Installation Guide

TCP/IP for VSE

Version 2 Release 2

TCP/IP is a communications facility that permits bi-directional communication between VSE-based software and software running on other platforms equipped with TCP/IP.

This manual explains the product installation and configuration process.

Published October 2017

CSI INTERNATIONAL

"Delivering what the competition can only promise"

www.csi-international.com • info@csi-international.com • 800.795.4914

TCP/IP FOR VSE Installation Guide
Version 2 Release 2
October 2017

Published by CSI International

Phone:	800-795-4914
Fax:	740-986-6022
Internet:	http://www.csi-international.com
Product questions:	info@csi-international.com
Technical support:	support@csi-international.com
Review comments:	documentation@csi-international.com

CSI International Technical Support

During Business Hours Monday through Friday, 9:00 A.M. through 5:00 P.M. EST/EDT.

Telephone:	Toll Free in the USA	800-795-4914
	Worldwide	740-420-5400
Email:	support@csi-international.com	
Web:	http://csi-international.com/problemreport_vse.htm	

Emergency Service 24/7 After business hours and 24 hours on Saturday and Sunday:

Telephone:	Toll Free in the USA	800-795-4914
	Worldwide	740-420-5400

CSI International provides support to address each issue according to its severity.

Updates to This Manual

The following table describes updates to this manual. Updates may be identified by a fix number in CSI International's support database.

October 2017

ID	Change Description	Page
	Ch. 3, "Installation": Updated the product installation procedure.	32
	Ch. 6, "Configuring FTP Daemons": Added TLS 1.2 support (the SET TLS12 command) to the section "Creating an External FTP Server."	104

Table of Contents

To return from a hyperlink jump, press <Alt> <◄>

1

Fundamentals of TCP/IP

Overview

This chapter presents an overview of the Transmission Control Protocol/Internet Protocol (TCP/IP) and its principles. It provides enough information to allow a new TCP/IP user to configure and use TCP/IP FOR VSE. If you are an experienced TCP/IP user, you can skip this chapter. If you are a new TCP/IP user, we recommend that you supplement this material with a TCP/IP textbook or workshop.

TCP/IP Protocol

TCP/IP is a communications protocol that was developed for the United States Advanced Research Projects Agency (ARPA). The TCP/IP protocol is a set of rules that enable different types of computers to communicate with each other. The computers communicate by using the standard TCP/IP protocol, or format, to transfer or share data.

The TCP/IP protocol rules are established and maintained by the Internet Engineering Task Force (IETF). The IETF is an international community of network designers, operators, vendors, and researchers concerned with the Internet's architecture and operation. The IETF's mission is to produce technical and engineering documents that influence the way people design, use, and manage the Internet with the view of making it work better. These documents include protocol standards, best current practices, and informational updates of various kinds. Each document is commonly referred to as an RFC, or Request for Comments.

For more information on the IETF, visit www.ietf.org.

Why on VSE?

Most computers already use TCP/IP to communicate with each other. Adding the TCP/IP protocol to VSE permits VSE-based systems to communicate and share data with virtually any other computer in the world.

Structure of TCP/IP

This section explains the basic structure of TCP/IP. Understanding this structure is important to the TCP/IP FOR VSE user.

Clients, Servers, and Daemons

TCP/IP activity begins with a request made by a *client*. In this context, a client is a program that initiates a request from a remote location. Typical clients include

- Web browsers
- File Transfer Protocol (FTP) programs
- Line Printer (LPR) programs, also known as printer emulators
- Telnet programs, also known as terminal emulators

Remember that VSE is sometimes a client. When VSE requests an action from a remote server, VSE is a client of the remote server.

A *daemon* or *server* is a program that receives and processes requests from clients. These terms are interchangeable.

Host

A *host* is a system that runs TCP/IP software. A host can be:

- A VSE partition. Each VSE partition that runs TCP/IP for VSE is a host.
- A personal computer (PC). Each PC on your network that runs TCP/IP is a host.
- Any other type of computer or peripheral drive. UNIX and AS/400 systems are hosts.

Generally, a host links a set of clients and/or servers to the TCP/IP network. *Multi-homing* occurs when a single computer contains multiple LAN adapters and possibly multiple IP addresses. TCP/IP FOR VSE supports multi-homing.

TCP and UDP

Transmission Control Protocol (TCP) software accepts requests directly from clients and servers. TCP accepts data transmission requests of any length and transports the data across the network. When required, TCP breaks the data into smaller pieces called datagrams or packets. It uses checksums, sequence numbers, timestamps, time-out counters, and retransmission algorithms to ensure reliable data transmission.

The User Datagram Protocol (UDP), an alternate transport function, runs parallel to TCP. UDP software does not guarantee reliable data transmission. UDP is the method of choice in situations where speed is critical and reliability is secondary.

For example, UDP is used for Internet radio broadcasting because the occasional garbled byte or missing packet is acceptable in view of the high transmission rates that are required. To circumvent this possible problem, both clients and servers use retransmission and verification key algorithms to ensure that packets are used in the correct order and are complete.

IP

The Internet Protocol (IP) software is the part of TCP/IP that actually performs the communication function. After IP receives packets of data from TCP or UDP, it ensures that the packet size meets the requirements of the transmission path and physical adapters, such as Ethernets and CTCs. IP changes the packet size if necessary and transmits the data.

It is important to understand that IP software is not designed to be reliable. TCP expects IP to transmit the data immediately, so IP sends the data with no further checks. If transmission is delayed for any reason, the data is discarded. TCP, however, uses its verification systems to ensure that the data is received. If TCP does not receive acknowledgement of a complete transmission, it retransmits the data. One consequence of this system is that retransmissions increase when a communication path becomes saturated, and that causes CPU consumption to increase.

IPv4 Network Addressing

TCP/IP recognizes several types of address structures. At the lowest level, each piece of communications hardware has its own address structure. For example, each Ethernet card has a unique identification number that is assigned at the factory. When TCP/IP transmits across an Ethernet network, it must coordinate the network address with the Ethernet address. To do this, TCP/IP uses an internal system called Address Resolution Protocol (ARP).

IP Address

One of TCP/IP's best features is its consistent way of identifying each TCP/IP host. When you connect to the Internet, you must follow standard Internet procedures. First, you register your network with the Internet Network Information Center (InterNIC), which assigns you a unique network ID. For registration information, contact your *Internet provider*. Your Internet provider is the organization that connects you to the Internet.

The IPv4 address is a 32-bit number. This number contains both a network ID and a host ID. The network ID identifies your network to other networks. The host ID identifies the individual hosts within your own network.

The first part of the address identifies the address class. The address class determines the format of the other bits in the address. There are three address classes, which are described in the following sections.

Class A

The class A format is as follows:

0	1.....7	8......................31
0	Network	Host

The class A address begins with a bit of 0, followed by a 7-bit network ID and a 24-bit host ID. A class A address supports a maximum of 128 networks, and each network can have 16 million hosts. To obtain a class A address, you have to explain why you need one. Class A addresses are easily recognizable because they have values of 1 through 127, inclusive, in the first field of the IP address.

Class B

The class B format is as follows:

0.	2.............15	16......................31
10	Network	Host

The class B address begins with bits of 10, followed by a 14-bit network ID and a 16-bit host ID. A class B address supports a maximum of 16,383 networks, and each network can have 65,535 hosts. Class B addresses have values of 128 through 191, inclusive, in the first field of the IP address.

Class C

The class C format is as follows:

0..	3...................23	24......31
100	Network	Host

The class C address begins with bits of 110, followed by a 21-bit network ID and an 8-bit host ID. The class C address is the most common type of address. A class C address supports a maximum of 2,097,151 separate networks, and each network can have 256 hosts. In practice, this works well because most networks have fewer than 256 hosts. Class C addresses have values of 192 through 223, inclusive, in the first field of the IP address.

Address Notation

Regardless of its class, an IPv4 address is always written as four decimal numbers separated by periods. Each number represents one byte of the address. This notation is not standard for mainframe programmers, but it eliminates ambiguity between machines that order bits and bytes differently.

Multiple Addresses

A host can have more than one address. This situation occurs when a host is part of multiple networks and is assigned an address on each network.

Subnetworks

If your network consists only of some PCs on an Ethernet or a Token Ring, converting TCP/IP network addresses to physical addresses (for the Ethernet or Token Ring hardware) is not a problem. TCP/IP's address resolution protocol (ARP) obtains physical addresses by broadcasting "where are you" requests. When you attach multiple physical networks to a single host, such as TCP/IP FOR VSE, addresses are more difficult to obtain. Because the ARP facility cannot extend beyond the physical network to which it is attached, you must provide ROUTE statements that link each network address to a physical network and location.

Instead of providing large numbers of ROUTE statements, you can assign network addresses based on a subnetwork. To do this, you provide a subnet mask value that is applied to the host portion of the TCP/IP address. The resulting value is the subnetwork number. A single ROUTE statement maps the subnetwork number to a physical network.

The subnet mask value is a binary value that is logically ANDed with an IP address to obtain the subnetwork number. There are two methods that are commonly used to determine the subnet mask value. The first method is to include only the subnet portion, such as 0.0.255.0. The other method is to include the network number, such as 255.255.255.0. TCP/IP FOR VSE accepts both methods. The examples that follow use the first format.

The subnet number must begin with the left-most bit of the host number and must consist of contiguous bits.

The following subsections show an example of a subnetwork mask applied to each class of network address.

Class A With 24 bits reserved for the host number, class A addresses provide the most capability for subnetworking. If the subnetwork mask is 0.255.0.0 (X'00FF0000'), the network address appears as follows:

0	1.....7	8............15	16..............31
0	Network	Subnetwork	Host

Note that the subnetwork number is still a part of the host number. In this example, host numbers of 1 (0 is reserved) through 65,535 (X'00FFFF') belong to subnetwork 0, hosts 65,536 (X'010000') through 131,071 (X'01FFFF') belong to subnetwork 1, and so on.

Class B Class B addresses reserve 16 bits for the host address. This means that 65,535 host numbers can be subdivided into subnetworks. If the subnetwork mask is 0.0.255.0 (X'0000FF00'), the network address appears as follows

0.	2............15	16.......23	24......31
10	Network	Subnetwork	Host

This partitioning of the host number permits up to 255 hosts on each subnetwork. Specifically, hosts 1 (0 is reserved) through 255 (X'00FF') belong to subnetwork 0, hosts 256 (X'010000') through 511 (X'01FFFF') belong to subnetwork 1, and so on.

Class C The class C address has 8 bits (256 values) set aside for host numbers. Subnetworking is still possible, but the distribution of hosts across subnetworks must be carefully planned. If the subnetwork mask is 0.0.0.192 (X'000000C0'), the network address appears as follows:

0..	3...................23	.	26....31
100	Network	SN	Host

In this example, there are four possible subnetworks. Hosts 1 (0 is reserved) through 63 (X'6F') belong to subnetwork 0, hosts 64 (X'10') through 127 (X'7F') belong to subnetwork 1, and so on.

