a System from a Multiple-System Configuration

The following sections describe how to delete a system from a multiple-system configuration.


- "Deleting a System Permanently" on page 4-37
- "Deleting a System Temporarily" on page 4-38


### Deleting a System Permanently

To delete a system permanently from a multiple-system OSL configuration, follow these steps.

1. Remove the system's entry from the `new_backbone_systems.tin` file on each bridge module in the configuration, including the bridge module of the system being deleted.
2. On each bridge module, re-create the `new_backbone_systems.table` file with the `create_table` command, and check that the file resides in the `(master_disk)>system>configuration` directory of each bridge module.
3. On the master module of one of the systems that will remain in the configuration, remove from the `new_systems.tin` file the entry for the system to be deleted. Re-create the `new_systems.table` file with the `create_table` command. Issue the `broadcast_file` command to copy the `new_systems.table` file to all other nonbridge modules in the current system.

4. On each nonbridge module in the current system, issue the `delete_system` command to delete the system, then issue the `delete_system` command on the bridge module. Alternatively, issue the `configure_systems` command with the `-reset` argument from a privileged process on each module in the current system to enable the new table to take effect during the current bootload.

   **CAUTION**

   Using the `-reset` argument with the commands `configure_systems` and `configure_modules` may interrupt cross-system and cross-module communications.

5. Repeat steps 3 and 4 for all other systems in the configuration.

### Deleting a System Temporarily

To delete a system for the duration of the current bootload only (that is, temporarily), issue the `delete_system` command. When you issue the `delete_system` command, VOS no longer recognizes the specified system. (For a description of the `delete_system` command, see Chapter 6.)

**CAUTION**

Before issuing the `delete_system` command, check that users working on any module in any system do not require the continued use of the system you plan to delete.

The following manuals describe how to make other configuration changes.

- For an explanation of how to **rename a system**, see the discussion on changing a module or system identifier (or a site ID) in the manual *VOS System Administration: Administering and Customizing a System* (R281).
- For an explanation of how to **change a system number**, see the discussion on changing a module or system identifier (or a site ID) in the manual *VOS System Administration: Administering and Customizing a System* (R281).
- For an explanation of how to change the device configuration, see the discussion on configuring devices in the manual *VOS System Administration: Configuring a System* (R287).

# Chapter 5
# Troubleshooting OSL

This chapter presents information about troubleshooting common errors that can occur while configuring OSL. It contains the following sections.

- "Checklist: Avoiding Common Configuration Problems" on page 5-1 presents a checklist for avoiding common hardware and software configuration problems.
- "Configuration Error Messages" on page 5-3 describes configuration error messages that may appear on the terminal's screen. Additional error messages appear in the system error log file (`syserr_log.`*`date`*) when the `module_start_up.cm` file tries to execute commands.

Other error messages that apply to OSL include the traditional network error messages such as `Invalid module number` and `Invalid system number`. For descriptions of these traditional error messages, see the *VOS Codes and Messages Reference Manual* (R132).

## Checklist: Avoiding Common Configuration Problems

The following checklist can help you avoid common hardware/software configuration problems. This checklist addresses aspects of STCP configuration, hardware configuration, and OSL client/server configuration.

- Check that STCP is configured properly on the module. Check that in the `module_start_up.cm` file, the command lines for STCP are uncommented (that is, the ampersand and space have been deleted). To determine that STCP devices are properly configured, issue one of the following commands.
  - `list_devices -type streams`, which displays a list of the STREAMS devices on the module, including the OSL multiplexor
  - `netstat`, which reports the status of the logical interfaces

  Item checked: ☐

- Check that the IP addresses you specified are correct and unique (duplicate IP addresses are not allowed). The network portion of the IP address must identify a network/subnet to which the module is directly connected. When the required TCP/IP components (for example, physical interfaces) are configured correctly, you can issue the `ping` command.

  Item checked: ☐

- Check that the physical interface (for example, an Ethernet PCI adapter) is installed properly, and that all cables are connected securely. If a physical interface or a cable is pulled, a timeout period occurs, after which a message (`Timeout period has expired. No response from server`) appears in the `syserr_log.`***`date`*** file, and the `list_modules` command identifies the module as `offline`.

  Item checked: ☐

- Check that the number of `osl_server` processes on the module equals the following.
  - On non-bridge modules, the number of `osl_server` processes should typically equal twice the value you specify for the `max_open_servers` field of the `new_modules.tin` file (or for the `-max_open_servers` argument of the `add_module` command).
  - On bridge modules, the number of `osl_server` processes that you start should typically equal four times the value you specify for the `max_open_servers` field of the `new_backbone_systems.tin` file (or for the `-max_open_servers` argument of the `add_system` command).

  Item checked: ☐

- Check that the `new_systems.table` file on a nonbridge module does not specify IP addresses or a funnel. The IP addresses used for cross-system communications or the intermediary funnel system used to reach a given system apply only to a system's designated bridge module, not to a nonbridge module. The file `new_backbone_systems.table` on a bridge module contains these IP addresses or a funnel in any given system entry.

  Item checked: ☐

- Check that a route exists to each module. If no route exists, a message (`Warning: No way to reach this destination from this module`) appears on the terminal's screen and/or in the `syserr_log.`***`date`*** file.

  Item checked: ☐

For additional information, see one or more of the following manuals, as appropriate for your configuration.

- For information about STCP, including the STCP commands `netstat`, `ifconfig`, and `ping`, see the *VOS STREAMS TCP/IP Administrator's Guide* (R419). Additional requirements are described in "STCP Requirements" on page 1-9. To run OSL with STCP, you **must** follow the procedures described in that section in addition to the standard STCP configuration procedures.
- For information about migrating from OS TCP/IP to STCP, see the *VOS STREAMS TCP/IP Migration Guide* (R418).
- For information about the `list_devices` command, see the *VOS Commands Reference Manual* (R098).

## Configuration Error Messages

This section describes the error messages associated with the configuration of OSL. These messages are written to the terminal's screen if you issue commands such as `configure_modules` or `configure_systems` with incorrect configuration information. In some cases, the error message has a prefix that identifies the command generating the error (for example, `configure_modules` or `list_users`) and a suffix that identifies the problem entry and the path name of the configuration-table file. Since different commands can return some of these messages, the descriptions do not identify a specific command. The messages are listed in alphabetical order.

▶ `An entry for the current system is required.`

**Error code name and number:** `e$define_cur_system` (5156)

**Description:** The `configure_systems` command returns this error when the current system is not in the configuration-table file being processed. The command abandons the current file and searches for the next configuration-table file.

▶ `Invalid Link station number. module_name.`

**Error code name and number:** `e$invalid_station` (1372)

**Description:** The `configure_modules` command returns this error when the station number specified (for example, `0`) does not belong to any module in this system.

▶ `Number of servers configured is not within system limits.`

**Error code name and number:** `e$too_many_connections` (5145)

**Description:** The system returns this error when the configured number of TCP ports exceeds the system limit of 32. (The configured number of TCP ports is twice

the value specified for the `-max_open_servers` field in the `new_modules.table` file and the `-max_open_servers` field in the `new_backbone_systems.table` file. For information on the `new_modules.table` file, see "Using the New Modules Configuration-Table File" on page 3-6. For information on the `new_backbone_systems.table` file, see "Using the New Backbone Systems Configuration-Table File" on page 4-12.)

- `The current system cannot have a funnel specified.`

  **Error code name and number:** `e$bad_funnel` (`5159`)

  **Description:** The `configure_systems` command returns this error when you specify a funnel for the current system. The command abandons the current configuration-table file and searches for the next configuration-table file. Check that the entry for the current system in the `new_backbone_systems.tin` file does not include the `funnel` field. (The `new_systems.tin` file should **never** have the `funnel` field for any system entry.)

- `The hostname supplied duplicates an existing hostname.`

  **Error code name and number:** `e$dup_hostname` (`5144`)

  **Description:** The IP address that you specified for a TCP/IP logical interface has already been specified. Check the IP addresses specified in the `new_modules.tin` file and/or the `new_backbone_systems.tin` file.

- `The station number of the bridge module must be supplied.`

  **Error code name and number:** `e$bridge_station_req` (`5155`)

  **Description:** The `configure_systems` command returns this error when the station number of a bridge module for the current system is not specified or is 0. The command abandons the current configuration-table file and searches for the next configuration-table file. Check that the `new_systems.tin` and `new_backbone_systems.tin` files supply the station number for the current system's bridge module.

- `The target module is offline.`

  **Error code name and number:** `e$module_down` (`1134`)

  **Description:** The system returns this error when one or more of the adapters or cables have been pulled and the timeout value has expired. Check that the adapters and cables are installed properly.

- `The target server is not in operation. system_name/module_name.`

  **Error code name and number:** `e$server_down` (`1103`)

  **Description:** The system returns this error when a user issues any command (such as `list_users`) to reach a server on another module, and the local driver for OSL is not running or not listening to the expected socket number. The message identifies the system name and module name associated with the target module.

- `This configuration file should be found only on a bridge module.`

  **Error code name and number:** `e$bridge_config_file` (`5151`)

  **Description:** The `configure_systems` command returns this error when the command finds the `new_backbone_systems.table` file on a nonbridge module. The command abandons the current file and searches for the next configuration-table file.

- `This feature is not yet implemented.`

  **Error code name and number:** `e$not_yet_implemented` (`1062`)

  **Description:** The system returns this error when you issue the `configure_modules` command and the module does not yet support OSL. The command continues processing the next entry in the appropriate configuration-table file.

- `Undefined module name.`

  **Error code name and number:** `e$module_not_found` (`1130`)

  **Description:** This error may indicate that the `new_modules.table` file does not contain an entry for the current module, or that there is no way to reach the destination module. In either case, a user sees this error message after trying to perform any transaction. The error message identifies the command generating the error (such as the `configure_modules` or `list_users` command) as well as the module and path name of the configuration-table file (for example, `#m10` in `%admin#m10_mas>system>configuration>new_modules.table`). If the `new_modules.tin` file does not contain an entry for the current module, create an entry for the current module in the `new_modules.tin` file.

- `Warning: Mutually incompatible definitions. "=funnel"/"hostname" for` ***`system_name`*** `in` ***`table_path`***`.`

  **Error code name and number:** `e$defs_incompatible` (`5158`)

  **Description:** The `configure_systems` command returns this error when a system entry contains values for the `funnel_name` field and any `hostname`*`n`* field. This error identifies the system name (for example, `%admin`) and the path name of the configuration-table file. The command continues processing the entry.

- `Warning: No way to reach this destination from this module.`

  **Error code name and number:** `e$no_route_warning` (`5154`)

  **Description:** The `configure_modules` or `configure_systems` command returns this error when the remote module or system has no communications paths in common with the current module or system. Check that the current module or system supports a communications scheme used by the remote module or system and has an appropriate configuration-table file entry, and that the entry for the remote module or system correctly represents its capabilities. The command continues processing the next entry in the appropriate configuration-table file.

- `Warning: This field should be null on a non-bridge module.`

  **Error code name and number:** `e$invalid_for_nonbridge` (`5157`)

  **Description:** The system returns this error when the `new_systems.table` file on a nonbridge module contains a value for the `funnel_name` field or any `hostname`*`n`* field. The message identifies the system (for example, `%admin`), the field, and the path name of the appropriate configuration-table file (for example, `%admin#m10_mas>system>configuration>new_systems.table`). The `configure_systems` command continues processing the entry.

# Chapter 6
# OSL Administrative Commands

This chapter documents the following OSL administrative commands in alphabetical order. These commands let you create and manage configurations that use OSL.

- add_module
- add_system
- configure_modules
- configure_systems
- delete_module
- delete_system
- osl_admin
- osl_daemon
- osl_overseer
- osl_server

All examples in the command descriptions refer to modules used in the sample configurations in Chapter 3 and Chapter 4. You may want to refer to these configurations as you examine the examples in this chapter.

The manual *VOS Communications Software: X.25 and StrataNET Administration* (R091) also documents the commands add_system, configure_systems, and delete_system, but this manual documents the most recent changes to the add_system and configure_systems commands.