Addressing Laws

The IPv4 addressing laws are summarized in the following list. For an explanation of each law, see the next section.

1. Addresses are represented by four decimal numbers. Each number has a value in the range of 0 through 255. Each number represents one byte of the four-byte address.
2. Each address consists of a network number and a host number.
3. The network number consists of the left-most one, two, or three bytes, depending on the value of the left-most byte. The values that correspond to the number of bytes are as follows:
   - 1 through 127 = one byte
   - 128 through 191 = two bytes
   - 192 through 223 = three bytes
4. A portion of the host number may designate a subnetwork number. You can determine the host number and subnetwork number by using the following procedure:

   A. Remove the network portion of the address.

   B. Apply the subnetwork mask to the remainder of the address. Use a logical AND operation to determine the subnetwork number.

   C. Remove the subnetwork number to determine the host number.
5. A host number cannot be zero or all ones.
6. All devices on a single physical Ethernet or Token Ring have the same network and subnetwork numbers.
7. Devices that are not on the same physical Ethernet or Token Ring cannot share a common network and subnetwork number.
8. All devices on the same network must use the same subnetwork mask.
9. No two devices may use the same address.
10. Devices that connect to multiple networks have a different address on each network.
11. All rules are likely to change.

Law Explanations

Each addressing law is explained below.

1. Addresses are represented by sets of four decimal numbers. Each number has a value in the range of 0 through 255. Each number represents one byte of the four-byte address. Thus, the address of 001.002.003.004 is equivalent to X'01020304', and 10.255.0.16 is equivalent to X'0AFF0010'.

2. Each address consists of a network number and a host number. The network number identifies the physical network. The host number identifies the individual devices on the network.

3. The network number consists of the left-most one, two, or three bytes, depending on the value of the left-most byte. Three types of network/host addresses are possible:

   - Class A addresses use a one-byte network number and a three-byte host number.

   - Class B addresses use a two-byte network number and a two-byte host number.

   - Class C addresses use a three-byte network number and a one-byte host number. The network type is determined by the value of the left-most byte in the address, as explained earlier in the section "IPv4 Network Addressing" on page 4.

4. A portion of the host number may designate a subnetwork number. In addition to the network number, users may arbitrarily use the left-most portion of the host number to identify a subnetwork number. Subnetworks are used because, in the real world, there is a shortage of network numbers. Remember that if your network connects to the Internet at any point, all of your addresses must be unique.

   To alleviate the network number shortage, you can use part of the host address field as a subnetwork number. The subnet mask that determines which portion of the host number to use is completely arbitrary and entirely up to the user, provided that all users on the network use the same mask value.

5. A host number cannot be zero or all ones. These values are reserved for special purposes, such as broadcast messages.

6. All devices on a single physical Ethernet or Token Ring have the same network and subnetwork numbers. TCP/IP routing relies on the correspondence of physical networks to network and subnetwork numbers.

7. Devices that are not on the same physical Ethernet or Token Ring cannot share a common network and subnetwork number.

8. All devices on the same network must use the same subnetwork mask.

9. No two devices may use the same address. This happens more often than people care to admit. When this happens, the results are bad.

10. Devices that connect to multiple networks have a different address on each network. This is called *multi-homing*. All gateways use multi-homing. If TCP/IP FOR VSE is connected to multiple networks, it is assigned an address on each network.

11. Change is inevitable. As the Internet expands, TCP/IP grows and changes. As IP addresses become scarce, new IP versions are released to provide more capacity. IP Version 6, or IP TNG (The Next Generation), replaces the current IPv4 version. IPv6 uses a completely different addressing notation. Fortunately, IPv4 and IPv6 will coexist for some time. Look for IPv6 implementations in routers and bridges, and in hosts such as VM and VSE.

Routing and Gateways

Routing and Gateways

If you connect a PC to your mainframe using an Ethernet or a Token Ring, the PC has a real address that is recognized by the physical hardware. This address is sometimes referred to as a Machine Address Control address (MAC address). This is the only address that you can use when you send data to the PC using the Ethernet or the Token Ring. In fact, regardless of how you connect your PC to the mainframe, the only way to communicate with it is by using the real hardware address.

Fake Software Addresses

TCP/IP assigns each host an internal address, which is really a “fake” software address. Because the hardware does not recognize TCP/IP internal addresses, there is no way to send messages using these addresses. To solve this problem, TCP/IP uses its address resolution protocol to convert the internal address to the MAC address. The ARP protocol is described in more detail below.

Convenience

Although internal addresses are useless for delivering messages across a physical network, they are convenient. Internal addresses provide consistency across an Internet that contains many different types of hardware. In fact, this one reason justifies TCP/IP’s arbitrary method of addressing.

Address Translation

When you use TCP/IP, you use the internal address to send your data. The IP portion of TCP/IP translates the internal address into a MAC address, regardless of the hardware involved.

ARP

ARP enables TCP/IP to determine the hardware address of a TCP/IP host. To see how this works, consider an example. Assume that you are the flight attendant on an airline, and you want to give a lucky passenger a free first-class ticket. Because you do not know where the passenger is sitting, you use the loudspeaker to ask Mr. Lucky to ring the flight attendant call button. Everyone on the plane hears the announcement, but only Mr. Lucky responds. ARP works in a similar manner. If TCP/IP FOR VSE wants to find a specific PC, it broadcasts an ARP message over the network. Everyone receives the ARP message, but only the specific PC responds.

When a message is destined for a PC on an Ethernet or Token Ring and TCP/IP does not know its MAC address, it broadcasts an ARP message on the Ethernet or Token Ring. Every adapter on the network accepts the ARP message and passes the request to the TCP/IP software running on the machine. The PC with the matching address responds by sending its hardware address. TCP/IP saves this address in a table and keeps the messages flowing.

Routing

ARP requires that the address it is searching for be on the same physical network. If this is the case, the machine you are searching for replies to ARP’s request and you can communicate with it. If this is not the case, then you must tell TCP/IP how to find the path to the target computer.

TCP/IP FOR VSE uses one of two methods to map an internal address to a MAC network. The first method is to provide a ROUTE command for each TCP/IP address, linking it to a specific physical network. A second and more convenient method is to establish logical subnetworks, where ranges of TCP/IP addresses are mapped to physical networks.

Gateways

A *gateway* is a device (hardware, software, or combination) that is connected to two or more physical networks. When a gateway receives a message, it forwards the message to its destination using the appropriate physical network.

ARP protocols, described above, are ineffective with gateways, unless the gateway is sophisticated enough to respond to ARP requests for all devices on the attached physical networks. In practice, TCP/IP addresses are grouped into subnetworks based on physical networks. Explicit routing information enables TCP/IP to send messages to the appropriate gateway. TCP/IP FOR VSE, when used as a gateway, is sophisticated enough to respond to ARP requests for multiple IP addresses. This is important if you have multiple VSE machines, or VSE and VM machines connected with a channel-to-channel adapter (CTCA). See the DEFINE ALTIP command description in the *TCP/IP FOR VSE Command Reference* for more information about this function.

File Systems

Because the IBM platforms are latecomers to the TCP/IP arena, most client software available now understands file structure from a UNIX and a PC perspective. The TCP/IP FOR VSE file system is designed to allow existing clients to access VSE-based datasets.

PC File Structure

A PC's file structure has the following characteristics.

Names

In the PC structure, a dataset name consists of a directory, subdirectories, name, and extension. Only the name is required. A typical name follows.

```
\root\level_1\level_2\name.ext
```

The directories are delimited by backward slashes (\ ). The name and its extension are separated by a period. The first backslash indicates that *root* is in the base directory of the PC file system. Omitting the first backslash indicates that *root* is a subdirectory of the current working directory. Older PC file systems limited names and directory names to eight alphanumeric (and a few other printable) characters. The file extension was, at most, three characters. Windows has eliminated these restrictions and even permits embedded blanks in the names. This type of file system is often referred to as a *hierarchical file system*.

File Content

A PC file consists of a single string of bytes. There is no provision for records except for the convention that text files can be broken into lines by including the carriage-return or line-feed pairs.

UNIX File Structure

The UNIX file structure has the following characteristics.

Names

UNIX systems follow a pattern similar to PCs. A UNIX file name might appear as follows:

```
/root/level_1/level_2/name.ext
```

Note that the separator character is a forward slash. Also, the file extension is simply a part of the file name and has no significance to UNIX. Like the Windows file name format, individual directory and file names can be lengthy. One final consideration is that UNIX file names are case-sensitive.

Although a UNIX file name is similar to a PC file name, the mechanism that UNIX uses to describe its files using the FTP protocol is different. Many PC-based software clients expect output to be returned in UNIX format, even though such output might not make sense in a VSE context. To support these clients, TCP/IP FOR VSE uses a concept called *UNIX Mode* in which the external representation of the TCP/IP FOR VSE file system is returned in UNIX format.

File Content

Like a PC file, a UNIX file is a single string of bytes. There is no provision for records except for the convention that text files may be broken into lines by the inclusion of carriage-return or line-feed pairs.

VSE File Structure

VSE’s file structure is more complex than the structure on either a PC or a UNIX system. VSE uses multiple file systems such as VSAM, VSE/POWER, and Librarian that can be used in different ways, depending on what is appropriate for an application.

TCP/IP FOR VSE File Structure

TCP/IP FOR VSE has structured its file system to permit a reasonable mapping of VSE and PC/UNIX facilities. This mapping takes into consideration the large variety of VSE file types, objects that should be addressed as files (such as VSE/POWER queue entries), and processes defined through APIs. This enables VSE to provide a comfortable interface to the entities (including humans and client software products) that expect a file system to be organized as a hierarchical file. So how do we take a non-hierarchical file system and represent it to the outside world as a hierarchical one? The next few sections describe this process.

Public Names

Making information in the VSE world available to the TCP/IP community is difficult because naming conventions are inconsistent. To solve this problem, we use a *name space*. A name space contains names that correspond to VSE entities. The names are known as *public names* because they are names used by clients external to VSE.

Structure

A public name has the following structure:

```
directory.directory.directory.name
```

The *name* field is required. The *directory* fields are optional; if present, there can be as many as 21 levels. Note that the directory structure is completely arbitrary and is intended to make VSE dataset access easy and logical to the end user. A typical way to configure a file system is to have root level entries for each type of file. For example, at the root of a TCP/IP FOR VSE file system, you could have the following entries:

- POWER, to represent the VSE/POWER queue file (IJQFILE)
- MASTCAT, to represent entries in the VSAM master catalog
- PRD1, to represent the PRD1 library
- ICCF, to represent the ICCF library (DTSFILE) label

You can define each node in the hierarchy in a similar manner. TCP/IP FOR VSE defines certain types of files, such as POWER, for you.

It is important to note that all components of TCP/IP FOR VSE share the same file system. So if you define a file for one purpose, such as FTP, you have also defined it for other purposes, such as GPS and the VSE web server.

Syntax

To permit maximum flexibility in meeting the needs of three different platform types, TCP/IP FOR VSE uses periods, forward slashes, and backward slashes somewhat interchangeably. As a result, the following public names all refer to the same pseudo file (for example, all entries in the POWER LST queue class A):

- POWER.LST.A
- POWER/LST/A
- POWER\LST\A

The following rules apply:

1. Periods are acceptable except when accessing HFS files. (The period is an allowed character in an HFS resident file.)
2. Forward slashes are restricted to UNIX simulation mode.
3. Backward slashes are restricted to PC mode.

For FTP transfers, the mode is determined by the change directory (CD) command. To set UNIX mode, use the CD / format. To set PC mode, use the CD \ format. To set the FTP daemon's default mode of operation, use the DEFINE FTPD command or a client SITE command.

Qualifying Public Names

In general, a public name, as it appears in the file structure, points to a VSE file defined by a DLBL statement, a POWER queue, an ICCF library, a Librarian library, or a user-defined process. When a TCP/IP client uses a public name, it specifies one of the following:

- The public name that is assigned to a file
- The public name that is assigned to a file, with additional qualifiers. The additional qualifiers are used when the public name refers to a POWER queue or a library.

For example, if VSE.PRD2.LIBRARY is the public name of a Librarian file, a client might request file VSE/PRD2/LIBRARY/TEST/PROG.A. The additional levels of qualification specify the sublibrary TEST, the member PROG, and the type A.

2

Planning for Installation

Overview

Before installing TCP/IP FOR VSE, you need to make several decisions about your needs and capabilities. This chapter presents a series of questions and provides a worksheet that you can use to determine your installation-dependent values.

If you are viewing this document online, we recommend that you produce a printed copy of this chapter and use it as a worksheet.

Compatibility

TCP/IP FOR VSE Version 2.1 can be run on any release of the VSE operating system, including VSE/ESA 2.7, z/VSE 3.1, z/VSE 4.3, and z/VSE 5.1.

Storage Requirements

To install TCP/IP FOR VSE, you need a VSE library with a minimum of 9,000 available blocks. You need two sublibraries: one for the distributed TCP/IP FOR VSE system materials, and one for your site's custom data. Using a separate sublibrary for your site's custom data allows you to keep your data intact during reinstallation and maintenance. We recommend that the library and sublibrary names be PRD2.TCPIP and PRD2.CONFIG, respectively. Using this naming standard reduces confusion when you talk with Technical Support and when you perform standard reinstallation and maintenance tasks. Fill in the chart below with your site's names.

Description	Recommended Name	Your Name
Library Name	PRD2.TCPIP	
Sublibrary Name	PRD2.CONFIG	

Communication Adapters

Before TCP/IP FOR VSE can communicate with other TCP/IP hosts, you must provide one or more communications adapters. Adapter types include LAN Channel Station adapters, Common Link Access to Workstation routers, and channel-to-channel adapters.

LCS Devices

Many communications adapters use a protocol called the LAN Channel Station (LCS). The LCS protocol became popular with the IBM 8232 and continues to be supported in IBM control units such as the 3172, 2216, and the Open Systems Adapter. Many OEM vendors offer control units that emulate the LCS, including the Netshuttle for VSE, a Bustech device. Other OEM products that emulate a 3172 include the Polaris StarGate and the Interlink 3762. In this chapter, we use the generic term *LCS* to mean any of these devices.

Each LCS occupies an even-odd pair of hardware unit addresses. Depending on the model and features, your LCS can be addressed by more than one pair of unit addresses. This permits the device to be shared across partitions by different programs. For example, the same LCS might be used by VTAM at addresses 050/051 and by TCP/IP at addresses 052/053.

Up to 16 adapters can reside within an LCS device. Each adapter serves as a physical connection to a network and is addressed by its position within the device. The first adapter, the default, is identified as “adapter 0.” When coding TCP/IP FOR VSE initialization statements, you must explicitly define each adapter available to TCP/IP FOR VSE.

You must configure the adapters to reject non-IP traffic and traffic not explicitly addressed to the adapter. Consult your LCS documentation for details. For example, with Netshuttle for VSE, there is an IP filter flag that is accessible through the configuration utility supplied with the device. The default for most devices is to filter out non-IP traffic, but you should verify this for your specific device. If you do not filter out non-IP traffic, all traffic is passed to TCP/IP FOR VSE even if it is not intended for delivery. TCP/IP FOR VSE must then evaluate each message and either discard it or continue to process it. This additional processing can increase CPU consumption and greatly degrade TCP/IP response time.

MTU Size

You must select a maximum transmission unit (MTU) size during installation. The MTU size determines how much data TCP/IP FOR VSE can send in one operation. The best MTU size for your installation depends on your control unit and other network characteristics. See “MTU Size” in Chapter 12, “Performance,” for more information on MTU size and data fragmentation.

MTU sizes vary depending on the adapter type, the manufacturer, the network, and the devices attached. In all cases, TCP/IP requires a minimum size of 576. The maximum size is the value allowed by the most restrictive device on the path between your adapter and the remote device(s) with which the adapter must communicate. The following table summarizes values that are acceptable in most TCP/IP FOR VSE installations.

Adapter	Default	Minimum	Maximum
Ethernet	1500	576	1500
Token Ring	1500	576	Ring dependent: 4 Mbit/sec ~4000 16 Mbit/sec ~8000
FDDI	1500	576	2000

We recommend using the default or larger MTU size for each adapter. This value can then be restricted selectively for each path by overrides supplied by DEFINE ROUTE commands. TCP/IP FOR VSE includes the DISCOVER client, which allows you to determine the optimal MTU size for any path. This client is documented in the *TCP/IP FOR VSE User Guide*, Chapter 5, "Ping, Traceroute, DISCOVER Clients."

Cisco Router

The Cisco router is a high speed channel-attached device that supports Ethernet, Token Ring, X.25, and other communications adapters. The Cisco router is connected to TCP/IP FOR VSE using the Common Link Access to Workstation (CLAW) interface. MTU size specifications are identical to those of the Channel-Attached RS/6000 and are explained in the following section.

Channel-Attached RS/6000

This interface is commonly referred to as the Common Link Access to Workstation (CLAW). Each device occupies an even-odd pair of unit addresses.

The MTU size must conform to the following rules:

- The minimum size is 576.
- The maximum size supported by TCP/IP for VSE is 16K. The selected size must be acceptable to TCP/IP running on the RS/6000.
- The default size is 4096.

In addition to this information, you must assign a host name, a host application name, a workstation name, and a workstation application name. The maximum length for each name is eight bytes. As far as TCP/IP FOR VSE is concerned, the first three names are arbitrary. The fourth name, which is the workstation application name, must be TCPIP. The attached device requires you to supply the names as part of its configuration data.

You also need to determine the input and output buffers sizes because these values must be specified in the attached device's configuration. Too small a value results in additional I/O operations. Too large a value wastes fixed storage. We recommend a size of 4096 bytes (4K) for the DEFINE LINK. This should then be reduced by supplying DEFINE ROUTE statements for paths that require a smaller MTU size.

Before proceeding with the installation, fill in the following chart:

Unit Address	
MTU Size	
Host name (maximum of eight bytes)	
Host application name (maximum of eight bytes)	
Workstation name (maximum of eight bytes)	
Workstation application name (maximum of eight bytes)	TCPIP
Buffer size	

CTC Adapter, 3088 MCCU, or Virtual Channel-to-Channel Adapter

TCP/IP FOR VSE can communicate with other TCP/IP hosts executing under VSE, MVS, or VM by use of a real or virtual channel-to-channel adapter (CTCA). Each connection requires two CTCAs at adjacent unit addresses. The first address must be an even value.

MTU size must conform to the following rules:

- The minimum size is 576.
- The maximum size is 32,768. The value that you select must be acceptable to the TCP/IP implementation that owns the other side of the CTC. Selection of the MTU size for a virtual CTC connection can be complicated.
- The default size is 4096.

You should select an MTU size of 32,768 on the VSE side in the following situations:

- You are connecting two VSE systems.
- You are connecting a VSE system with a VM system and traffic flows only between those two systems.

You should determine an MTU size that is consistent with the physical hardware connection in the following situations:

- You are routing VSE traffic through VM, and VM is using a hardware communications adapter.

- You are routing VM traffic through VSE, using VSE as a gateway.

Regardless of the MTU size selected for the DEFINE LINK, you must ensure that each DEFINE ROUTE statement includes an MTU override when the path being defined requires a smaller value.

Before proceeding with the installation, write your unit address and MTU size in the following chart:

Unit Address	
MTU Size	

IOCP Gen

If your computer requires you to modify IOCP before you install hardware, you may need to perform an IOCP gen. Put a check mark in the appropriate box below after you complete the IOCP gen. If you do not need an IOCP gen, put a check mark in the other box.

I have performed the required IOCP gen.	
An IOCP gen is not required.	

Software Values

During installation, you must supply the following custom information.

Product Keys

TCP/IP FOR VSE requires a product key (also known as a product code) that permits it to run at your site and only at your site. The product key is a string of five words (CSI supplied) or digit strings (IBM supplied). Depending on the optional TCP/IP FOR VSE components that you license, you may have multiple product keys. You must include each key as a separate PRODKEY macro call in the job used to generate the PRODKEYS phase. TCP/IP FOR VSE requires separate product keys for the base component, GPS, SecureFTP, SSL/TLS, and See-TCP for VSE. TCP/IP FOR VSE examines the first copy of PRODKEYS.PHASE that it can find and determines the best key (the key with the most time until expiration) for each product or feature.

When TCP/IP FOR VSE cannot find a valid key for the stack itself, it starts in demonstration mode. This mode provides full functionality, but it limits the number of daemons that can be used. A stack running in demonstration mode shuts down automatically after 60 minutes. You can restart the stack in demonstration mode for an additional hour, but the total run time for all stacks operating without valid product keys is limited to 4 hours per IPL. The demonstration mode is provided as an aid for new installations and disaster recovery only; a valid key is still required.

TCP/IP FOR VSE always uses the real serial number of the CPU. This is true even if you run VSE in a virtual machine under VM/ESA. To obtain the real serial number of the CPU, use the System Information Request (SIR) AR command under VSE/ESA. TCP/IP FOR VSE ignores the first two digits of the serial number, so if you are running TCP/IP FOR VSE in an LPAR, you can use the same product key for all VSE logical partitions running on the same physical machine.

Before proceeding with the installation, fill in the following chart:

Product key	
Serial number	

Does your installation have a disaster recovery plan? If so, you should call CSI International to obtain a product key for your disaster recovery machine. You need this product key to use TCP/IP FOR VSE at the disaster recovery site, both for testing and for genuine disasters. You can include your disaster-recovery site keys in the same phase as your production keys. The correct set is determined automatically based on the CPU ID.

Network Address and Subnet Mask

Before TCP/IP FOR VSE can communicate with another TCP/IP host, it must have a unique network address. The address is represented in the TCP/IP standard dotted-decimal notation. In this notation, the four bytes of the address are converted individually to decimal and are printed with a period separating the bytes. If you already have a TCP/IP network in place, see your network administrator for this address. If you are unfamiliar with TCP/IP addressing conventions, see the section "Network Addressing" in Chapter 1, "Fundamentals of TCP/IP."