This manual references some administrative commands that are documented in the following manuals.

- The manual *VOS System Administration: Administering and Customizing a System* (R281) documents the network_watchdog commands.
- The manual *VOS System Administration: Registration and Security* (R283) documents the create_user_sysdbs, login_admin, and registration_admin commands.

- The manual *VOS System Administration: Disk and Tape Administration* (R284) documents the `add_disk`, `display_disk_label`, and `update_disk_label` commands.
- The manual *VOS System Administration: Configuring a System* (R287) documents the `broadcast_file`, `configure_comm_protocol`, `configure_devices`, `configure_disks`, and `create_table` commands.
- The *VOS Commands Reference Manual* (R098) documents the following general-user commands.
  - `copy_file`
  - `list_devices`
  - `list_modules`
  - `list_systems`
  - `login`
  - `start_process`
  - `verify_system_access`

# add_module *Privileged*

## Purpose

The add_module command enables VOS on the current module to recognize the specified module for the duration of the current bootload.

## Display Form

```
-------------------------------- add_module ----------------------------------
module_name:
module_number:
station_number:
-stratalink_hw:       yes
-open_socket_number:  0
-max_open_servers:    0
-base_port:           3000
-hostnames:
```

## Command-Line Form

```
add_module module_name  module_number
        [station_number]
        [-no_stratalink_hw]
        [-open_socket_number number]
        [-max_open_servers number_of_ports]
        [-base_port tcp_port_number]
        [-hostnames hostname1 hostname2 ... hostname8]
```

## Arguments

- *module_name* **Required**
  The name of the module to be recognized during the current bootload.

- *module_number* **Required**
  The number of the specified module. A system can contain up to 32 modules. The value you specify, however, can be an integer from 1 through 127.

- *station_number*
  The station number of the specified module. You can omit this argument only if, on the master disk label, the station number is identical to the module number for the specified module. (This value is typically the same as the module number; a system can contain up to 32 modules, but you can specify this value as an integer from 1 through 127.)

- -no_stratalink_hw CYCLE
  Specifies that the module you are adding to the system does not have proprietary StrataLINK hardware and is not part of a proprietary StrataLINK configuration. You must specify this argument. (Modules running VOS Release 15.0.0 (or later) do not support proprietary StrataLINK hardware.) By default, the command assumes that the specified module has proprietary StrataLINK hardware (including StrataLINK hardware for connections to StrataLINK subrings or backbone rings).

- -open_socket_number *number*
  In this field, specify the value 31. The default value is 0.

- -max_open_servers *number_of_ports*
  Specifies a value that determines the number of TCP ports available to the server side of SOSL Net driver. The TCP ports and SOSL Net driver are located on the specified module. The server side of the driver listens on these TCP ports for client requests. The number of TCP ports is twice the value specified by the -max_open_servers argument. For example, the value 2 enables the server side of the driver to listen on four TCP ports. The default value is 0; the maximum value is 32.

  For typical configurations, the value for the -max_open_servers argument of the add_module command should equal the value specified for the max_open_servers field in the new_modules.tin and new_modules.tin nnew_backbone_systems.tin files. If, however, you need to specify different values, the value for the -max_open_servers argument of the add_module command (or the value for the max_open_servers field in the new_modules.tin file) should be greater than (or equal to) the value specified for the new_backbone_systems.tin file.

- -base_port *tcp_port_number*
  Specifies the lowest-numbered TCP port where the server side of SOSL Net driver listens for client requests. The TCP ports and SOSL Net driver are located on the specified module. The TCP port number must be an integer from 3000 through 20,000. The default listen port number for SOSL Net driver is 3000; additional ports are numbered 3001, 3002, and so on. For more information, see the Explanation.

NOTE

The value that you specify for the `-base_port` argument of the `add_module` command **must not** be in the range of port numbers determined by the value that you specify for the `-base_port` argument of the `add_system` command.

- `-hostnames` *`hostname1 hostname2 ... hostname8`*
  Specifies the IP addresses of the STCP logical interfaces used to receive OSL requests on the specified module. A sample IP address is `134.111.5.10`. Separate each value with a space. If you use this argument to specify IP addresses, make sure that you specify values for the `-max_open_servers` and `-base_port` arguments. Note that OSL does not support fully qualified host names.

## Explanation

The `add_module` command lets you add a module to the system for the duration of the current bootload. The module will be accessed by the current module using OSL. On a multimodule system, you must issue this command on each module to ensure that each module recognizes the new module.

NOTE

The `add_module` command also lets you change the **current module**'s OSL configuration during the current bootload. On the current module, you simply issue an `add_module` command that specifies the current module's updated OSL information. Although the command returns a message stating that the specified module is already defined, it accepts the new information. (The message is generated because the current module cannot be deleted from the configuration.) Also, you must issue the `add_module` command on each module that will be affected by the change to the current module. If the updated information is included in the `new_modules.table` file and the table is broadcast to each module, you can implement the table changes by issuing the `configure_modules` command on each module.

The command verifies the configuration, processes the information you supply, and then adds the specified module so that it is accessible using OSL. If both modules do not have a common communications scheme, the command generates the error

e$no_route_warning (5154) to indicate that the current module cannot reach the destination module.

SOSL Net driver on the module specified by the *module_name* argument uses the *number_of_ports* value of the -max_open_servers argument and the *tcp_port_number* value of the -base_port argument to establish a range of TCP port numbers on which the server side of SOSL Net driver listens for requests. The highest number in the range equals the following value.

*tcp_port_number* + (*number_of_ports* * 2) - 1

For example, when the *number_of_ports* value is 2 and the *tcp_port_number* value is 3000, SOSL Net driver on the specified module establishes TCP ports 3000 through 3003 for its server side to listen for client requests. With a *number_of_ports* value of 10, the range is 3000 to 3019.

The add_module and configure_modules commands take effect during the current bootload and enable VOS to recognize a module for the duration of the current bootload. However, the commands differ in the following ways.

- The add_module command typically enables VOS to recognize a single module that has not been defined previously. (See the previous note concerning the current module.) This command does not require the module to be defined in the new_modules.table file, which means that you can add the module **temporarily** (that is, until the next bootload).
- The configure_modules command configures all modules defined in the new_modules.table file (or in the older modules.table file) that have not been defined previously. The module_start_up.cm file contains this command to ensure that VOS accesses the appropriate modules configuration-table file at each bootload automatically.

The add_module arguments shown in the display form correspond to the fields in the new_modules.table file. (The first three arguments correspond to the fields in the older modules.table file.) For information about each of the fields in the new_modules.table file, see "Using the New Modules Configuration-Table File" on page 3-6.

## Examples

In the following example, the add_module command enables VOS on the current module, #m10 in the OSL system %admin, to recognize a fourth module, #m13, for the duration of the current bootload. (For a depiction of the three-module OSL system

%admin, see Figure 3-1.) The module number for #m13 is 13, and its station number is 13.

```
add_module m13 13 13 -no_stratalink_hw -max_open_servers 2
  -base_port 3000 -hostnames 134.111.5.13 134.111.6.13
```

## Related Information

For information about creating an OSL system, see Chapter 3. Also in Chapter 3, ‘‘Using the New Modules Configuration-Table File’’ on page 3-6 describes how to work with the new_modules.table file. See also the configure_modules command description later in this chapter.

# add_system *Privileged*

## Purpose

The add_system command enables VOS on the current module to recognize the specified system for the duration of the current bootload.

## Display Form

```
---------------------------------- add_system -----------------------------------
system_name:
system_number:
station_number:
socket_number:      2
-max_open_servers: 0
-base_port:        0
-hostnames:
-backbone:         no
-funnel:
```

## Command-Line Form

```
add_system system_name  system_number  station_number
          [socket_number]
          [-max_open_servers number_of_ports]
          [-base_port tcp_port_number]
          [-hostnames hostname1 hostname2 ... hostname8]
          [-backbone]
          [-funnel system_name]
```

## Arguments

- *system_name* **Required**
  The name of the system to be added for the duration of the current bootload.

- *system_number* **Required**
  The number of the system specified in the *system_name* field. This value must be an integer from 1 through 255. The number assigned to each system in the network must be unique.

- *station_number* **Required**

  If the current module is a nonbridge module, this is the station number of the bridge module in the **current system** that provides network access to the remote system specified in the *system_name* argument. Applying the configuration conventions that Stratus supports, if the current system is in a multiple-system OSL configuration, this module is always the current system's bridge module.

  If the current module is a bridge module, the argument specifies the station number of the bridge module (or gateway/bridge module) that provides network access to the remote system specified in the *system_name* argument. Applying the configuration conventions that Stratus supports, if the remote system is accessible from the current bridge module using the multiple-system OSL software, this is the current module's station number.

  The value for this argument must be an integer from 1 through 127.

- *socket_number*

  In this field, specify the value 31. The default value is 2.

- -max_open_servers *number_of_ports*

  Specifies a value that determines the number of TCP ports available to the server side of SOSL Net driver. The TCP ports and SOSL Net driver are located on the remote module providing network access to the specified system (that is, the specified system's bridge module). The server side of SOSL Net driver listens on these TCP ports for client requests. The number of TCP ports equals the value specified by the -max_open_servers argument. For example, the value 2 enables the server side of SOSL Net driver to listen on two TCP ports. The default value is 0; the maximum value is 32.

  For typical configurations, the value for the -max_open_servers argument of the add_system command should equal the value specified for the max_open_servers field in the new_modules.tin and new_modules.tin nnew_backbone_systems.tin files. If, however, you need to specify different values, the value for the -max_open_servers argument of the add_system command (or the value for the max_open_servers field in the new_modules.tin file) should be greater than (or equal to) the value specified for the new_backbone_systems.tin file.

- -base_port *tcp_port_number*

  Specifies the lowest-numbered TCP port where the server side of SOSL Net driver listens for client requests. The TCP ports and SOSL Net driver are located on the remote module providing network access to the specified system (that is, the specified system's bridge module). The default value for this argument is 0. The recommended TCP port number is 4000; the number must be an integer from 3000 through 20,000. The default TCP port number for SOSL Net driver on a remote system's bridge module is 4000; additional ports are numbered 4001, 4002, and so on. For more information, see the Explanation.

NOTE

The value that you specify for the `-base_port` argument of the `add_system` command **must not** be in the range of port numbers determined by the value that you specify for the `-base_port` argument of the `add_module` command.

- `-hostnames` *`hostname1 hostname2 ... hostname8`*
  Specifies the IP addresses of the STCP logical interfaces on the remote module providing network access to the specified system using OSL and STCP (the remote system's bridge module). Separate each value with a space. Specify an IP address as it appears in the `hosts` file. A sample IP address is `134.111.5.10`. Remember to specify two IP addresses if you support dual routes of communications among the systems. You specify IP addresses only if the current system can access the remote system directly.

NOTES

1. OSL does not support fully qualified host names.
2. If you specify one or more values for the `-hostnames` argument, do not specify the `-funnel` argument; these arguments are mutually exclusive.

- `-backbone` CYCLE
  Specifies that the current module (the module on which you are issuing the `add_system` command) is the current system's bridge module. If the current module is not the bridge module, do not specify this argument. By default, the command assumes that the current module is not a bridge module.

- `-funnel` *`system_name`*
  Specifies the system name associated with the remote bridge module. If the current system uses multiple-system OSL to indirectly access the remote system through another system (called a funnel), you must specify this argument; that is, you specify the `-funnel` argument only if the current system cannot access the remote system directly.

NOTE

If you specify the `-funnel` argument, do not specify the `-hostnames` argument; these arguments are mutually exclusive.

## Explanation

The `add_system` command enables VOS on the current module to recognize a specified system for the duration of the current bootload. The specified system will be accessed using multiple-system OSL. If both systems cannot access each other using the information you supply, the command generates an error.

NOTE

If the current module is the bridge module, the `add_system` command lets you change the cross-system communications scheme supported by the current system during the current bootload. On the current bridge module, you simply issue an `add_system` command that specifies the current bridge module's new cross-system information. Although the command returns a message stating that the specified system is already defined, it accepts the new information. (The message is generated because the current system cannot be deleted from the configuration.) Also, you must issue the `add_system` command on the bridge module of each system that will be affected by the change. If the updated information is included in the `new_backbone_systems.table` file on each affected bridge module, you can put the table-file changes into effect by issuing the `configure_systems` command on each bridge module.

SOSL Net driver on the specified system's bridge module (that is, the driver on the remote module that provides network access to the system specified by the *`system_name`* argument) provides TCP ports on which the server side of the driver listens for client requests. The driver provides these ports by establishing a range of TCP port numbers using the following values.

- the *`number_of_ports`* value of the `-max_open_servers` argument
- the *`tcp_port_number`* value of the `-base_port` argument

The highest number in the range equals the following value.

*`tcp_port_number`* + *`number_of_ports`* - 1

For example, when the *`number_of_ports`* value is 2 and the *`tcp_port_number`* value is `4000`, SOSL Net driver on the specified system's bridge module establishes TCP ports 4000 and 4001 for its server side to listen for client requests. With a *`number_of_ports`* value of `10`, the range is 4000 to 4009.

The add_system and configure_systems commands take effect during the current bootload and enable VOS to recognize a system. However, the commands differ in the following ways.

- The add_system command typically enables VOS to recognize, for the duration of the current bootload, a single system that has not been defined previously. (See the previous note concerning the current system.) This command does not require the system to be defined in either the new_systems.table or new_backbone_systems.table file (or the older systems.table or backbone_systems.table file), which means that you can add the system **temporarily** (that is, until the next bootload). In a multimodule system, you must issue the add_system command on each module that you want to recognize the system.
- The configure_systems command configures all systems defined in the appropriate systems configuration-table file that have not been defined previously. (The file can be new_systems.table or systems.table if the current module is a nonbridge module; on a bridge module, the file can be new_backbone_systems.table or backbone_systems.table.) The module_start_up.cm file contains this command to ensure that VOS accesses the appropriate systems configuration-table file at each bootload automatically.

The add_system arguments correspond to the fields in the appropriate systems configuration-table files.

To add the system permanently to the network configuration, you must add an entry for the system in the appropriate systems configuration-table file, and then create, install, and activate an updated configuration-table file.

## Examples

This section provides two examples.

- The first example enables a bridge module to recognize a system that will be accessed using OSL.
- The second example enables a nonbridge module to recognize a system that will be accessed using OSL via the local bridge module.

In the first example, an add_system command enables VOS on the current module, bridge module #m20 in the system %mfg, to recognize the system %admin for the duration of the current bootload. (The system %admin is the sample system shown in Figure 3-1; the systems %admin and %mfg are shown in Figure 4-1.) The system number for %admin is 1, and the station number of the bridge module providing access to it is 20 (bridge module #m20 in %mfg). In addition, the -backbone argument

indicates that the current module is a bridge module. The IP addresses identify logical interfaces on the bridge module in %admin.

```
add_system admin 1 20 -backbone -max_open_servers 2
  -base_port 4000 -hostnames 134.111.5.10 134.111.6.10
```

In the second example, an add_system command enables VOS on the current module, nonbridge module #m31 in the system %sales, to recognize the system %admin for the duration of the current bootload. The system number for %admin is 1, and the station number of the module providing access to it is 30 (bridge module #m30 in the current system, %sales). Since you issue the command from a nonbridge module, you do not specify the -max_open_servers, -base_port, and -hostnames arguments.

```
add_system admin 1 30
```

## Related Information

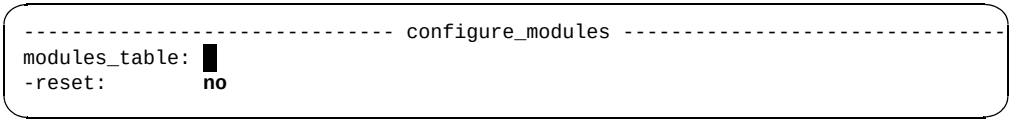
For information about creating multiple-system OSL configurations, see Chapter 4. Chapter 4 also describes the new_systems.table and new_backbone_systems.table files. The manual *VOS Communications Software: X.25 and StrataNET Administration* (R091) describes the systems.table and backbone_systems.table files in detail.

# configure_modules *Privileged*

## Purpose

The configure_modules command enables VOS on the current module to immediately recognize all modules that are defined in the new_modules.table file (or the older modules.table file) and that have not previously been installed.

## Display Form




## Command-Line Form

```
configure_modules [modules_table]
                  [-reset]
```

## Arguments

- *modules_table*
  The modules configuration-table file to be installed (new_modules.table). If you do not specify a configuration-table file, VOS automatically searches for the new_modules.table file in its (master_disk)>system directory.

- -reset CYCLE
  Specifies that the configure_modules command will read all entries from the specified module configuration-table file and overwrite any existing entries for the modules involved. By default, the -reset argument does **not** specify that the command will read all entries from the appropriate configuration-table file and overwrite any existing entries. **Note that using the -reset argument with the configure_modules and configure_systems commands may interrupt cross-system and cross-module communications.**

## Explanation

This command is included in the sample `module_start_up.cm` file, but you can issue `configure_modules` from command level if you want VOS on the current module to recognize newly defined or modified modules in the appropriate modules configuration-table file during the current bootload.

If you modify the specified configuration-table file, issue the `broadcast_file` (or `copy_file`) command to install the updated table in the `(master_disk)>system` directory of all modules in the system.

The `-reset` argument lets you change information such as module names and addresses by updating the configuration-table file and then issuing the `configure_modules` command. You do not have to issue the `delete_module` command followed by the `add_module` command.

**CAUTION**

Using the `-reset` argument with the commands `configure_systems` and `configure_modules` may interrupt cross-system and cross-module communications.

If the current module is not in the `new_modules.table` file, the command generates the error `e$module_not_found` (`1130`). This error causes the command to terminate. If the remote module has no communications paths in common with the current module, the command generates the error `e$no_route_warning` (`5154`) and continues processing the next configuration-table file entry. For more information about error handling, see Chapter 5.

The `configure_modules` and `add_module` commands take effect during the current bootload and enable VOS to recognize a module. However, the commands differ in the following ways.

- The `configure_modules` command enables VOS to recognize all modules defined in the appropriate modules configuration-table file that have not been defined previously. The `module_start_up.cm` file contains this command to ensure that VOS accesses the appropriate modules configuration-table file at each bootload automatically.
- The `add_module` command enables VOS to recognize a single module that has not been defined previously. This command does not require that the module be defined in the appropriate modules configuration-table file, which means that you can add the module **temporarily** (that is, until the next bootload).

## Examples

The following command, issued on module `#m10` in the system `%admin`, enables VOS on `#m10` to recognize all modules defined in the `new_modules.table` file (and overwrite any existing entries) during the current bootload.

```
configure_modules -reset
```

## Related Information

For information about the `new_modules.table` file, see "Using the New Modules Configuration-Table File" on page 3-6. See also the `add_module` and `delete_module` command descriptions in this chapter.

# configure_systems *Privileged*

## Purpose

The configure_systems command enables VOS on the current module to recognize immediately all systems that are defined in the appropriate systems configuration-table file and that have not previously been installed.

## Display Form

```
----------------------------- configure_systems --------------------------------
system_file:
-reset:      no
```

## Command-Line Form

```
configure_systems [system_file]
                  [-reset]
```

## Arguments

- *system_file*
  The systems configuration-table file to be installed, which is one of the following: new_backbone_systems.table or the new_systems.table. If you do not specify a systems configuration-table file, VOS automatically searches for the new_backbone_systems.table file in the (master_disk)>system directory. If the file exists, VOS activates it; if it does not exist, VOS searches for the new_systems.table file.

- -reset CYCLE
  Specifies that the configure_systems command will read all entries from the appropriate systems configuration-table file and overwrite any existing entries for the systems involved. By default, the -reset argument does **not** specify that the command will read all entries and overwrite any existing entries. **Note that using the -reset argument with the commands configure_modules and configure_systems may interrupt cross-system and cross-module communications.**

## Explanation

This command is included in the sample `module_start_up.cm` file, but you can issue `configure_systems` from command level if you want VOS on the current module to recognize either newly defined or modified systems in the appropriate systems configuration-table file during the current bootload.

If you modify the relevant systems configuration-table file on the module, issue the `broadcast_file` (or `copy_file`) command to install the updated configuration-table file in the `(master_disk)>system` directory of all modules that need the file. Install the `new_systems.table` or `systems.table` file in the `(master_disk)>system` directory of all nonbridge modules in the system.

The `-reset` argument lets you change system names or addresses by updating the appropriate systems configuration-table file and then issuing the `configure_systems` command. You do not have to issue the `delete_system` command followed by the `add_system` command.

**CAUTION**

Using the `-reset` argument with the commands `configure_systems` and `configure_modules` may interrupt cross-system and cross-module communications.

If the command detects an error, it generates one of the following system error messages.

- If the current system is not in the configuration-table file being processed, the command generates the error `e$define_cur_system` (5156). The command then abandons the current file and searches for the next configuration-table file.
- If the station number of a bridge module for the current system is not specified or is 0, the command generates the error `e$bridge_station_req` (5155), abandons the current file, and searches for the next configuration-table file.
- If the command finds the `new_backbone_systems.table` file on a nonbridge module, it generates the error `e$bridge_config_file` (5151). The command then abandons the current file and searches for the next configuration-table file.
- If a funnel is specified for the current system, the command generates the error `e$bad_funnel` (5159). The command then abandons the current file and searches for the next configuration-table file.
- If a system entry contains values for the `funnel_name` field and any `hostname`*n* field, the command generates the error `e$defs_incompatible` (5158) as a warning message and continues processing the entry.

- If the `new_systems.table` file on a nonbridge module contains values for the `funnel_name` field or any `hostname`*`n`* field, the command generates the error `e$invalid_for_nonbridge` (`5157`) as a warning message and continues processing the entry.
- If the remote system has no communications paths in common with the current system, the command generates the error `e$no_route_warning` (`5154`) and continues processing the next table entry.

For more information about error handling and complete descriptions of the error messages in the preceding list, see Chapter 5.

The `configure_systems` and `add_system` commands take effect during the current bootload and enable VOS to recognize a system. However, the commands differ in the following ways.

- The `configure_systems` command enables VOS to recognize all systems defined in the appropriate systems configuration-table file that have not been defined previously. (The configuration-table file can be `new_systems.table` or `systems.table` if the current module is a nonbridge module; on a bridge module, the file can be `new_backbone_systems.table` or `backbone_systems.table`.) The `module_start_up.cm` file contains this command to ensure that VOS accesses the appropriate systems configuration-table file at each bootload automatically.
- The `add_system` command enables VOS to recognize, for the duration of the current bootload, a single system that has not been defined previously. This command does not require the system to be defined in the `new_systems.table` or `new_backbone_systems.table` file (or the older `systems.table` or `backbone_systems.table` file), which means that you can add the system **temporarily** (that is, until the next bootload). In a multimodule system, you must issue the `add_system` command on each module that you want to recognize the specified system.

## Examples

In the following example, a `configure_systems` command enables VOS on the current module, bridge module `#m20` in the system `%mfg`, to recognize immediately all systems defined in its own `new_backbone_systems.table`. (This system is part of the multiple-system configuration in Figure 4-1.) The `-reset` argument causes the command to overwrite any existing entries.

```
configure_systems (master_disk)>system>new_backbone_systems.table -reset
```

## Related Information

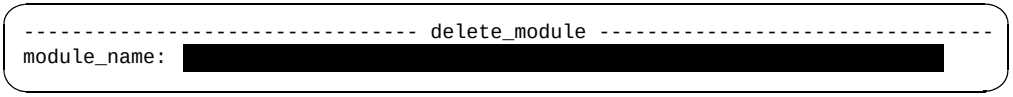
For information about the `new_systems.table` and `new_backbone_systems.table` files, see “Using the New Systems Configuration-Table File” on page 4-6 and “Using the New Backbone Systems Configuration-Table File” on page 4-12. See also the `add_system` and `delete_system` command descriptions in this chapter.