Another network value that you need to obtain from your network administrator is the subnet mask. If you do not have a TCP/IP network in place, see Appendix B, "Quickstart Guide," for suggestions on implementing a small TCP/IP network.

Before proceeding with the installation, fill in the following chart:

IP address	
Subnet mask	

VSE/POWER LST Queue

You can use TCP/IP FOR VSE to automatically trigger actions based on POWER listings. For example, you can set the following tasks to run automatically:

- Route print output from your POWER LST queue to any printer in the network.
- FTP listings from VSE to other computers in your network.
- Email the POWER listings to a remote destination. Transmission of LST files can also involve automatic conversion to PDF format.

To automatically spool listings to remote locations, you need to select one or more output classes for monitoring. For more information, see the DEFINE EVENT command in the *TCP/IP FOR VSE Command Reference*.

Before proceeding with the installation, fill in the following chart:

FTP output class	
LPR output class	
Email output class	

VTAM Values

To use telnet for communicating with a VSE-based application such as CICS, you need to know the application ID. In addition, each telnet session (virtual terminal) requires an LU name.

Target Application IDs

The following chart shows some target application IDs:

VTAM Application	Target Application ID
CICSPROD	DBDCCICS
CICSTEST	PRODCICS

Telnet LU Names

Each telnet session requires a VTAM LU name (virtual terminal). We recommend that you assign a range of IDs, such as TCP0001 through TCP0999, since this enables you to configure all of your telnet daemons with a single statement. You can configure up to 99 telnet daemons with one DEFINE TELNETD statement. If you configure more telnet daemons than you need, you waste CPU and storage resources.

Enter your telnet IDs below:

Telnet IDs	Telnet IDs

TCP/IP for VSE File System

As discussed in Chapter 1, "Fundamentals of TCP/IP," the TCP/IP FOR VSE hierarchical file system is a combination of several different file systems. The file system is used for all file access within TCP/IP FOR VSE. This includes access by FTP, LPR, LPD, the TCP/IP FOR VSE web server (HTTP), and user- and vendor-written software.

Defining Your File System

TCP/IP FOR VSE provides several mechanisms for defining the file system. The following methods are the most common:

- Method 1: Select the appropriate files individually and give them short, arbitrary public names. This is the recommended method.
- Method 2: Use the FILESYS batch program to build a set of DEFINE FILE commands based on your standard labels. The resulting command set must be edited to ensure complete information and to remove files that should not be made available.

As an example, consider the following sample commands used to define a VSE file system:

```
DEFINE FILE,PUBLIC='VSE.LIBRARY.SYSRES',DLBL=IJSYSRS,TYPE=LIBRARY
DEFINE FILE,PUBLIC='VSE.LIBRARY.PRD2',DLBL=PRD2,TYPE=LIBRARY
DEFINE FILE,PUBLIC='VSE.POWER',DLBL=IJQFILE,TYPE=POWER
DEFINE FILE,PUBLIC='VSE.VSAM.PAYROLL',DLBL=PAYFILE,TYPE=ESDS
```

In this example, the file structure has the following characteristics:

- An FTP client running on a PC sees that VSE is the highest level directory and that it contains three subdirectories named LIBRARY, POWER, and VSAM.
- The subdirectory LIBRARY contains SYSRES and PRD2. For TYPE=LIBRARY all the sublibraries defined in SYSRES and PRD2 are subdirectories, and the members of SYSRES and PRD2 are the file names.
- The POWER subdirectory automatically contains the RDR, PUN, and LST queues. The file names are the spool files identified by the job name and number.
- VSE.VSAM.PAYROLL is a VSAM ESDS and contains no subdirectories.

Using Public Names

In our example, we have defined the following public names:

- VSE.LIBRARY.SYSRES
- VSE.LIBRARY.PRD2

- VSE.POWER
- VSE.VSAM.PAYROLL

The public names you specify can consist of as many as 21 levels of directories and the file name. The directory levels are one- to eight-character names and are separated by periods. The file name is the last field of the public name.

Each public name corresponds to a DLBL statement but has no relationship to the real dataset name. If the statement identifies a VSAM file or sequential dataset, then the user must specify the fully qualified public name. If the public name identifies a library, then the user adds an additional level of qualification to reach the member.

For more information on public names, see "File Systems" in Chapter 1, "Fundamentals of TCP/IP." For more information on the DEFINE FILE command, see the *TCP/IP FOR VSE Command Reference*.

Setting Up Your Directory

When you set up your file system, you could start with the following sample file definitions:

```
DEFINE FILE,PUBLIC='POWER',DLBL=IJQFILE,TYPE=POWER,
DEFINE FILE,PUBLIC='PRD1',DLBL=PRD1,TYPE=LIBRARY,
DEFINE FILE,PUBLIC='VSAMUCAT',TYPE=VSAMCAT,DLBL=VSESPUC
```

In this case, the directory structure is as follows:

```
POWER     <POWER queues>
PRD1      <Library>
VSAMUCAT  <VSAM catalog>
```

Notice how the virtual file system is a seamless combination of POWER spool files, VSE libraries, and VSAM catalog entries.

When you implement TCP/IP FOR VSE as a simulated UNIX or PC environment, you should take the time to design the pseudo directory and its name structure so that it has a familiar look and feel to the users.

Using DEFINE FILE Command

When you use the DEFINE FILE command to define your file system, you must specify the keyword parameter TYPE=. The TYPE= parameter tells TCP/IP FOR VSE how to define the file and what directory structure to use. The table below lists the valid values.

TYPE=	Description
POWER	Defines the VSE/POWER queues as TCP/IP files. A subdirectory structure is created automatically under the public name you specify in the DEFINE FILE statement. For example: `DEFINE FILE,PUBLIC='POWER',TYPE=POWER` The statement creates pseudo file POWER at the root of the TCP/IP FOR VSE file system. The next level of the file system contains entries RDR, LST, and PUN, which correspond to the RDR, LST, and PUN queues. Under each queue, TCP/IP FOR VSE creates the following 38 entries: • Entries for each possible POWER class, including A to Z and 0 to 9. • The word ALL, representing all entries in the specified queue. • The word BIN, representing all queues in binary mode. TCP/IP FOR VSE automatically performs binary transfers to and from any POWER queue entry accessed through the BIN class. You can issue GET, PUT, DELETE, and RENAME commands against entries on the POWER queues. Wild cards are supported for all operations.

TYPE=	Description

LIBRARY	Defines a VSE library, all sublibraries, and all members of each sublibrary as TCP/IP files. A subdirectory structure is automatically created under the public name you specify in the DEFINE FILE statement. For example,
	`DEFINE FILE,PUBLIC='PRD2',TYPE=LIBRARY,DLBL=PRD2`
	PRD2 is defined at the root of the TCP/IP FOR VSE file system. All PRD2 sublibraries are defined at the next level of the hierarchy, making all members of each sublibrary accessible as individual files.
	For example, assume you have a VSE library named JOHNLIB with sublibraries PROD and TEST. Each sublibrary has two members, which are REXXPROG.A and REXXPROG.JCL. You can issue the following command:
	`DEFINE FILE,PUBLIC='JOHNLIB',TYPE=LIBRARY,-`
	`DLBL=JOHNLIB`
	Then you can create JOHNLIB at the root level of the hierarchy and PROD and TEST at the next level. Note that if you issue a change directory command to JOHNLIB.PROD, you cannot issue another change directory command to REXXPROG.
	You can issue GET, PUT, DELETE, APPEND, and RENAME commands against library members. Wild cards are supported for all commands except DELETE and RENAME.

TYPE=	Description
VSAMCAT	Defines a VSE VSAM catalog and all files contained within as TCP/IP files. A subdirectory structure is created under the public name you specify. For example: `DEFINE FILE,PUBLIC='IJSYSUC',DLBL=IJSYSUC, - TYPE=VSAMCAT` IJSYSUC is defined at the root of the TCP/IP FOR VSE file system. All files accessible through this catalog are defined at the next level of the hierarchy. Note the following: • When you issue a GET command for a file in a VSAM catalog defined with TYPE=VSAMCAT, TCP/IP for VSE checks to see if a DLBL exists for the file in the TCP/IP partition. If it does, TCP/IP for VSE uses it. If not, TCP/IP for VSE dynamically creates a DLBL. • You cannot access VSAM-managed SAM files with a catalog defined with TYPE=VSAMCAT. You must use TYPE=SAM for these files. • You cannot issue a PUT command for a file accessed with a TYPE=VSAMCAT catalog if the file does not exist. FTP does not create the file for you. • If you issue a DELETE command for a file accessed with a TYPE=VSAMCAT catalog, TCP/IP for VSE directs IDCAMS to perform the DELETE CLUSTER. • The RENAME command is not supported for files accessed with a TYPE=VSAMCAT catalog. • The APPEND command is supported for ESDS files only. • Wild cards are supported only for directory lists.
KSDS	Defines an individual KSDS file, but it does not create a subdirectory structure. You can issue GET and PUT for a TYPE=KSDS file, but you cannot RENAME or DELETE.
ESDS	Defines an individual ESDS file, but it does not create a subdirectory structure. You can issue GET, PUT, or APPEND for a TYPE=ESDS file, but you cannot RENAME or DELETE. If you try to APPEND to a TYPE=ESDS file, the DLBL for the file must specify DISP=(OLD,KEEP) and the definition on DEFINE CLUSTER must include the NOREUSE parameter.

TYPE=	Description

ICCF	Defines the ICCF file system and all files within, but does not create a subdirectory structure. Here is an example:
	`DEFINE FILE,PUBLIC='ICCF',TYPE=ICCF`
	`GET ICCF.`*`librarynumber.membername`*
	The first statement defines ICCF at the root of the TCP/IP FOR VSE file system. The second statement allows you to access individual ICCF files. The variable *librarynumber* is the ICCF library number and the variable *membername* is the name of the member to be accessed. Because there is no directory structure, you cannot issue a CD command to the ICCF file system. You cannot WRITE, DELETE, or RENAME.

BIM-EDIT	For more information, see the appropriate BIM-EDIT documentation.
SAM	Defines a SAM file to TCP/IP FOR VSE and makes it accessible, but does not create a subdirectory structure. You can issue READ and WRITE commands for the SAM file, but you cannot RENAME, APPEND, or DELETE.
VTOC	Defines an entire VTOC to the TCP/IP FOR VSE file system. A directory structure is created consisting of the file names in the VTOC. You can RENAME and DELETE entries in the directory but you cannot READ, WRITE, or APPEND.

Using the FILESYS Program

The FILESYS batch program helps you build a file system based on the partition or standard labels that exist in your system. The program generates individual DEFINE FILE commands and stores them in a library member. After you generate this member, you must edit it to remove undesired files and duplicates, add and correct TYPE= values, and modify any other file options as needed.

You then apply these definitions by modifying your initialization deck to contain “INCLUDE *member*,” where *member* is the name of this file-definition member.