# delete_module *Privileged*

## Purpose

The `delete_module` command enables VOS on the current module to delete the specified module. The current module then no longer recognizes the specified module.

## Display Form




## Command-Line Form

`delete_module` *`module_name`*

## Arguments

- *`module_name`* **Required**
  The module to be deleted. Communications with the deleted module are terminated abruptly; therefore, take the necessary precautions regarding active user processes before deleting it (for example, notify all users).

## Explanation

You can use this command to correct an error resulting from an incorrect `add_module` command or to delete, for the current bootload, a module whose information in the `new_modules.table` or `modules.table` file has changed.

## Examples

The following command deletes the module `#m13` from VOS on the current module (`#m10` in the system `%admin`).

```
delete_module m13
```

## Related Information

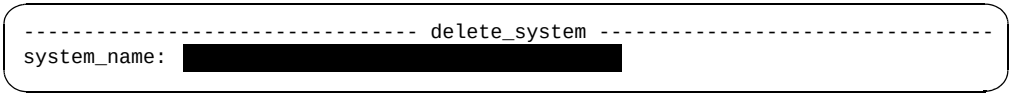
For information about the `new_modules.table` file, see “Using the New Modules Configuration-Table File” on page 3-6. See also the `add_module` command description in this chapter.

# delete_system *Privileged*

## Purpose

The `delete_system` command enables VOS on the current module to delete the specified system. As a result, the current module on the current system no longer recognizes the specified system. On a multimodule system, you must issue this command on each module that should no longer recognize the specified system.

## Display Form




## Command-Line Form

`delete_system` *`system_name`*

## Arguments

- *`system_name`* **Required**
  The system to be deleted. Communications with the deleted system are terminated abruptly. Therefore, take the necessary precautions regarding active user processes before deleting it (for example, notify all users).

## Explanation

You can use this command to correct an error resulting from an incorrect `add_system` command or to delete, for the current bootload, a system whose information in the appropriate systems configuration-table file has changed.

## Examples

The following command deletes the system `%admin` from VOS on the current module (`#m20` in the system `%mfg`).

```
delete_system admin
```

## Related Information

For information about the systems configuration-table files, see ‘‘Using the New Systems Configuration-Table File’’ on page 4-6. See also the `add_system` and `configure_systems` command descriptions in this chapter.

# osl_admin *Privileged*

## Purpose

The osl_admin command enables you to perform certain administrative tasks on an OSL network and to monitor the network.

## Display Form

```
----------------------------------- osl_admin ----------------------------------
-request_line:
-quit:          no
```

## Command-Line Form

```
osl_admin [-request_line request]
          [-quit]
```

## Arguments

- -request_line *request*

  The value of *request* must be one or more osl_admin requests. You specify a request by using the name of the request and any of its arguments. Enclose multiple requests in apostrophes (') and separate them with semicolons (;). In the command-line form, you must also enclose in apostrophes a request that is followed by one or more arguments. The following requests are available.

```
adjust_saved_trace          merge_trace_buffers
compare_configuration       resize_trace_buffer
create_trace_buffer         restart_tracing
disable_destination         save_trace_buffer
display_trace               set_default_trace_flags
enable_destination          set_default_trace_size
get                         set_monitoring
help                        set_parameters
quit                        set_trace_flags
reset_port                  sleep
list_saved_traces           status
match
```

After all specified requests have finished executing, your process remains at the `osl_admin` request level, indicated by the `osl_admin:` prompt, if you omitted the `-quit` argument. Your process returns to command level if you specify the `-quit` argument.

If you do not specify a value for `-request_line` and you omit the `-quit` argument, your process remains at the `osl_admin` request level.

- `-quit` CYCLE
  Returns your process to command level after requests specified in the `-request_line` argument have finished executing. If you omit this argument, your process remains at the `osl_admin` request level after specified requests have finished executing.

## Explanation

The `osl_admin` command and its requests enable you to perform certain administrative tasks on an OSL network. You can use the command interactively on the command line or in a command macro such as the `modules_start_up.cm` file. For VOS to execute at each reboot the `osl_admin` commands in the `module_start_up.cm` file, you must activate these commands by deleting the ampersand and the space preceding each command.

Several requests have the argument *`destination`*. Some requests also have the argument `-from`. For both arguments, you specify *`module_name`* as a value, which enables you to specify a module in the system other than the module that is executing the `osl_admin` command. The module that you specify for the *`destination`* argument receives the action of the request. The module that you specify for the `-from` argument originates the action of the request.

For additional information about requests of the `osl_admin` command, see “The OSL Administrative Tool `osl_admin`” on page 1-11 as well as the request descriptions in “The OSL Administration Requests” on page 6-27.

## The OSL Administration Requests

The following sections describe the requests of the `osl_admin` command.

## The `adjust_saved_trace` Request

The `adjust_saved_trace` request enables you to adjust the time stamp on trace entries. You can more easily compare traces when the times on them are equal. The `adjust_saved_trace` request enables you to adjust the times so that they are equal.

## Display Form

```
------------------------------ adjust_saved_trace ------------------------------
saved_trace: [redacted]
-index:
-hours:
-minutes:
-seconds:
-jiffies:
```

## Command-Line Form

```
adjust_saved_trace saved_trace
        [-index value]
        [-hours hours]
        [-minutes minutes]
        [-seconds seconds]
        [-jiffies jiffies]
```

## Arguments

- *`saved_trace`* **Required**
  The name of the saved trace file that you want to adjust.

- `-index` *`value`*
  The index value of the trace record whose time stamp you are adjusting.

- `-hours` *`hours`*
  The hour of the time stamp adjustment.

- `-minutes` *`minutes`*
  The minutes of the time stamp adjustment.

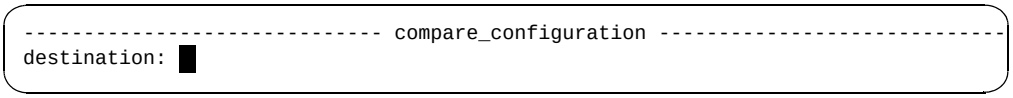
- `-seconds` *`seconds`*
  The seconds of the time stamp adjustment.

- `-jiffies` *`jiffies`*
  The jiffies of the time stamp adjustment (one jiffy is 1/65,536 of a second).

## The compare_configuration Request

The compare_configuration request compares an OSL network as configured on the current module with the OSL network as configured on other modules.

## Display Form




## Command-Line Form

compare_configuration *destination*

## Arguments

- *destination* **Required**

  A module name that you specify to compare the current module with the specified destination module. By default, the request compares the current module with each of the other modules in the system.

## Explanation

If you specify a value for *destination*, the compare_configuration request compares only the current module with the specified destination module. For example, in a system of modules m1 (the current module), m2, and m3, compare_configuration compares, by default, m1 with m2, and m1 with m3, but not m2 with m3. The request compare_configuration m3 compares m1 (the current module) with m3.

The request compares configurations by checking values within the kernel that VOS derives from the max_open_servers, station_number, and base_port fields of the new_modules.tin file. The request also checks for modules that may be defined in one file but not in another.

## Examples

In the first example from system %osl, m1 is running older versions of VOS and OSL that do not support the osl_admin command, m2 is the current module, m3 is offline,

and m4 is running versions of VOS and OSL that support the osl_admin command. This system is configured correctly.

```
osl_admin: compare_configuration

Module %osl#m1 does not support this function.

Module %osl#m3 is offline.

Comparing %osl#m2 with %osl#m4... No differences found.
```

In the second example, the new_modules.tin file on %osl has been modified so that m2 and m4 each derive a different number of ports for SOSL Net driver on m4. (VOS derives the number of ports from values specified for the base_port and max_open_servers fields of the new_modules.tin file.) On m2, VOS derives 10; on m4, VOS derives 5. This type of error is common. (In the output, max_connects refers to the number of ports.)

```
osl_admin: compare_configuration

Module %osl#m1 does not support this function.

Module %osl#m3 is offline.

Comparing %osl#m2 with %osl#m4... Differences found:
max_connects for m4 (%osl#m2 = 10, %osl#m4 = 5) does not
match.
```

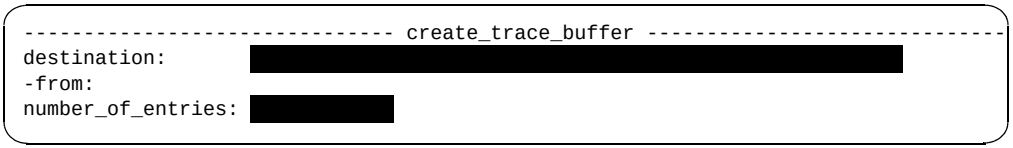
For the final example, the command delete_module m1 is issued on m2, which deletes m1 from the OSL configuration of m2.

```
osl_admin: compare_configuration

Module %osl#m3 is offline.

Comparing %osl#m2 with %osl#m4... Differences found:
Modules defined on %osl#m4 but not on %osl#m2: m1
```

## The `create_trace_buffer` Request

The `create_trace_buffer` request enables you to create a trace buffer on a module.

## Display Form




## Command-Line Form

```
create_trace_buffer destination
        [-from module_name]
        number_of_entries
```

## Arguments

- *`destination`* **Required**
  The module that you specify for *`destination`* creates a trace buffer only when it receives requests from or sends requests to the module specified by the `-from` argument (which is, by default, the current module).

- `-from` *`module_name`*
  The module that you specify for the `-from` argument originates the action of the request. The default value is the current module. When this module sends requests to or receives requests from the *`destination`* module, the *`destination`* module creates the trace buffer.

- *`number_of_entries`* **Required**
  The number of entries in the array that is the trace buffer.

## Explanation

When tracing is enabled, OSL creates trace buffers only on demand. The module specified by the *`destination`* argument creates a trace buffer only when it receives requests from or sends requests to the module specified by the `-from` argument.

You can use this request in the `module_start_up.cm` file if you want a specific destination module to have trace buffers of a size that is different from the default size, and trace-flags settings that are different from the default settings.

## The disable_destination Request

The disable_destination request temporarily stops sending outgoing requests from one module to another.

## Display Form

```
------------------------------ disable_destination ------------------------------
destination:
-from:
```

## Command-Line Form

```
disable_destination destination
        [-from module_name]
```

## Arguments

- *destination* **Required**

  The name of the module that you want to disable from the OSL network. The request disables this module from receiving outgoing requests from the current module (by default) or from the module that you specify with the -from argument.

  Use one of the following formats to specify *destination*:

  - %*system#name* (for example, %os1#m2)
  - *N-N*, which is the system number followed by the module number (for example, 43-22)
  - %*system* (for example, %os3)
  - *N-0*, which is the system number and a "0" for the module number (for example, 43-0)

- -from *module_name*

  The destination module that will be disabled from receiving outgoing requests from the module that you specify with *module_name*. The default value for *module_name* is the current module; by default, the disable_destination request disables the destination module from receiving outgoing requests from the current module.

## Explanation

The disable_destination request stops outgoing requests from the current module (by default) to the destination specified by *destination*. For example, disable_destination m1 stops sending outgoing requests from the current module to m1. The current module can resume sending outgoing requests to the destination

module when you issue the `enable_destination` request, when you reboot the destination module, or when you delete the destination module from the OSL network (using the `delete_module` command) and then re-add it (using the `configure_modules` command).

NOTES

1. If you issue this request on a module that is already disabled, the request returns an error message.
2. This command disables only outgoing requests. Incoming requests from the disabled module to an active module continue to be processed.

To disable a remote module from a bridge module, you must specify *`destination`* as `%sys#module`. You cannot specify a star name. You cannot disable a remote system from a non-bridge module.

If you specify *`module_name`* for the `-from` argument, the request tells *`module_name`* to stop sending outgoing requests to the module *`destination`*. For example, the following request stops `m2` from sending outgoing requests to `m1`:

```
disable_destination m1 -from m2
```

## The `display_trace` Request

The `display_trace` request displays the contents of a trace buffer.

## Display Form

```
--------------------------------- display_trace ---------------------------------
destination: 
-from:
-from_path:
-tid:
-follow:      no
-all:         no
-last:        30
-client:      yes
-server:      yes
```

## Command-Line Form

```
display_trace destination
          [-from module_name]
          [-from_path path_name]
          [-tid number]
          [-follow]
          [-all]
          [-last number]
          [-no_client]
          [-no_server]
```

## Arguments

- *destination* **Required**
  The name of the module whose trace buffer is displayed. The default value is the current module.

- `-from` *module_name*
  The name of the module that originates the request. The default value is the current module.

- `-from_path` *path_name*
  The name of the module that contains the trace buffer.

- `-tid` *number*
  The transaction ID of trace entries that are displayed.

- `-follow` CYCLE
  The request continuously monitors the trace and displays new entries.

- `-all` CYCLE
  The request displays all trace entries in the trace buffer.

- `-last` *`number`*
  The request displays the last trace entry in the trace buffer.

- `-no_client` CYCLE
  The request displays only client entries in the trace buffer.

- `-no_server` CYCLE
  The request displays only server entries in the trace buffer.

## The enable_destination Request

The enable_destination request enables a module that had been previously disabled from sending outgoing requests to resume sending those requests.

## Display Form

```
------------------------------ enable_destination ------------------------------
destination:
-from:
-activate_now: no
```

## Command-Line Form

enable_destination *destination*
[-from *module_name*]
[-activate_now]

## Arguments

- *destination* **Required**

  The name of the module that you want to enable to resume communications with an OSL network. The request enables this module to resume receiving outgoing requests from the current module (by default) or from the module that you specify with the -from argument.

  Use one of the following formats to specify *destination*:

  - %*system*#*name* (for example, %os1#m2)
  - *N*-*N*, which is the system number followed by the module number (for example, 43-22)
  - %*system* (for example, %os3)—You can specify this format only if you do not specify the -activate_now argument.
  - *N*-0, which is the system number and a "0" for the module number (for example, 43-0)—You can specify this format only if you do not specify the -activate_now argument.

  If you specify -activate_now, *destination* applies to all defined matching module names. If you do not specify -activate_now, *destination* applies to all matching module names currently in use, according to the module specified in the -from argument.

- -from *module_name*

  The destination module will be enabled to resume receiving outgoing requests from the module that you specify with *module_name*. The default value for

*module_name* is the current module; by default, the enable_destination request enables the destination module to resume receiving outgoing requests from the current module. If you specify the -activate_now argument, -from can only specify the current module.

► -activate_now CYCLE

Sends messages to *destination*, thus ensuring that one or more active connections to *destination* exists. If you do not specify -activate_now, and OSL has not already accessed *destination* at least once from this module, specifying the enable_destination request results in an Object not found error. You cannot specify -activate_now if the -from argument specifies a module other than the current module. If the current module is not a bridge module and *destination* is on a different system, specifying -activate_now also activates the bridge module.

## Explanation

The enable_destination request enables the current module (by default) to resume sending outgoing requests to the module specified by *destination*. For example, enable_destination m1 enables the current module to resume sending outgoing requests to m1. If you specify *module_name* for the -from argument, the request tells *module_name* to resume sending outgoing requests to the module *destination*. For example, the following request enables m2 to resume sending outgoing requests to m1: enable_destination m1 -from m2

NOTE

If you issue the enable_destination request on a module that is already enabled, the request returns an error message. However, the request still activates the module.

## The get Request

The get request returns information about the state of the OSL network or its configuration. The request displays information on the command line, or it returns the information through the command function (command_status). You can use this request in macros (for example, the module_start_up.cm file).

## Display Form

```
---------------------------------- get ---------------------------------
parameter:       disabled
-destination:
-from:
-display:        yes
-command_status: yes
```

## Command-Line Form

get *parameter*
[-destination *module_name*]
[-from *module_name*]
[-no_display]
[-no_command_status]

## Arguments

- *parameter* CYCLE

  The *parameter* argument enables you to specify the type of information that the get request retrieves. Values are the following:

  - bridge_id returns the module_id of the bridge module. If you specify bridge_id, the get request ignores the -destination argument.
  - bridge_module returns the module number of the bridge module. If you specify bridge_module, the get request ignores the -destination argument.
  - bridge_name returns the module name of the bridge module. If you specify bridge_name, the get request ignores the -destination argument.
  - disabled states whether the specified destination module is disabled. (This parameter does not cause a request to be sent to the specified destination. It only returns the current OSL state of the destination module.) The value disabled returns 1 if the destination has been disabled and 0 if it has not been disabled. If you specify disabled, you must also specify the -destination argument.

- `i_am_bridge` states whether the current module is the bridge module by default. You also specify the `-destination` argument to determine whether the destination is the bridge module.
- `max_servers` returns the `max_open_servers` value of the `new_modules.tin` file. For typical configurations, the `max_open_servers` values in the `new_modules.tin` and `new_backbone_systems.tin` files are equal. If, however, the values are different, the `max_open_servers` value in the `new_modules.tin` file should be greater than (or equal to) the `max_open_servers` value in the `new_backbone_systems.tin` file.

  If you specify the `-destination` argument, the request returns the value for that destination. If you do not specify the `-destination` argument, this parameter returns the value for the current module.

- `num_waiting` returns the number of processes that OSL is currently blocking to the specified destination module. For example, if the `max_open_servers` value for the target module is 2, and ten processes are sending requests to that module, this parameter returns 8. If you specify `num_waiting`, you must also specify the `-destination` argument.
- `online` states whether the specified destination module is online. (This parameter does not cause a request to be sent to the specified destination. It only returns the current OSL state of the destination module.) If you specify `online`, you must also specify the `-destination` argument.
- `servers_idle` returns the number of `osl_server` processes that are currently idle. If you specify `servers_idle`, the `get` request ignores the `-destination` argument.

► `-destination` *`module_name`*
The name of the destination module about which you want information. The destination module is also referred to as the target module. The default value is the current module.

► `-from` *`module_name`*
The name of the module that originates the request. The default value is the current module.

► `-no_display` CYCLE
The value `yes` (the default) enables the request to display the value that the *`parameter`* argument returns.

► `-no_command_status` CYCLE
The value `yes` (the default) enables the request to return information to the `(command_status)` function.

## The help Request

The help request lists information about osl_admin requests.

## Display Form

```
---------------------------------- help ---------------------------------
-match:
```

## Command-Line Form

help [-match *request*]

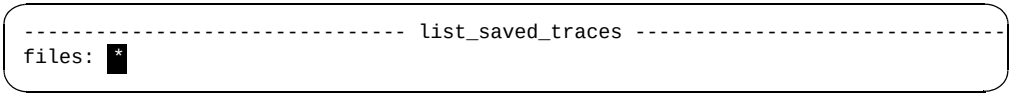
## Arguments

- -match *request*
  Displays all of the request names within the osl_admin subsystem that contain the string *request*. If you omit this argument, the request displays all osl_admin requests.

## The `list_saved_traces` Request

The `list_saved_traces` request displays a list of the names of the saved trace files.

### Display Form




### Command-Line Form

`list_saved_traces` [*files*]

### Arguments

- *files*
  The name (or names) of the file (or files) that are displayed.

### Explanation

Before the `restart_tracing` request restarts tracing, it searches for trace buffers that have been stopped automatically. If `restart_tracing` finds such trace buffers, it saves them to a file. The `list_saved_traces` request displays a list of the names of these saved trace files.

## The `match` Request

The `match` request enables you to restrict the output displayed by the next request to that which matches the specified string. You must issue this request before the request for which you want to selectively display output.

## Display Form

```
---------------------------------- match ----------------------------------
match_string:
-and:
-or:
-min_lines:   1
-and_first:   yes
```

## Command-Line Form

```
match [match_string]
      [-and string]
      [-or string]
      [-min_lines number]
      [-no_and_first]
```

## Arguments

- *`match_string`*
  The character string to be matched. If you omit *`match_string`*, the default matches everything and therefore displays the entire output of the next request. Note that *`match_string`* is a caseless argument.

- `-and` *`string`*
  Displays lines that contain this string **and** the match string.

- `-or` *`string`*
  Displays lines that contain this string **or** the match string.

- `-min_lines` *`number`*
  Displays the specified number of lines starting with the line in which a match was found. If another match is found on a line fewer than *`number`* lines from the previous match, the `match` request restarts the line count. You cannot specify a value of less than `1`. The default value is `1`.

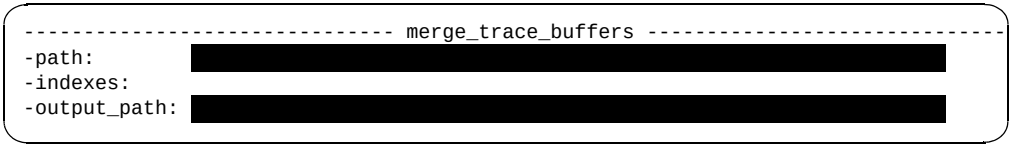
- `-no_and_first` CYCLE
  Specifies the order of precedence when both the `-and` *`string`* and `-or` *`string`* arguments are specified. If you specify `yes`, the request matches output lines containing both the *`match_string`* and `-and` *`string`* values, or just `-or` *`string`*. If you specify `no`, the request matches output lines containing either the

*match_string* or -or *string* values, and also containing -and *string*. The default value is yes.

## The `merge_trace_buffers` Request

The `merge_trace_buffers` request merges trace buffers into one file.

## Display Form




## Command-Line Form

```
merge_trace_buffers [-path path_name]
                    [-indexes number(s)]
                    [-output_path path_name]
```

## Arguments

- `-path` *`path_name`*
  The name of a trace buffer into which additional indexes are merged.

- `-indexes` *`number(s)`*
  The number(s) of one or more indexes that are merged into the output file.

- `-output_path` *`path_name`*
  The name of the file that contains the merged trace buffers.

## The `quit` Request

The `quit` request exits the `osl_admin` subsystem and returns your process to command level.

## Display Form

```
---------------------------------- quit ----------------------------------
No arguments required. Press ENTER to continue.
```

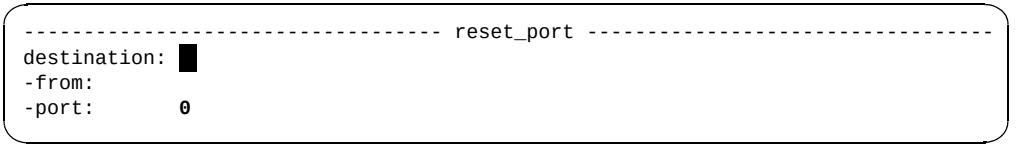
## Command-Line Form

```
quit
```

## The `reset_port` Request

The `reset_port` request enables you to reset connections to a destination module after OSL recovers from a network outage.

## Display Form




## Command-Line Form

```
reset_port destination
          [-from module_name]
          [-port port_number]
```

## Arguments

- *`destination`* **Required**
  The name of the module whose ports you want to reset. The request enables this module to reconnect to an OSL network. The default value is the current module.

- `-from` *`module_name`*
  The name of the module that you want to originate the request. The default value is the current module.

- `-port` *`port_number`*
  The number of an individual port that you want to reset. The value `0` (the default) resets all ports on the destination module.

## Explanation

The `reset_port` request enables you to redistribute network connections after a network returns to service. If, for example, OSL is running over two networks, `net_a` and `net_b`, and `net_a` becomes disabled, OSL recovers by moving all `net_a` connections to `net_b`, and modules communicate with each other over one network, `net_b`. The `reset_port` request enables you to redistribute network connections to both networks after `net_a` returns to service.

The `reset_port` request instructs OSL to set a `reset` bit in its per-connection structure. When this bit is set, OSL closes the TCP socket the next time a request completes on that connection, and then clears the bit.

## Examples

In this first example, one network is down but network traffic is still being generated. All connections have moved to the 192.168.101.3 network.

```
osl_admin: status m3

Status for %osl#m3:

Destination is online.
6 RPC connections:
  6 connections on 192.168.101.3

No FQP connections are active.
```

When the other network returns to service, the OSL administrator issues the request `reset_port m3`.