To execute the FILESYS batch program, use a job similar to the following example:

```
// JOB FILESYS EXECUTION
// LBDEF *,SEARCH=lib.sublib   *TCP/IP base library
// EXEC FILESYS
FILESYS SET command 1
FILESYS SET command 2
...
/*
/&
```

If your site requires upper-case output, then replace the EXEC FILESYS statement with the following line:

```
// EXEC FILESYS,PARM='UPPERCASE'
```

The FILESYS SET commands control the output.

SET Command	**Description**
SET OUTPUT= [SHORT | FULL]	By default, files that are not useful for client or server access, such as the POWER job accounting file or the CICS dump file, are excluded from DEFINE FILE generation. If you want to include every possible file, then use the FULL option.
SET SYSLOG= [ON | OFF]	Write the commands to SYSLOG. By default, they are not displayed on SYSLOG.
SET SYSLST= [ON | OFF]	Write the commands to SYSLST. By default, they are not displayed on SYSLST.
SET SYSPCH= [ON | OFF]	Write the commands to SYSPCH. By default, they are not displayed on SYSPCH. Note: The SYSLOG, SYSLST, and SYSPCH commands can be used together or individually when writing to the library. For example, you can echo the data to SYSLST while writing it to your library.
SET LIB=*lib.sublib*	To place a member in a VSE library, you must indicate which library to use. There is no default.
SET MEMBER= *name.type*	To create a member, you must specify the member name and type to use. Because the TCP/IP INCLUDE command accepts only ".L" members, we recommend that you use that type. There is no default.

The following examples show how to use different FILESYS options.

Example 1: Write to a VSE library member and echo it to SYSLST.

```
// JOB FILESYS EXECUTION
// LBDEF *,SEARCH=PRD2.TCPIP
// EXEC FILESYS
SET LIB=TCPIP.CONFIG
SET MEMBER=FILESYS.L
SET SYSLST=ON
/*
/&
```

Example 2: Write to the punch queue.

```
// JOB FILESYS EXECUTION
// LBDEF *,SEARCH=PRD2.TCPIP
// EXEC FILESYS
SET SYSPCH=ON
/*
/&
```

Example 3: Write to a VSE library. All messages sent to SYSLOG are in upper case.

```
// JOB FILESYS EXECUTION
// LBDEF *,SEARCH=PRD2.TCPIP
// EXEC FILESYS,PARM='UPPERCASE'
SET LIB=TCPIP.CONFIG
SET MEMBER=FILESYS.L
/*
/&
```

Securing Your File System

Many customers have expressed concern about the security of making a VSE file system available over a TCP/IP network or even over the Internet. Here are two ways to keep your files secure:

1. Control file access:
   - Do not define the file to the TCP/IP FOR VSE file system.
   - Use the LOCAL_DLBL OFF command setting, or configure FTP daemons using the DYNFILE=NO parameter setting. These settings prevent remote users from accessing local VSE files directly. See the LOCAL_DLBL and the DEFINE FTPD commands for details.
2. Use the TCP/IP FOR VSE automatic security exit, or create a custom security exit.

3

Installation

Overview

This chapter explains the TCP/IP FOR VSE installation procedure. Please review the entire chapter before beginning. You need the values determined in chapter 2, "Planning for Installation," page 16, to code some of the configuration statements.

If you are also installing any of TCP/IP FOR VSE's optional features—GPS, TLS/SSL for VSE, SecureFTP, or See-TCP for VSE—you will need to see additional information in the *TCP/IP FOR VSE Optional Features Guide*. The installation procedure indicates when to refer to this information.

We recommend that you install a minimal configuration of TCP/IP FOR VSE. Once you have completed this task, it is a simple matter to expand your configuration to the desired size and complexity. For configuration examples, see "Appendix B: Quickstart Guide" on page 230.

Step 1: Download and Extract Files

To obtain TCP/IP FOR VSE from CSI International, download the product archive (a ZIP file) from CSI International's webpage (www.csi-international.com) or FTP server (ftp://ftp.csi-international.com).You must use the ID and password that are provided for you when you set up a CSI account. To arrange for access to the FTP server, you must contact a CSI Technical Support representative (support@csi-international.com).

Extracting Files

The installation documents and files are distributed in the downloaded TCPIP*vrm*.ZIP file, where *vrm* is the version, release, and modification level. The *vrm* is currently 222 but may change as new maintenance releases are created. Replace *vrm* in all the examples below with the *vrm* of the .zip file.

When extracted, the following files should be present in a folder on your PC:

- $Readme.txt
- $Install-AWS.txt
- TCPIP*vrm*.AWS
- T*vrm*SHA1.JCL
- $Release_Notes.pdf

Readme File

The $Readme.txt file describes the files in the product archive.

Note:

Review the $Readme file in the distribution file first to check for any updates to the installation files.

A sample $Readme.txt file follows. This sample is typical and may not apply to the latest release. The actual $ReadMe in the distribution file may be different.

Sample $Readme.txt File

```
*
TCPIPvrm.ZIP contains the compressed distribution files for installing
the TCP/IP for VSE product from CSI International.
*
$Readme.txt describes the files contained in the TCPIPvrm.ZIP archive.
*
$Install-AWS.txt - instructions for installing the TCPIPvrm.AWS
virtual tape file.
TCPIPvrm.AWS - a virtual tape containing a backup of the TCP/IP
lib.sublib that can be installed with the LIBR utility.
*
After successful installation, you should cycle TCP/IP and see the
following message:
*
IPN209I CSI Service Pack vv.rr.mm.(yyyy-mm-dd) has been applied.
*
The QUERY VERSION command can also be used to see the IPN209 message.
*
TvrmSHA1.JCL - is an assembly/link-edit job that contains the values
for the SHA-1 phase verification utility(CIALSHPH).
The CIALSHPH utility is documented in the SSL/TLS chapter of the
TCP/IP for VSE Optional Features guide.
*
$Release_Notes.pdf contains important information and
considerations about this new release.
*
```

Important:

The *vv.rr.mm* and *yyyy-mm-dd* strings in the IPN209 message represent the service pack *vrm* and the date the service pack was created. When you check the IPN209 message on your system after installing the software, make sure that the *vv.rr.mm* and *yyyy-mm-dd* displayed in that message match the numbers in the $Readme.txt file for this installation. If the numbers do not match, contact CSI Technical Support.

Software File

The TCPIP*vrm*.ZIP archive file contains the following virtual tape file:

TCPIP*vrm*.AWS

You must install TCP/IP FOR VSE from this .AWS virtual tape. Go to "Step 2: Install From the .AWS Virtual Tape" on page 35.

Step 2: Install From the .AWS Virtual Tape

This procedure explains how to install TCP/IP FOR VSE from the extracted .AWS virtual tape. The LIBR utility is used in the installation job.

Required Tools

The IBM VSE Virtual Tape Server must be running before you can install software using this method. This free utility can be obtained from the IBM z/VSE downloads web page: http://www-03.ibm.com/systems/z/os/zvse/downloads.

Information on using the VSE Virtual Tape Server to install the .AWS virtual tape is available at this web page: http://www-03.ibm.com/systems/z/os/zvse/products/vtape.html.

Note:

These websites are maintained by IBM and may change. Contact IBM for assistance with them.

Procedure

Follow these steps to install the software:

1. Copy the unzipped TCPIP*vrm*.AWS file to a folder on the PC running the VSE Virtual Tape Server.
2. Create an installation job that uses LIBR. Copy the following sample job and edit it as described below.

```
// JOB LIBRREST
// OPTION SYSPARM='00',LOG
// ON $CANCEL OR $ABEND GOTO VTAPSTOP
VTAPE START,UNIT=590,LOC=ip-addr:2386, -
X
               FILE='C:\VTAPE\TCPIPvrm.AWS'
MTC REW,590
// ASSGN SYS007,590
// EXEC LIBR,SIZE=256K
RESTORE SUBLIB=CSITCP.TCPPvrm:lib.sublib TAPE=SYS007

/*
/* VTAPE STOP MUST BE AFTER /&
/&
/. VTAPSTOP
VTAPE STOP,UNIT=590
```

Customize this job for your site as follows:

- Change “590” to the virtual tape drive address configured on your VSE system.

- Change “ip-addr” to the address of the remote system (the PC) running the VSE Virtual Tape Server.
- Change “C:\VTAPE\TCPIP*vrm*.AWS” to the actual path of the AWS file on the PC in step 1 above.
- Change “lib.sublib” to the *library.sublibrary* name for your system. The *library* must exist, but the *sublibrary* must not exist. LIBR will create the sublibrary in the library during the restore. The library must contain at least 9000 free blocks.

3. Ensure that the VSE Virtual Tape Server is running, and then run the installation job. If there are no errors, the job restores the entire TCP/IP FOR VSE product.
4. After the restore is complete, it is recommended that you shut down all other stacks. This is needed because a new system control block may be allocated that is shared by all stacks, including non-CSI stacks and the Linux Fast Path stack. See also step 6.

   Note:

   If you are running multiple stacks, it is recommended that they all use the new stack version.
5. It is recommended that you shut down all external partitions that use TCP/IP services. See also step 6.

   Also, ensure that the new installation’s lib.sublib is in the libdef phase search chain of each of these external partitions. This is required to load the newly installed phases.
6. An IPL of the z/VSE system can be performed and will accomplish the shutdown and restart of the stacks and the external partitions described in steps 4 and 5 above.

Go to “Step 3: Verify the Installation” on page 37.

Step 3: Verify the Installation

Verify the Release Level

Cycle TCP/IP FOR VSE and check for the following message. This message verifies the installation's release level.

```
IPN209I CSI Service Pack vv.rr.mm.(yyyy-mm-dd) has been applied. Status is ...
```

You can also use the QUERY VERSIONS command to display this message.

Important:

Make sure that the service pack date in this message (*yyyy-mm-dd*) matches the date in the $Readme.txt file for this installation. If the dates do not match, contact CSI Technical Support.

Verify the Product Phases (Optional)

As an option, follow the steps in the "SHA-1 Phase Verification" procedure to verify the integrity of the product phases you downloaded and installed. This procedure is documented in the "TLS/SSL for VSE" chapter in the *TCP/IP FOR VSE Optional Features Guide*. This procedure guarantees that the SHA-1 values of the installed phases match the original, distributed values.

You must use the file T*vrm*SHA1.JCL that is included in the distribution ZIP file.

The SHA-1 Phase Verification procedure is also available in a standalone document along with the software on CSI's FTP server (ftp://ftp.csi-international.com).

The TLS/SSL for VSE optional feature must be enabled on your system before you can perform this procedure.

Step 4: Install TCP/IP for VSE Optional Features

If you are installing any TCP/IP FOR VSE optional features, see the TCP/IP FOR VSE *Optional Features Guide* for additional information. Some optional features require setup steps beyond activating the feature's product key.

Step 5: Supply the Product Key

Before running TCP/IP FOR VSE in production mode, you must supply a phase that contains one or more product keys. Product keys are provided by CSI International or IBM based on the terms of your license agreements. As each product key approaches expiration, you are notified daily by a console message (IPN594W).