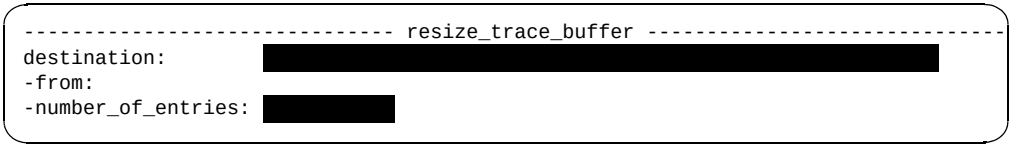
```
osl_admin: reset_port m3
osl_admin: status m3

Status for %osl#m3:

Destination is online.
6 RPC connections:
  3 connections on 192.168.101.3
  3 connections on 192.168.100.3

No FQP connections are active.
```

In this case, the load was redistributed quickly (in less than one second) because `m3` was receiving a lot of traffic. When a module is receiving no traffic, you need to issue the request six times.

## The `resize_trace_buffer` Request

The `resize_trace_buffer` request enables you to change the number of entries in an existing trace buffer.

## Display Form




## Command-Line Form

```
resize_trace_buffer destination
         [-from module_name]
         -number_of_entries number
```

## Arguments

- *`destination`* **Required**
  The name of the module whose buffers you are resizing. The default value is the current module.

- `-from` *`module_name`*
  The name of the module that originates the request. The default value is the current module.

- `-number_of_entries` *`number`* **Required**
  The number of entries for the buffer.

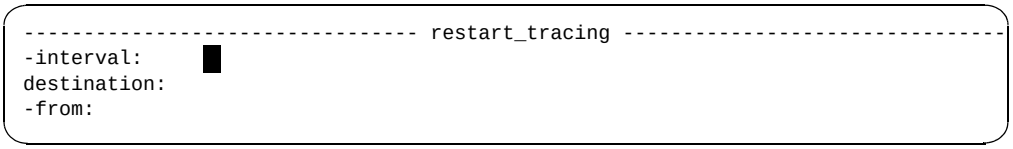
## Explanation

When you resize a buffer, the contents of the old buffer are lost; they are not copied to the new, resized buffer.

## The `restart_tracing` Request

The `restart_tracing` request searches for buffers that have been automatically stopped, saves them to a file, and restarts tracing.

## Display Form




## Command-Line Form

```
restart_tracing [-interval seconds]
         destination
         [-from module_name]
```

## Arguments

- `-interval` *`seconds`*
  Specifies that the request run at the specified interval. If auto-stop is enabled, a process should be started in `module_start_up.cm` that runs `restart_tracing` at intervals.

- *`destination`* **Required**
  The name of the module whose trace buffers have been automatically stopped. The default value is the current module.

- `-from` *`module_name`*
  The name of the module that originates the request. The default value is the current module.

## Explanation

The `restart_tracing` request restarts tracing to a destination module where tracing has been stopped automatically. By default, tracing is enabled and each trace buffer contains 500 entries. Also by default, OSL automatically halts tracing if it marks another module offline.

If you do not specify a value for *`destination`*, the request searches through all modules for trace buffers that have been automatically stopped. If you specify a value for *`destination`*, the request searches only the specified module.

When the command finds a destination that has been automatically stopped, it saves the trace buffer to a file and restarts tracing to that destination. This information is saved

to the file `osl_trace.`*`module`*`.(date)` in the current directory, where *`module`* is the module ID of the destination module.

## Examples

For each module, the `status` request displays one of the following lines to provide tracing information:

```
Tracing is enabled with auto-stop.
Tracing is enabled.
Tracing has been auto-stopped.
Tracing is not enabled.
```

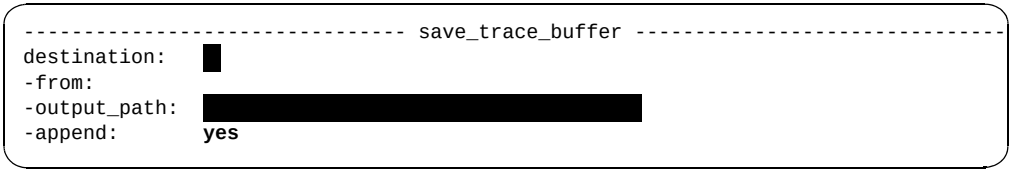
The following output shows two of these messages.

```
osl_admin: set_trace_flags m1 -no_tracing; status m1

Status for %osl#m1:

  Destination is online.

  Tracing is not enabled.

  500 entries allocated for this trace buffer

  1 RPC connection:
   1 connection on 192.168.101.1

  No FQP connections are active.

osl_admin: set_trace_flags m1 -tracing; status m1

Status for %osl#m1:

  Destination is online.

  Tracing is enabled.
  500 entries allocated for this trace buffer

  1 RPC connection:
   1 connection on 192.168.101.1

  No FQP connections are active.
```

## The `save_trace_buffer` Request

The `save_trace_buffer` request enables you to save a trace buffer to a file.

## Display Form




## Command-Line Form

```
save_trace_buffer destination
        [-from module_name]
        [-output_path path_name]
        [-no_append]
```

## Arguments

- *destination* **Required**
  The name of a module whose trace buffer you want to save. The default value is the current module.

- `-from` *module_name*
  The name of a module that you want to originate the request. The default value is the current module.

- `-output_path` *path_name*
  The name of the file that will contain the trace buffer.

- `-no_append` CYCLE
  The value `yes` (the default) specifies that the trace buffer will be appended to another file.

## The `set_default_trace_flags` Request

The `set_default_trace_flag` request sets the values for the flags of default tracing, which determines the default tracing behavior for OSL.

## Display Form

```
------------------------------ set_default_trace_flags ---------------------------
-tracing:       yes
-auto_stop:     yes
-trace_brief:   no
-trace_client: yes
-trace_server: yes
```

## Command-Line Form

```
set_default_trace_flags [-no_tracing]
                        [-no_auto_stop]
                        [-trace_brief]
                        [-no_trace_client]
                        [-no_trace_server]
```

## Arguments

- `-no_tracing` CYCLE
  The value `yes` (the default) specifies that tracing is enabled by default.

- `-no_auto_stop` CYCLE
  The value `yes` (the default) specifies that OSL stops tracing to a destination when it marks that destination offline.

- `-trace_brief` CYCLE
  Reduces the number of entries each transaction uses. Use this argument only to gather performance data.

- `-no_trace_client` CYCLE
  The value `yes` (the default) enables tracing of outgoing traffic from the current module to a remote module.

- `-no_trace_server` CYCLE
  The value `yes` (the default) enables tracing of incoming traffic from other modules to the current module.

## Explanation

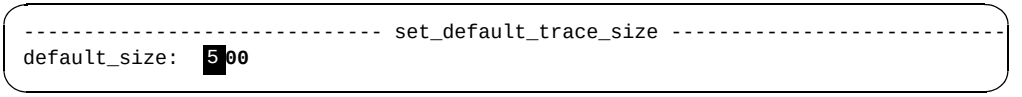
You can use the `set_default_trace_flags` and `set_default_trace_size` requests in `module_start_up.cm` to control whether or not OSL uses tracing and how much memory it consumes.

You can disable automatic tracing with the `set_default_trace_flags` or `set_trace_flags` request. By default, tracing is enabled and each trace buffer contains 500 entries. Also by default, OSL automatically stops tracing when it marks another module offline. The `restart_tracing` request searches for buffers that have been automatically stopped, saves them to a file, and restarts tracing.

## The `set_default_trace_size` Request

The `set_default_trace_size` request sets the default number of entries in each trace buffer.

## Display Form




## Command-Line Form

`set_default_trace_size` *`default_size`*

## Arguments

- *`default_size`* **Required**
  Specifies the default number of entries in each trace buffer. The default value is `500`.

## Explanation

You can use the `set_default_trace_size` request and the `set_default_trace_flags` request in `module_start_up.cm` to control whether or not OSL uses tracing and how much memory it consumes.

Because `set_default_trace_size` has no effect on buffers that have already been allocated, this request should be run in `module_start_up.cm` only before the `osl_net_driver` is loaded.

## The set_monitoring Request

The set_monitoring request allows you to monitor OSL connections and to expedite shadow-notify requests. **This is a privileged request.**

## Display Form

```
 ------------------------------ set_monitoring ------------------------------
destination:          [redacted]
-from:
-notify_mode:         normal
-monitoring:          no
-monitor_interval:    5
-min_connects:        3
-max_txn_time:        240
-activate_now:        no
```

## Command-Line Form

```
set_monitoring destination
        [-from module_name]
        [-notify_mode]
        [-monitoring]
        [-monitor_interval interval]
        [-min_connects number_of_ports]
        [-max_txn_time time_allowed]
        [-activate_now]
```

## Arguments

- *destination* **Required**

  The name of the destination module that you want to monitor. The destination module is also referred to as the *target module*. Use one of the following formats to specify *destination*:

  - a module star name (for example, %os#*)
  - %*system*#*name* (for example, %os1#m2)
  - *N*-*N*, which is the system number followed by the module number (for example, 43-22)
  - %*system* (for example, %os3)—You can specify this format only if you do not specify the -activate_now argument.
  - *N*-0, which is the system number and a “0” for the module number (for example, 43-0)—You can specify this format only if you do not specify the -activate_now argument.

If you specify `-activate_now`, *`destination`* applies to all defined matching module names. If you do not specify `-activate_now`, *`destination`* applies to all matching module names currently in use, according to the module specified in the `-from` argument.

- `-from` *`module_name`*
  The module whose destination(s) are affected by the `set_monitoring` command. If OSL cannot determine the current settings (that is, if *`destination`* does not indicate a specific module), it uses the current global default values (that is, those values set as initial system defaults or set by the `set_parameters` command) for *`module_name`*.

- `-notify_mode` CYCLE
  Expedites shadow-notify requests. *Shadow-notify requests* are requests sent over OSL to update the state of a remote event. Each remote event has a *shadow event* that exists on the local module.

  OSL does not start the daemon process required for expediting shadow-notify requests until it is needed (for example, when monitoring is turned on). Setting `-notify_mode` to `expedited` starts the monitor either when *`destination`* is initially established or the next time any activity is directed to *`destination`*. If you specify the `normal` value (the default), OSL does not expedite shadow-notify requests.

- `-monitoring` CYCLE
  Specifies whether OSL monitors connections. By default (the value `no`), OSL does not monitor connections, and it ignores any value specified for `-monitor_interval`.

- `-monitor_interval` *`interval`*
  Specifies the amount of time, in seconds, that a probe message is sent to *`destination`*. The *`interval`* value must be in the range 1 to 3600 if `-monitoring` is set to `yes`. The default value is `10`.

- `-min_connects` *`number_of_ports`*
  Indicates the number of connections established (that is, the number of ports attached) before one is reused. OSL makes additional new connections greater than *`number_of_ports`* only when needed (for example, when a client request is made and all established connections are in use). If you specify `-activate_now`, OSL makes *`number_of_ports`* connections if less than *`number_of_ports`* connections are currently established. Otherwise, *`number_of_ports`* connections are established as requests are made to the relevant destination. The *`number_of_ports`* value must be in the range 2 to 32 and cannot be greater than the maximum configured connections. (The `max_open_servers` field in the `new_modules.tin` file and the `-max_open_servers` argument of the

add_system or add_module command determine the maximum configured connections.) By default, *number_of_ports* is 3.

- -max_txn_time *time_allowed*
  Specifies, in minutes, the maximum time allowed for any transaction. If you do not specify -max_txn_time, a transaction can continue for no longer than four minutes after the remote module has sent a positive response. By default, *time_allowed* is 30; allowed values are 5 to 30.

- -activate_now CYCLE
  Sends messages to *destination*, thus ensuring that one or more active connections to *destination* exists. If you do not specify -activate_now, and OSL has not already accessed *destination* at least once from this module, specifying the set_monitoring request results in an Object not found error. You cannot specify -activate_now if the -from argument specifies a module other than the current module. If the current module is not a bridge module and *destination* is on a different system, specifying -activate_now also activates the bridge module.