We recommend that you place your production product key in the sublibrary allocated to configuration data, and that you make this sublibrary first in the search order. In this way, applying maintenance updates or reinstalling the product will not overlay your PRODKEYS phase.

Example

You can use the following job stream to install the product key(s) you have obtained from your CSI International account manager. Both standard and optional-feature product keys are installed in this job.

```
// JOB KEY
// LIBDEF *,SEARCH=PRD2.TCPIP
// LIBDEF PHASE,CATALOG=PRD2.CONFIG
// OPTION CATAL
// EXEC ASSEMBLY
    PRODKEY AXED-BEET-CARE-GENT-NAPS     TCP/IP Full System
    PRODKEY AXED-BEET-CARE-GIBE-MYTH     TCP/IP Secure FTP
    PRODKEY BCOY-BEET-CARE-GENT-QUAY     TCP/IP SSL/TLS

         END
/*
// EXEC LNKEDT
/*
/&
```

Notes:

- You can include multiple keys in the PRODKEYS phase by adding additional PRODKEY statements. If you have more than one of our products installed, the key for each product must be represented in the above job stream. You can also place product keys for multiple CPUs in the same PRODKEYS.PHASE member. Doing this allows you to share the PRODKEYS.PHASE member.
- You must cycle TCP/IP FOR VSE if you change the PRODKEYS.PHASE member.
- In the example above, PRD2.CONFIG is the name of the library into which the TCP/IP FOR VSE configuration data is being installed.

- After you have completed a license agreement for the software, you must replace the string of five words with the product key provided by your CSI International representative. The keys that appear in the example above are for illustration only.
- If you obtained your TCP/IP FOR VSE license from IBM, you must also include a CUSTDEF phase. This phase must reside in the same sublibrary as your PRODKEYS phase. Contact your IBM representative for information on generating a CUSTDEF.PHASE.

Step 6: Configure VTAM

If you want to permit access to your VSE VTAM-based applications using TN3270, you must define one or more VTAM application IDs to be used as virtual terminal LU names. One ID (LU name) is required for each concurrent telnet TN3270 session. Each application name is referenced by a corresponding TCP/IP FOR VSE DEFINE TELNETD command.

For your convenience, we provide the library member TCPAPPL.B. This member contains a standard definition for several VTAM applications. You can use this member or provide equivalent definition information yourself.

After the library member is complete, you can add it to the VTAM startup list ATCCON00 or you can use the VTAM VARY command following VTAM initialization.

Example

Here is an example of a VTAM application ID definition.

```
TCPAPPL  VBUILD  TYPE=APPL
TELNLU01 APPL    AUTH=(ACQ),EAS=1
TELNLU02 APPL    AUTH=(ACQ),EAS=1
TELNLU03 APPL    AUTH=(ACQ),EAS=1
TELNLU04 APPL    AUTH=(ACQ),EAS=1
```

Notes:

- TELNLU01 through TELNLU04 define four virtual terminals that are to be used by four separate telnet daemons.
- You must add an APPL statement for each additional virtual terminal you require. “EAS=1” is an optional parameter that tells VTAM that only one LU-LU session is expected for the LU name being defined. If this is omitted, then VTAM reserves an additional 4K of storage when the APPL is opened. This can quickly add up when large numbers of terminals are to be supported.
- The application names are arbitrary. You can choose any convenient value. If you are defining numerous virtual terminals, we recommend that you use virtual terminal names with a prefix and numeric suffix, such as TELNLU*xx*, where *xx* is the numeric suffix. By using this approach, you can more easily define these virtual terminals to TCP/IP FOR VSE.
- VTAM requires one megabyte of dataspace storage for initialization, plus an additional megabyte of dataspace storage for each application. TCP/IP FOR VSE therefore increases by one megabyte the amount of dataspace storage that VTAM obtains. To increase VTAM’s storage,

modify the DSPACE parameter on the EXEC card in the VTAM startup procedure.

Depending on your system dataspace definitions, which can be displayed with the QUERY DSPACE console command, you may need to increase the system dataspace size. The SYSDEF DSPACE command is used for this purpose. If you run multiple TCP/IP partitions, VTAM requires one additional megabyte for each partition running telnet daemons. See the IBM z/VSE documentation for more information on VTAM dataspace requirements.

Step 7: Configure CICS

TCP/IP FOR VSE includes several CICS-based clients. These clients enable CICS users to perform the following tasks:

- Log on from a CICS terminal to other platforms and applications using telnet. For example, a user could log on to a UNIX system from CICS.
- Initiate a file transfer between the TCP/IP FOR VSE FTP client and a remote FTP server using the FTP Interactive Client.
- Initiate a print request between the TCP/IP Line Printer Requester and a remote Line Printer daemon.
- Initiate a Ping request to test network connectivity, or run a TRACERT request to trace the network path to a destination.
- Interactively use TCP/IP FOR VSE'S EMAIL facilities to send email.
- Use the REXEC facility interactively to issue a command on a remote system.
- Probe for the maximum MTU size to a given host.

Set Up the CICS Interface

To set up the CICS interface, you must add the installation lib.sublib to your CICS partition's search chain. To do this, modify your CICS startup JCL as follows:

```
// LIBDEF *,SEARCH=(…,lib.sublib,…)
```

Then, define the programs and transactions to CICS. To do this, modify and run the job stream below.

```
* $$ JOB JNM=DEFINE,CLASS=0,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB DEFINE
// LIBDEF *,SEARCH=(PRD2.TCPIP,PRD1.BASE)
// EXEC DFHCSDUP
MIGRATE TABLE(DFHPPTIP) TOGROUP(TEMP)
COPY GROUP(TEMP) TO(TCPIP) REPLACE
DELETE ALL GROUP(TEMP)
MIGRATE TABLE(DFHPCTIP) TOGROUP(TEMP)
COPY GROUP(TEMP) TO(TCPIP) REPLACE
DELETE ALL GROUP(TEMP)
ADD GROUP(TCPIP) LIST(VSELIST)
/*
/&
* $$ EOJ
```

Notes:

- The PCT and PPT tables are shipped as assembled under CICS/VSE Version 2.3. If you need to reassemble the PCT or PPT statements before using them, the source code is in DFHPCTIP.A and DFHPPTIP.A.
- In the example, PRD2.TCPIP is the installation library and sublibrary containing the TCP/IP FOR VSE phases. If you installed TCP/IP FOR VSE into a different library.sublib, you must make the appropriate changes to the job.
- The use of the group TCPIP and list VSELIST is arbitrary. You can make any adjustments that your site requires.
- The above migration creates the following definitions in the CICS PCT and PPT:

```
DFHPCT TYPE=ENTRY,TRANSID=TRAC,PROGRAM=CLIENT01,RSL=PUBLIC
DFHPCT TYPE=ENTRY,TRANSID=trac,PROGRAM=CLIENT01,RSL=PUBLIC
DFHPCT TYPE=ENTRY,TRANSID=REXE,PROGRAM=CLIENT01,RSL=PUBLIC
DFHPCT TYPE=ENTRY,TRANSID=rexe,PROGRAM=CLIENT01,RSL=PUBLIC
DFHPCT TYPE=ENTRY,TRANSID=DISC,PROGRAM=CLIENT01,RSL=PUBLIC
DFHPCT TYPE=ENTRY,TRANSID=disc,PROGRAM=CLIENT01,RSL=PUBLIC
DFHPCT TYPE=ENTRY,TRANSID=EMAI,PROGRAM=CLIENT01,RSL=PUBLIC
DFHPCT TYPE=ENTRY,TRANSID=emai,PROGRAM=CLIENT01,RSL=PUBLIC
DFHPCT TYPE=ENTRY,TRANSID=PING,PROGRAM=CLIENT01,RSL=PUBLIC
DFHPCT TYPE=ENTRY,TRANSID=ping,PROGRAM=CLIENT01,RSL=PUBLIC
DFHPCT TYPE=ENTRY,TRANSID=TELN,PROGRAM=TELNET01,RSL=PUBLIC
DFHPCT TYPE=ENTRY,TRANSID=teln,PROGRAM=TELNET01,RSL=PUBLIC
DFHPCT TYPE=ENTRY,TRANSID=TELC,PROGRAM=TELNET01,RSL=PUBLIC
DFHPCT TYPE=ENTRY,TRANSID=TELW,PROGRAM=TELNET01,RSL=PUBLIC
DFHPCT TYPE=ENTRY,TRANSID=TELR,PROGRAM=TELNET01,RSL=PUBLIC
DFHPCT TYPE=ENTRY,TRANSID=FTP,PROGRAM=FTP01,RSL=PUBLIC
DFHPCT TYPE=ENTRY,TRANSID=ftp,PROGRAM=FTP01,RSL=PUBLIC
DFHPCT TYPE=ENTRY,TRANSID=FTPC,PROGRAM=FTP01,RSL=PUBLIC
DFHPCT TYPE=ENTRY,TRANSID=FTPW,PROGRAM=FTP01,RSL=PUBLIC
DFHPCT TYPE=ENTRY,TRANSID=FTPR,PROGRAM=FTP01,RSL=PUBLIC
DFHPCT TYPE=ENTRY,TRANSID=TCPC,PROGRAM=CLIENT01,RSL=PUBLIC
DFHPCT TYPE=ENTRY,TRANSID=TCPW,PROGRAM=CLIENT01,RSL=PUBLIC
DFHPCT TYPE=ENTRY,TRANSID=TCPR,PROGRAM=CLIENT01,RSL=PUBLIC
DFHPCT TYPE=ENTRY,TRANSID=LPR,PROGRAM=CLIENT01,RSL=PUBLIC
DFHPCT TYPE=ENTRY,TRANSID=lpr,PROGRAM=CLIENT01,RSL=PUBLIC
DFHPPT TYPE=ENTRY,PROGRAM=TELNET01,RSL=PUBLIC,PGMLANG=ASSEMBLER
DFHPPT TYPE=ENTRY,PROGRAM=FTP01,RSL=PUBLIC,PGMLANG=ASSEMBLER
DFHPPT TYPE=ENTRY,PROGRAM=CLIENT01,RSL=PUBLIC,PGMLANG=ASSEMBLER
```

Notes for Users of the VSE Interactive User Interface

A CICS Node Error Program (NEP) should be installed. NEP is useful in cases where users who telnet into CICS do not log off cleanly. Occasionally, a user remains logged on to the Interactive Interface or ICCF after terminating a telnet session. The NEP allows CICS to log these users off automatically, thus freeing their sessions.

The easiest way to install NEP is to use the sample NEP, which is found in ICCF library 59 in member IESZNEP. Follow the instructions in that member. If you already have an NEP installed, the member IESZNEPS or IESZNETX might be more appropriate.

Step 8: Configure TCP/IP

Before you initially define TCP/IP FOR VSE, you must collect the following configuration information:

Category	Required Configuration Information
Links	Identification information for each device, controller, and connection mechanism TCP/IP FOR VSE is to use for external communication.
Daemons	Definition information for each TCP/IP FOR VSE server (daemon) you plan to use. All of the standard servers such as FTPD and TELNETD should be selected ahead of time, along with their many options.
Routing	Routing information for TCP/IP FOR VSE. This information enables TCP/IP to route IP datagrams through the network to their destination. Your Network Administrator may need to assist you in determining the necessary ROUTE details for your environment.
General	General configuration information. For example, you must specify the HOST IP address for TCP/IP FOR VSE.

There are also file systems to define, security to be considered, and a myriad of other details you can identify by reviewing the various commands available. See the *TCP/IP FOR VSE Command Reference* for details.

TCP/IP FOR VSE contains commands and parameters that enable you to control all aspects of its configuration and operation. These commands should be contained in one or more configuration files for automatic execution during stack initialization. For testing, you can choose to omit one or more commands from the configuration file and enter them from the console. This process allows you to measure each command's impact.

To develop your custom configuration, begin with the VSE library member IPINIT*xx*.L, where *xx* is the TCP/IP FOR VSE system identifier (SYSID).

Note 1:

Do not confuse this member with the IPINITXX.L example file in the installation library. Most of the definition commands in the example file are commented out. The IPINIT*xx*.L mentioned above is created from the IBM interactive installation. If you do not use that facility to create an IPINIT*xx*.L member, you can use the provided example file as a guide.

Note 2:

Place your configuration member(s) in a sublibrary that is different from the installation library. You will need to add this configuration sublibrary to the LIBDEF search chain.

Library Initialization Member

During initialization, TCP/IP FOR VSE reads a member from the partition's source library search chain. This library member (an 'L' book), also known as the initialization deck, contains configuration commands that TCP/IP FOR VSE executes during initialization.

The name of this member is one of the following:

- The value you coded for INIT= in the IPNET execution parameter list,

  –or–

- If INIT= is not coded, IPINIT*xx*.L, where *xx* is the TCP/IP FOR VSE system identifier (SYSID).

The SYSID is coded in the // EXEC IPNET,PARM='ID=*xx*,...' JCL statement that is used to start the product. If you choose to run multiple stacks on a single VSE system, each SYSID must be unique. If you have two different VSE systems, and each one has only one stack, then each SYSID can be the same or different—the choice is up to you. The default identifier is 00.

Caution:

If you set ID= to a value other than 00, you must specify this value to all TCP/IP FOR VSE-enabled applications you use. By default, all socket application calls use 00 as the system identifier.

The TCP/IP FOR VSE initialization member can become large and difficult to manage. To help manage this member, you can create separate members that contain routing definitions, user definitions, file definitions, or telnet daemon definitions and then include these members using the INCLUDE command. See the TCP/IP FOR VSE *Command Reference* for details on this command. As an example, see "Using the FILESYS Program," page 29, for information on including a file-definition member in your initialization deck.

See also "Appendix B: Quickstart Guide," page 230, for more information about configuring TCP/IP FOR VSE.

Optional Materials

The following materials are available from CSI International's website:

- Documentation
- Samples
- Preventive maintenance software.

Documentation

A complete set of TCP/IP FOR VSE documentation is supplied with the product from CSI International at www.csi-international.com/products/zVSE/TCP-IP/TCP-IP.htm .

The downloaded PDF files can be viewed using Adobe® Reader® software or another PDF viewer.

Preventive Maintenance

Preventive maintenance software is provided in service packs and fixes.

Each service pack contains cumulative maintenance. Service packs are in either beta or production status. Once a beta service pack undergoes a formal internal quality assurance test, and if all goes well at customer early-test sites, the service pack is promoted to production status. To apply a service pack, install it the same way you install the product.

Once a service pack is in production status, maintenance fixes for individual components are created to correct reported problems. Once a fix is confirmed to correct a confirmed error, it is made available for download from the CSI FTP server. The fixes can then be downloaded and applied.

Registered customers can obtain beta service packs or fixes at any time.

For your convenience, CSI upgrades the current service pack periodically by incorporating any outstanding fixes. Before running the installation job, check CSI International's website to be sure you have the latest version. You may find that reinstalling the updated software is a simpler task than downloading and installing separate fixes.

If you received your copy of TCP/IP FOR VSE from IBM, IBM's method requires you to install the product and apply maintenance using PTFs applied with MSHP.

4

Link Configuration

Overview

When you install and configure TCP/IP FOR VSE, your first task is to establish communications between your mainframe and your network or networks. To do this, you must take the following actions:

1. Decide how your mainframe is going to communicate with your network(s). TCP/IP FOR VSE uses links to communicate. In general, a *link* is a hardware device that connects to (1) the mainframe using a standard bus and tag or ESCON channel and (2) the network using a standard Ethernet, FDDI, or Token Ring connection.

2. Establish the link between the mainframe and the network(s). There are several steps required to establish a link. The steps you need to take depend on the type of link and your hardware and software.

Hardware

You must select and install the physical hardware. This can consist of an open systems adapter, a channel-to-channel (CTC) adapter, or one of many non-IBM adapters. For a list of communications adapters supported by TCP/IP FOR VSE, see chapter 2, "Planning for Installation," on page 16.

IOCP

In most cases, you need to modify your input/output configuration program (IOCP) to recognize the new hardware. Because many installations require considerable notice before an IOCP gen, you may want to plan ahead.

VM

If VSE is running in a virtual machine under VM, you need to define the device to VM and then dedicate it to your VSE virtual machine. You can allow VM to sense the device, or you can use the SET RDEVICE command to do this dynamically, or you can use the VM HCPRIO file.

VSE

Before you can use a new device, you must define it to VSE. To do this, use the VSE ADD statements in member $IPL*xxx*.PROC in the IJSYSRS.SYSLIB sublibrary.

TCP/IP

Finally, you must define the hardware to TCP/IP FOR VSE. To do this, use the DEFINE LINK command. If you are using an LCS, open systems adapter, or an equivalent device, you may also need to include one or more DEFINE ADAPTER commands.

After you define your devices to TCP/IP FOR VSE, you must also add DEFINE ROUTE commands to specify which links access which network addresses.

The remainder of this chapter addresses hardware types individually.

Generic LCS Controllers

A number of devices emulate LAN Channel Station (LCS) protocols. In general, you follow the same procedure to define all of these devices. We discuss the procedure in this section and mention the differences.

Products

Some devices that support LCS protocols are as follows:

- Bustech Netshuttle for VSE. This device is an inexpensive single adapter control unit. It supports a single attachment to the mainframe and a single 10-Mbps Ethernet attachment to the LAN. Other models of the Bustech Netshuttle line, including the Netshuttle 110, Netshuttle 120, and Netshuttle 140, also support TCP/IP PASSTHRU.
- IBM 3172. The 3172 supports up to four adapters, sequentially numbered 0 through 3. The adapter types can be Ethernet, Token Ring, or FDDI. The 3172 uses one of the following two operating systems for TCP/IP PASSTHRU:
  - The Interconnect Communications Program (ICP).
  - The Internet Protocol Channel Communications Program (IPCCP). This operating system, which uses the CLAW protocol, does not apply to this section.
- DBM's Open Systems Adapter (OSA/OSA-2).
- Polaris Communications StarGate with Fast Packet 3172 emulation.
- IBM 2216 N-Ways Multi-Access Control Unit.

In addition, there are other control units that support TCP/IP PASSTHRU using LCS protocols.

LCS control units are stand-alone devices that support from one to *n* LAN connection adapters. All LCS control units appear to the mainframe as two adjacent devices, one for input and one for output. The first address is always an even number, and it is this number that identifies the device to TCP/IP FOR VSE. These adapters can be any mixture of the adapter types listed in the next section.

Adapter Types

The allowable adapter types for the DEFINE ADAPTER command are listed below. This command is used to define a generic LCS adapter to TCP/IP FOR VSE, as described in the next section.

TYPE= Value	Adapter	Description
ETHERNET	Ethernet 802.3	The Ethernet LAN adapter allows connection to other Ethernet-equipped devices.

TYPE= Value	Adapter	Description
TOKEN_RING	Token Ring 802.5	The token ring adapter allows connections to single- or multi-ring Token Ring networks. Token Ring adapters typically transfer data at 4 or 16 Mbps.
FDDI	FDDI	A fiber optic data device interface adapter provides a transfer rate of 100 Mbps.

Defining to TCP/IP

To explicitly define an adapter for use by TCP/IP FOR VSE, you must use the DEFINE ADAPTER command along with the DEFINE LINK command. The following example shows definitions for three adapters in an OSA device. You can specify OSA, OSA2, LCS, or 3172 as the TYPE= value on the DEFINE LINK command for any of the generic LCS adapters.

```
DEFINE LINK,ID=LINK01,TYPE=OSA,DEV=500,MTU=1500
DEFINE ADAPTER,LINKID=LINK01,NUMBER=0,TYPE=ETHERNET
DEFINE ADAPTER,LINKID=LINK01,NUMBER=1,TYPE=TOKEN_RING,MTU=2000
DEFINE ADAPTER,LINKID=LINK01,NUMBER=2,TYPE=FDDI,MTU=3000
```

Notes:

- Three adapters are defined. The NUMBER= value of an adapter corresponds to its physical location within the OSA. A different adapter type is used in each case.
- The MTU sizes should match the ones you selected in Chapter 2, "Planning for Installation," page 16.
- See the *TCP/IP FOR VSE Command Reference* for details on the DEFINE LINK and the DEFINE ADAPTER commands.
- If you are using the IBM 2216 N-Ways Multi-Access Control Unit, the 2216 console command LIST NETS might help to determine the proper value for NUMBER.

Defining to VSE

The OSA appears as two adjacent devices, and it follows the same rules as CTC adapters. The device definition depends on your VSE release level. The following definition is for VSE/ESA 2.1 and above releases.

```
ADD 500:501,CTCA,EML
```

If this device is under VM control, you may need to add the EML parameter so that VSE bypasses inquiry on device status.

Defining to IOCP: Bus and Tag

An LCS control unit is defined to the IOCP as two adjacent devices. The addresses must be sequential and must begin with an even address, as shown in this example:

```
CHPID    PATH=(21),TYPE=BL
CNTLUNIT CUNUMBR=001,PATH=(21),UNITADD=((00,8)),         X
         UNIT=3088,PROTOCOL=S4,SHARED=N
IODEVICE ADDRESS=(500,32),CUNUMBR=001,UNIT=CTC,          X
         UNITADD=00,STADET=N,TIMEOUT=N
```

Notes:

- If you are defining an OSA to IOCP, the UNITADD parameter MUST specify X'00' and X'01' for data transfer across port 0. If you have two physical ports available, you MUST specify a unit address of X'02' and X'03' for the device pair that transfers data across port 1.
- You can SHARE the OSA port between multiple partitions in an LPAR environment. See the appropriate IBM documentation for information about this option.
- The LINK number specifies the link address (ESCD port number) to which the IBM 3172 is connected. See IBM manual number SC30-3572-02 for more information about 3172 IOCP considerations.
- If you have an IBM N-Ways 2216 Multi-Access Control Unit, the IODEVICE UNIT parameter should be set to 3172 (that is, UNIT=3172). See IBM manual number SC30-3886-02 for more information about obtaining the IOCP for a 2216.