## Explanation

The set_monitoring request controls whether monitoring is on or off, and whether shadow-notify requests are expedited. If you do not specify this request, OSL uses the system defaults, as specified by the set_parameters request, for these values for each destination module.

Monitoring allows OSL to quickly recognize when an interface is offline. It also allows OSL to quickly reconnect, using a different interface when one interface goes offline.

A monitor process (osl_monitor_*n*) sends requests at regular intervals that are handled directly by the operating system running on the remote module (not via a server process, which may need to be scheduled), thereby assuring that OSL can access interfaces on the module. If OSL cannot access any interfaces on a module, monitoring requests fail, and OSL recognizes more quickly that the module is offline. This prevents the continuation of existing requests that might otherwise wait for up to four minutes to be recognized as failed requests.

NOTE

A destination is considered accessed after OSL has established a connection to it, even if communication with the destination is interrupted (for example, if the interface goes down, or if the destination module is shut down or reconfigured).

## Examples

This section discusses the following topics:



### Forcing Access to a Destination

Typically, `osl_admin` requests operate only on destination modules that OSL has already accessed. The `-activate_now` argument forces OSL to access a destination. When you use this argument, the request sets monitoring parameters for a specific destination or set of destinations, and it also establishes the number of connections specified in the `-min_connects` argument. However, you can specify `-activate_now` only if the module specified in the `-from` argument is not specified or is the same as the current module.

In the following example, `set_monitoring` forces OSL to establish six active connections to `%rsys#mod`. If OSL had previously accessed `#mod`, the request uses the current monitoring parameters (as specified by the `set_parameters` request); if OSL had not previously accessed `#mod`, the request uses the system defaults.

```
set_monitoring #mod -min_connects 6 -activate_now
```

If the destination system is not the current system, the monitoring parameters apply to the bridge module as well as to the current module's access to the bridge (assuming that the current module is not the bridge). If you do not specify `-activate_now`, the destination can be a system name or a module name. You specify a system name symbolically, with a leading `%` (for example, `%rsys`). OSL expands a name such as *`mod`* (with no `%` or `#` character) to *`%sys#mod`*, where *`%sys`* is the current system. You specify a numeric system name as *`N`*`-0`, where *`N`* is the system number.

### Setting the Maximum Transaction Time

In the following example, `set_monitoring` changes the maximum transaction time from four minutes to five minutes for any transaction started on the current module, that is directed to any module on `%rsys`.

```
set_monitoring %rsys -max_txn_time 5
```

The preceding example requires that the current module be either a bridge that has established %rsys as a destination or a module that has communicated with such a bridge. If the current module is a module that has communicated with such a bridge, the preceding request affects the value of the -max_txn_time argument for the transactions between the current module and the bridge, as well as between the bridge and the remote system.

### Establishing Connections

You can use either the set_monitoring request or the enable_destination request to establish one or more connections from the current module to %rsys#rmod, possibly establishing a connection from the current module to the bridge (if needed).

- The set_monitoring request uses alternating interfaces to establish the number of connections specified in -min_connects. For example:

```
set_monitoring %rsys#rmod -activate_now
```

- The enable_destination request establishes at least two connections. For example:

```
enable_destination %rsys#mod -activate_now
```

### Monitoring a Module

In the following example, if #this_mod is the current module, the module is monitored from %rsys#rmod. This request affects #rmod as well as the bridge on %rsys if #rmod is not a bridge. Because you cannot specify -activate_now in this situation, an error occurs if #rmod is not the bridge and has not yet communicated with the bridge. This situation does not affect monitoring attributes of the bridge on the system containing #this_mod.

```
set_monitoring #this_mod -from %rsys#rmod -monitoring on
```

### Setting Monitoring for All Established Destinations

The following example sets monitoring for all established destinations of %rsys#rmod. If #rmod has not established any destinations, the request returns a warning.

```
set_monitoring * -from %rsys#rmod -monitoring on
    -monitor_interval 5
```

### Turning Monitoring Off for Specific Modules

The following example turns off monitoring from the current system to all modules that the current module has already accessed and whose names begin with the string mo. If mod2 is a module on the system that has not yet been accessed, it will have the default monitoring attributes when first accessed.

```
set_monitoring #mo* -monitoring off
```

### Turning Monitoring Off for All Configured Modules

The following example establishes a connection and turns off monitoring from the current module to all other configured modules. If the current module has never accessed one or more configured modules, and cannot access them at this time, the request returns a warning but does not change monitoring attributes. Defaults are used when the module is first successfully accessed.

```
set_monitoring * -monitoring off -activate_now
```

### Changing the Default Monitoring for the Current Module

The following example changes the default monitoring for the current module. This request has no effect on existing destinations; it only affects new destinations that have never been accessed.

```
set_parameters -monitoring off
```

## The `set_parameters` Request

The `set_parameters` request allows you to set system-wide tuning values. **This is a privileged request.**

## Display Form

```
 ------------------------------ set_parameters ------------------------------
-from:
-connect_time:       4
-xfer_time:          5
-xfer_retries:       13
-notify_mode:        normal
-monitoring:         no
-monitor_interval:   0
-min_connects:       3
-max_txn_time:       240
-backoff_time:       60
-reestablish_time:   60
```

## Command-Line Form

```
set_parameters
          [-from module_name]
          [-connect_time connect_time]
          [-xfer_time transfer_time]
          [-xfer_retries num_retries]
          [-notify_mode]
          [-monitoring]
          [-monitor_interval interval]
          [-min_connects number_of_ports]
          [-max_txn_time time_allowed]
          [-backoff_time wait_time]
          [-reestablish_time timeout]
```

## Arguments

- `-from` *`module_name`*
  The module that you specify for the `-from` argument originates the action of the request. The default value is the current module. If `set_parameters` cannot determine the current settings, it uses the current global default values (that is, those values set as initial system defaults or set by a previous `set_parameters` command) for *`module_name`*.

- `-connect_time` *`connect_time`*
  Specifies the length of time, in seconds, that OSL attempts to connect to each configured interface. OSL attempts to connect to each configured interface twice:

first, for one second, and then, for *connect_time*. By default, *connect_time* is 4; allowed values are 1-60.

- -xfer_time *transfer_time*
  Specifies the length of time, in seconds, that OSL waits for an ongoing request to complete. If -monitoring is set to yes and the modules being monitored are running VOS Release 15.1.0 (or later), OSL attempts to reconnect after waiting *transfer_time* seconds, with no retries for a monitoring failure and the number of retries specified in -xfer_retries *num_retries* for any other failure. By default, *transfer_time* is 5; allowed values are 1-60.

- -xfer_retries *num_retries*
  Specifies the number of times that OSL attempts to reconnect to *module_name*. By default, *num_retries* is 13; allowed values are 1-240.

- -notify_mode CYCLE
  Expedites shadow-notify requests. OSL does not start the daemon process required for expediting shadow-notify requests until it is needed (for example, when monitoring is turned on). Setting -notify_mode to expedited starts the monitor either when *destination* is initially established or the next time any activity is directed to *destination*. If you specify the normal value (the default), OSL does not expedite shadow-notify requests.

- -monitoring CYCLE
  Specifies whether OSL monitors connections. By default (the value no), OSL does not monitor connections, and it ignores any value specified for -monitor_interval.

- -monitor_interval *interval*
  Specifies the amount of time, in seconds, that a probe message is sent to *destination*. The *interval* value must be in the range 1 to 3600 if -monitoring is set to yes. The default value is 10.

- -min_connects *number_of_ports*
  Specifies the minimum number of connections established before one is reused for any destination. The *number_of_ports* value must be in the range 2 to 32. By default, *number_of_ports* is 3. See the description of the -min_connects argument of the set_monitoring request for more information.

- -max_txn_time *time_allowed*
  Specifies, in minutes, the maximum time allowed for any transaction. If you do not specify -max_txn_time, a transaction can continue for no longer than four minutes after the remote module has sent a positive response. By default, *time_allowed* is 30; allowed values are 5 to 30.

- `-backoff_time` *`wait_time`*
  Specifies, in seconds, the amount of time that OSL waits before it attempts to reconnect to a module after determining that it is inaccessible. By default, *`wait_time`* is `60`; allowed values are `1` to `3600`.

- `-reestablish_time` *`timeout`*
  Specifies, in seconds, the amount of time that OSL waits to time out after attempting to access an inaccessible module. OSL uses the *`timeout`* value until the module is once again online, overriding the values of `-xfer_time` and `-xfer_retries`. By default, *`timeout`* is `60`; allowed values are `1` to `3600`.

## Explanation

The `set_parameters` request sets the value of system-wide parameters. Some of these parameters control aspects of OSL behavior; changing these parameters changes the behavior immediately. Other parameters control the setting of OSL default values (such as those related to monitoring) that are used when OSL first accesses a remote destination. New values take effect only when OSL first accesses a new destination. You should consider adding `set_parameters` to `module_startup.cm` if you are using it for these purposes, since, at startup, other modules may be accessed and destinations established.

For example, the following line in `module_startup.cm` establishes the defaults for all destination modules that OSL subsequently accesses. Such a line should appear immediately after the OSL driver is loaded and before any remote modules are accessed.

```
osl_admin -request_line 'set_parameters -notify_mode
    expedited' -quit
```

You can assign non-default attributes to a specific destination interactively, as in the following example.

```
osl_admin: set_monitoring  #mod -monitoring on
    -monitor_interval 30
```

The preceding request affects only the destination `#mod` (on the current system). If OSL has not yet accessed `#mod`, OSL returns the `e$object_not_found (1032)` error message for this request.

## Examples

The Explanation in the description of the `set_monitoring` request provides examples that illustrate how to use many of the arguments of the `set_parameters` request.

## The `set_trace_flags` Request

The `set_trace_flags` request sets the tracing behavior for a specified trace buffer that has already been created.

## Display Form

```
---------------------------------- set_trace_flags -------------------------------
destination:   
-from:
-tracing:      no
-auto_stop:    yes
-auto_stopped: no
-trace_brief:  no
-trace_client: no
-trace_server: no
```

## Command-Line Form

```
set_trace_flags destination
                [-from string]
                [-tracing]
                [-no_auto_stop]
                [-auto_stopped]
                [-trace_brief]
                [-trace_client]
                [-trace_server]
```

## Arguments

- *`destination`* **Required**
  The name of a module whose trace flags you want to set.

- `-from` *`string`*
  The name of a module that you want to originate the request. The default value is the current module.

- `-tracing` CYCLE
  The value `no` (the default) specifies that tracing is not enabled.

- `-no_auto_stop` CYCLE
  The value `yes` (the default) enables OSL to automatically stop tracing when it determines another module is offline.

- `-auto_stopped` CYCLE
  The value `no` (the default) enables OSL to display if it has automatically stopped tracing to this destination. Changing this argument has no effect.

- `-trace_brief` CYCLE
  The value `yes` reduces the number of entries each transaction uses. The default value is `no`.

- `-trace_client` CYCLE
  The value `yes` enables tracing of outgoing traffic from the current module to a remote module. The default value is `no`.

- `-trace_server` CYCLE
  The value `yes` enables tracing of incoming traffic from other modules to the current module. The default value is `no`.

## Explanation

You can disable automatic tracing with the `set_trace_flags` or `set_default_trace_flags` request.

If OSL determines that another module is offline and the `-auto_stop` argument is set to `yes`, it stops the tracing to that other module. However, do not set `-auto_stop` to `yes` on a module that stops and starts frequently. If a system is stable, set `-auto_stop` to `yes` so that the CAC can examine the trace when a modules goes offline.

## The sleep Request

The sleep request suspends a process for a specified period of time.

## Display Form

```
---------------------------------- sleep ----------------------------------
-hours:   0
-minutes: 0
-seconds: 0
-until:
-forever: no
```

## Command-Line Form

```
sleep [-hours hours]
      [-minutes minutes]
      [-seconds seconds]
      [-until date_time]
      [-forever]
```

## Arguments

- `-hours` *hours*
  Suspends a process for the specified number of hours. By default, the value of *hours* is `0`.

- `-minutes` *minutes*
  Suspends a process for the specified number of minutes. By default, the value of *minutes* is `0`.

- `-seconds` *seconds*
  Suspends a process for the specified number of seconds. By default, the value of *seconds* is `0`.

- `-until` *date_time*
  Suspends a process until a specified date and time. The date value can be a character string in the standard form *yy-mm-dd_hh*:*mm*:*ss*. It can also be a character string in any form accepted by the `(date_time)` command function. In this case, the string must be enclosed in apostrophes.

- `-forever` CYCLE
  Suspends a process until it receives an interrupt.

## The `status` Request

The `status` request displays the current state of the OSL configuration. By default, it displays the number of OSL server processes that are currently idle and summarizes the state of other modules connected over OSL.

## Display Form

```
---------------------------------- status ----------------------------------
destination:
-from:
-long:       no
```

## Command-Line Form

```
status destination
       [-from module_name]
       [-long]
```

## Arguments

- *`destination`* **Required**
  The name of a module whose status you want to display. If you do not specify a value for this argument, the request displays the status of all modules on the OSL network.

- `-from` *`module_name`*
  The name of a module that you want to originate the request. The default value is the current module. If you specify a value, the request displays the status of the destination module, as determined by the module you specify for this argument.

- `-long` CYCLE
  Enables the request to display additional status information, such as individual port numbers and their state. By default (the value `no`), the request does not display additional status information.

## Explanation

The `status` request displays messages about possible error conditions. For example, when it checks the number of idle OSL server processes, it also checks that the value for OSL server processes is `0`. (Typically, some servers should always be idle, though some conditions can cause all OSL servers to be busy simultaneously.)

If the number of idle OSL server processes is `0`, the request displays the following message.

```
This may be an indication that more osl_server processes
should be started.
```

Also, when a destination module is specified, it may display the following message.

```
N processes are waiting for an RPC tsb.
```

This message indicates that the outgoing side of OSL is backed up to the destination module, and that the load on OSL is greater than it has been configured to handle. Increasing the value of `max_open_servers` or adding more OSL server processes to the remote module may solve this problem.

Because a knowledgeable OSL administrator needs to examine the network in response to both of these messages, the request also displays the following message:

```
Some conditions can cause all OSL servers to be busy at once,
but, in general, there should always be at least a couple of
servers idle.
```

## Examples

In the first example, `status` shows that four OSL servers are idle, indicating that resources are available for new requests. It also shows the status for the OSL modules. Module `m1` is online with one RPC connection and no FQP connections. Module `m3` is currently offline. OSL will block requests to that module for another 46 seconds. (OSL has a 1-minute backoff timer that blocks traffic until the timer expires.) When the backoff timer expires, the message changes to `Retry now possible`.

```
osl_admin: status

4 osl_server processes are currently idle.

Status for %osl#m1:

  Destination is online.

Tracing has been auto-stopped.
500 entries allocated for this trace buffer
Shadow notifies are not expedited
Monitoring disabled

3 interfaces configured; RPC connect limits 4 / 10
```

*(Continued on next page)*

```
  1 RPC connection:
   1 connection on 192.168.100.1
  No FQP connections are active.

Status for %osl#m3:

  Destination is offline. Retry possible in 46 seconds.

Tracing has been auto-stopped.
500 entries allocated for this trace buffer
Shadow notifies are not expedited
Monitoring disabled

3 interfaces configured; RPC connect limits 4 / 10

  4 RPC connections:
   2 connections on 192.168.101.3
   2 connections on 192.168.100.3

  No FQP connections are active.
```

This second example shows the messages that `status` displays when a destination module is disabled.

```
osl_admin: disable_destination m3
osl_admin:
osl_admin: status m3

Status for %osl#m3:

Tracing has been auto-stopped.
500 entries allocated for this trace buffer
Shadow notifies are not expedited
Monitoring disabled

3 interfaces configured; RPC connect limits 4 / 10

  Destination has been disabled.

  RPC connections:
   3 connections on 192.168.101.3
   1 connection on 192.168.100.3

  No FQP connections are active.
```

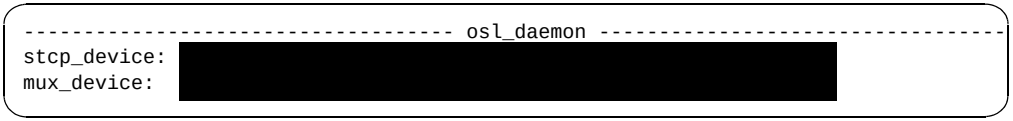
Note that one network has three connections and another network has one. The OSL administrator could issue the `reset_port` request to reset the network to two connections on each port.

# osl_daemon *Privileged*

## Purpose

The osl_daemon command starts the OSL daemon process for the OSL multiplexor and the STCP protocol driver.

## Display Form




## Command-Line Form

```
osl_daemon stcp_device
           mux_device
```

## Arguments

- *stcp_device* **Required**
  Specifies the name of the STCP protocol driver, which must have an entry in the devices.tin file. By convention, the name is stcp.*module_name*, and the value must begin with the pound sign (#).

- *mux_device* **Required**
  Specifies the name of the OSL multiplexor, which must have an entry in the devices.tin file. By convention, the name is stcp_osl_mux.*module_name*, and the value must begin with the pound sign (#).

## Explanation

You must start the OSL daemon process for the OSL multiplexor and the STCP protocol driver for the current bootload. To do so, issue the following command at the command line:

```
start_process 'osl_daemon #stcp.module_name
     #stcp_osl_mux.module_name' -privileged -priority 7
```

To start the OSL daemon process for subsequent bootloads, uncomment this command line in the `module_start_up.cm` file, substituting *`MODULE`* in the file with the module name. You can also use this command to restart a daemon that has been terminated.

If the OSL daemon process is terminated or is not started, the current module cannot establish incoming or outgoing connections; therefore, other modules cannot communicate with it.

If the OSL daemon process is terminated, connections that have already been established are not affected.

## Related Information

For additional information, see the following:

- ‘‘Adding the OSL Multiplexor to the `devices.tin` File’’ on page 1-10
- ‘‘Starting the OSL Daemon Process’’ on page 1-11
- *VOS STREAMS TCP/IP Administrator’s Guide* (R419) for information on creating the `devices.tin` file entry for the STCP protocol driver

# osl_overseer *Privileged*

## Purpose

The osl_overseer command starts the OSL overseer.

## Display Form

```
--------------------------------- osl_overseer ---------------------------------
No arguments required. Press ENTER to continue.
```

## Command-Line Form

```
osl_overseer
```

## Explanation

The OSL overseer starts and manages osl_monitor_*N* processes. You control these processes using the set_monitoring request of the osl_admin command.

To automatically start the osl_overseer process each time you reboot the module, uncomment the osl_overseer command lines in the module_start_up.cm file (in the directory (master_disk)>system). To start an osl_overseer process for the current bootload only, issue the command start_process 'osl_overseer' from command level.

## Related Information

For information on the set_monitoring request of the osl_admin command, see "The set_monitoring Request" on page 6-55.

# osl_server *Privileged*

## Purpose

The osl_server command starts an OSL server (osl_server). The OSL servers receive requests that require user-level processing from SOSL Net driver (sosl_net_driver) on the same module.

## Display Form

```
---------------------------------- osl_server ---------------------------------
-syserr:        yes
-bridge:         no
-super:          no
-server_suffix:
```

## Command-Line Form

```
osl_server [-no_syserr]
           [-bridge]
           [-super]
           [-server_suffix suffix]
```

## Arguments

- -no_syserr CYCLE

  Specifies that the osl_server process does not write error messages and other routine messages to the syserr_log.*date* file. By default, the osl_server process writes these messages to the syserr_log.*date* file (the default value is yes). If you specify the value no, the osl_server process writes the messages to the default output path of the process (typically, the *process_name*.out file).

- -bridge CYCLE

  Creates an inbound OSL server on modules that run both OSL and proprietary StrataLINK. *Inbound OSL servers* forward inbound requests from bridge modules on remote systems to nonbridge modules on the local system. When you create one inbound OSL server on a module, all of the OSL servers on the module must be inbound. By default, the OSL server does not receive inbound requests from remote systems (the default value is no). The arguments -bridge and -super

are mutually exclusive; if you specify the value `yes`, you cannot specify the `-super` argument. Do not specify the `-bridge` argument on modules that are running only OSL.

- `-super` CYCLE
  Creates an outbound/inbound OSL server (an OSL server that forwards outbound requests from the local system to remote systems and receives inbound requests from remote systems for the local system). Bridge modules typically have an outbound/inbound OSL server; specify the value `yes` for bridge modules. When you create one outbound/inbound OSL server on a module, all of the OSL servers on the module must be outbound/inbound. By default, the OSL server does not forward outbound requests to or receive inbound requests from remote systems (the default value is `no`). The arguments `-super` and `-bridge` are mutually exclusive; if you specify the value `yes`, you cannot specify the `-bridge` argument.

- `-server_suffix` *`suffix`*
  Specifies a suffix to create a unique name for the OSL server. The suffix is appended to the name `osl_server` to create a name of the format `osl_server`*`suffix`*. (For example, when you specify the value `1` for the `-server_suffix` argument, the OSL server name is `osl_server1`). The value you specify for *`suffix`* has a maximum length of 16 characters.

## Explanation

In a multimodule system, you must have `osl_server` processes on each module in the system that communicates using OSL and STCP. In a basic configuration, the number of `osl_server` processes that you start should equal the following.

- On non-bridge modules, the number of `osl_server` processes that you start should typically equal twice the value you specify for the `max_open_servers` field of the `new_modules.tin` file (or for the `-max_open_servers` argument of the `add_module` command).
- On bridge modules, the number of `osl_server` processes that you start should typically equal four times the value you specify for the `max_open_servers` field of the `new_modules.tin` file (or for the `-max_open_servers` argument of the `add_module` command).

To automatically start the `osl_server` processes each time you reboot the module, uncomment the `osl_server` command lines in the `module_start_up.cm` file (in the directory `(master_disk)>system`). To start the `osl_server` processes for the current bootload only, issue the command `start_process 'osl_server'` from command level for each server that you want to start.

A bridge module typically has outbound/inbound OSL servers. Outbound/inbound OSL servers forward outbound requests from nonbridge modules on the local system to other bridge modules on remote systems. They also receive inbound requests from

other bridge modules on remote systems for nonbridge modules on the local system. To create outbound/inbound OSL servers, specify the `-super` argument.

NOTE

All OSL servers on one module must be of the same type. If you specify the `-super` argument for one `osl_server` command, you must specify it for all `osl_server` commands issued on a module. If you specify the `-bridge` argument for one `osl_server` command, you must specify it for all `osl_server` commands issued on a module. If you do not specify the `-super` argument or the `-bridge` argument for one `osl_server` command, you must omit those arguments for all `osl_server` commands issued on a module.

## Examples

The following example shows how you might issue the `osl_server` command from the `module_start_up.cm` file. The command starts the outbound/inbound OSL server process `osl_serverBridge1` on bridge module `#m10` in the system `%admin`. (Note that the ampersand-plus (&+) characters at the end of a line indicate that the command continues on the next line.)

```
start_process (string osl_server -syserr -super &+
     -server_suffix Bridge1) -priority 9 -privileged
     -process_name osl_serverBridge1 -output_path &+
     osl_serverBridge1
```

## Related Information

To view the sample `module_start_up.cm` file that is shipped with the VOS installation software, see the `module_start_up.cm` file that resides in the `(master_disk)>system` directory.

For complete information on the `max_open_servers` field of the `new_modules.tin` file, see "Creating Entries in the `new_modules.tin` File" on page 3-7. For complete information on the `max_open_servers` field of the `new_backbone_systems.tin` file, see "Creating Entries in the `new_backbone_systems.tin` File" on page 4-15.

# Index

## C

## F



## G



## H

## I



## L



## M

## N

## O

## P



## Q



## R

## S

## T



## U



## V