Defining to IOCP: ESCON

An LCS control unit is defined to the IOCP as two adjacent devices. The addresses must be sequential and must begin with an even address, as shown in the following example.

```
CHPID    PATH=(30),TYPE=CNC
CNTLUNIT CUNUMBR=500,PATH=(30),UNITADD=((00,8)),         X
         UNIT=3172,LINK=C1,CUADD=1
IODEVICE ADDRESS=(500,8),CUNUMBR=500,UNIT=SCTC,          X
         UNITADD=00
```

This example defines a full series of eight devices. Although you may not need this many devices now, it allows for future expansion.

Notes:

- If you are defining an OSA to IOCP, the UNITADD parameter MUST specify X'00' and X'01' for data transfer across port 0. If you have two physical ports available, you MUST specify a unit address of X'02' and X'03' for the device pair that transfers data across port 1.
- You can SHARE the OSA port between multiple partitions in an LPAR environment. See the appropriate IBM documentation for information about this option.
- The LINK number specifies the link address (ESCD port number) to which the IBM OSA is connected. See IBM manual number SC30-3572-02 for more information about 3172 IOCP considerations.
- If you have an IBM N-Ways 2216 Multi-Access Control Unit, the IODEVICE UNIT parameter should be set to 3172 (that is, UNIT=3172). See IBM manual number SC30-3886-02 for more information about generating the IOCP for a 2216.

Defining to VM

You must define the LCS control unit to VM as a 3088. You can include the definition in the SYSTEM CONFIG file. VM also has the capability to sense the device automatically. This means that if you omit the statement, the device is still recognized and used. The following is a sample definition.

```
RDEVICE 500-501 TYPE CTCA
```

You must also include the appropriate information in the VSE virtual machine’s directory entry. The following is a sample entry:

```
DEDICATE 500 500
DEDICATE 501 501
```

CLAW Interface Devices

There are a number of devices that use a protocol called the Common Link Access to Workstation (CLAW) interface. Some devices that use the CLAW interface are:

- RS/6000 with a block multiplex channel attachment
- Cisco® 7500 series router with a channel interface processor (CIP) card
- Cisco 7200 series router with a CPA card
- IBM 3172 Interconnect Controller running the IPCCP operating system.

In all cases, there is a stand-alone processor at the other end of the CLAW interface that can be connected to a VSE mainframe with a channel attachment. TCP/IP FOR VSE passes IP traffic directly to the attached processor using a channel connection and channel speed, thus resulting in a very high speed connection.

The CLAW device appears to the mainframe as two adjacent devices, one for input and one for output. The first address is always an even number, and it is this number that identifies the device to TCP/IP FOR VSE.

Defining to TCP/IP

Define a CLAW to TCP/IP as follows:

```
DEFINE LINK,ID=LINK01,TYPE=CLAW,DEV=500,MTU=1500, -
       HOSTNAME=VSE,HOSTAPPL=TCPIP, -
       WSNAME=itsname,WSAPPL=TCPIP
```

Notes:

- The reference to device 500 also causes the assignment of device 501.
- The WSNAME parameter must match the Remote Host Name parameter in the Add a Subchannel panel in the SMIT BLKMUX installation utility on the RS/6000.
- For a Cisco router with a CIP card or a Cisco router with a CPA card, the CLAW interface is configured with commands entered directly on the router. Documentation is available at the vendor's website. Information on configuring the CLAW interface can be found at http://www.cisco.com/c/en/us/td/docs/ios/bridging/configuration/guide/15_0sy/br_15_0sy_book/br_claw_tcpip_offld.html.
- If you have an IBM 3172, consider running the 3172 using the ICP operating system. See the section on configuring for the LCS.

Defining to VSE

The CLAW appears as two adjacent device addresses and follows the rules of CTC adapters. The device definition depends on your VSE/ESA release level.

VSE/ESA 2.1 and Above

Use the following definition for this release level:

```
ADD 500:501,CTCA,EML
```

We recommend that you use the EML parameter so that VSE bypasses inquiry on device status.

VSE/ESA 1.4

Use the following definition for this release level:

```
ADD 500:501,3705,10,EML
```

This statement defines the devices in a manner that provides for channel translation but does not involve the missing interrupt handler.

Defining to VM

The CLAW appears as two adjacent devices and follows the rules of CTC adapters. The following box contains a sample of statements that you should add to the VM/ESA SYSTEM CONFIG file.

```
RDEVICE 500-501 TYPE CTCA
```

Defining to IOCP: Bus and Tag

The CLAW device is defined to the IOCP as two adjacent 3088s. The addresses must be sequential and must begin on an even boundary.

```
CHPID    PATH=(21),TYPE=BL
CNTLUNIT CUNUMBR=500,PATH=(21),UNITADD=((00,8)),        X
         UNIT=3088,PROTOCOL=S4,SHARED=N
IODEVICE ADDRESS=(500,8),CUNUMBR=001,UNIT=CTC,          X
         UNITADD=00,STADET=N,TIMEOUT=N
```

The example defines a series of eight devices. Although you may not need this many devices now, it allows for future expansion.

Defining to IOCP: ESCON

The CLAW is defined to the IOCP as an RS6K. The addresses must be sequential and must begin on an even boundary. This is shown in the following example.

```
CHPID    PATH=(21),TYPE=CNC
CNTLUNIT CUNUMBR=500,PATH=(21),UNITADD=((00,8)),      X
         UNIT=RS6K,LINK=CA
IODEVICE ADDRESS=(500,8),CUNUMBR=001,UNIT=SCTC
```

Notes:

- This example defines a series of eight devices. Although you may not need this many devices now, it allows for future expansion.
- Configuring the CLAW on the mainframe is easy. Configuring the CLAW on the RS6000 may not be. The utility you need to use is called SMIT. In addition to the standard manuals that explain SMIT, you may want to look at the following IBM manuals:
    - *RISC/6000 to Mainframe Using S/370 Channel Connections* (manual number SG24-4589)
    - *Block Multiplex User's Guide and Service Information* (manual number SC31-8196)

Virtual CTC Adapter: Connecting Under VM

To establish communication between TCP/IP FOR VSE and another copy of TCP/IP FOR VSE or TCP/IP for VM or MVS, you can use a virtual CTC adapter (CTCA). (This provides, of course, that both TCP/IPs are executing in virtual machines under the same VM image.) The connection requires two CTCAs: one for input, and one for output.

This section describes how to establish a virtual CTC connection when both TCP/IP implementations are running under VM control.

Defining to TCP/IP

Define a CTC adapter to TCP/IP FOR VSE as follows:

```
DEFINE LINK,ID=LINK01,TYPE=CTCA,DEV=500,STOPPED
```

Notes:

- The STOPPED operand prevents the link from initializing until a START LINK command is issued. If the other end of the VCTC connection is down, TCP/IP FOR VSE continues to try the link until the other end is restarted. Each time TCP/IP FOR VSE attempts a restart, it sends error messages to your VSE console. The STOPPED operand prevents this from happening. Alternatively, you can alter the amount of time between restart attempts with the RETRY_TIME parameter on the DEFINE LINK. See the *TCP/IP FOR VSE Command Reference* for more information about the RETRY_TIME parameter.
- It is important to note that the reference to device address 500 causes devices 500 and 501 to be assigned within TCP/IP FOR VSE.
- TCP/IP FOR VSE uses a special protocol to communicate with other implementations of TCP/IP FOR VSE. This allows TCP/IP FOR VSE to easily determine when the other side of the connection is initializing and terminating. Unfortunately, MVS and VM do not have this special protocol, so TCP/IP FOR VSE must rely on I/O errors to determine when an MVS, OS/390, or VM connection has been dropped. This can result in delays in determining that a link has terminated.

Defining to VSE

The virtual CTC adapter appears as two adjacent devices, and it follows the rules of CTC adapters. The device definition depends on your VSE release level. The following definition is for VSE/ESA 2.1 and above releases.

```
ADD 500:501,CTCA,EML
```

You need to add the EML parameter so that VSE bypasses inquiry on device status. This prevents problems when the CTCs are not already coupled at VSE IPL time.

Defining to VM

A virtual CTC adapter is defined to a virtual machine as two adjacent device addresses. Virtual CTCAs are created by using the SPECIAL command in each virtual machine's directory entry or by using the CP DEFINE CTCA command. The following example assumes that TCP/IP FOR VSE is executing in the virtual machine VSETEST and that we are connecting to IBM's TCP/IP running on virtual machine IBMTCP.

```
The following is added to the VSEPROD directory entry:
SPECIAL 500 3088
SPECIAL 501 3088

The following is added to the VMTCPIP directory entry:
SPECIAL 500 3088
SPECIAL 501 3088
```

In this example, we arbitrarily selected addresses 500 and 501 on each virtual machine. There is no requirement that they be identical values.

Coupling the Adapters

Once defined, the CTC adapters must be coupled before traffic can flow. The following example shows the commands issued from the VSEPROD virtual machine to couple it to the VMTCPIP virtual machine.

```
COUPLE 500 TO VMTCPIP 501
COUPLE 501 TO VMTCPIP 500

And from the VMTCPIP virtual machine:

COUPLE 500 to VSEPROD 501
COUPLE 501 to VSEPROD 500
```

Note:

We cross-couple an odd address to an even address. This is an absolute requirement, and problems will occur if you do not do this.

You can easily automate this process on both the VM and VSE sides. You probably already have a PROFILE EXEC that runs when you log on (or autolog on) to your VSE virtual machine, and this is an excellent place to add the above statements. On the TCP/IP for VM side, you can use either the PROFILE EXEC on the TCP/IP virtual machine or the TCP/IP for VM initialization exit. Although you need to issue the COUPLE command only once, it is best to have it execute on both the VM side and the VSE side so that either side can be brought up first.

Defining to TCP/IP on VM

The following example illustrates the configuration statements you need to place in the TCP/IP on VM's initialization deck. These statements have been tested with TCP/IP on VM Version 2, Release 3, and TCP/IP for VM FL310. We assume that the CTCs in use are at addresses 500 and 501 (as seen by TCP/IP for VM). We also assume that TCP/IP FOR VSE is not serving as a gateway. In other words, TCP/IP for VM uses the CTC connection only for traffic whose final destination is TCP/IP FOR VSE.

```
DEVICE CTC1 CTC 500
*  Define the CTC devices and assign symbolic name CTC1.
LINK CTCVSE CTC 0 CTC1
*  Define the CTC link and assign symbolic name CTCVSE.
*  The value 0 is the link number. It must be numeric and
*  it must be unique.
. . .
HOME nnn.nnn.nnn.nnn CTCVSE
*  The designated IP address is that of VM's TCP/IP.
*  You must add a HOME statement for the new link in VM TCP/IP.
. . .
GATEWAY mmm.mmm.mmm.mmm  = CTCVSE nnnn HOST
*  There may be several other statements under GATEWAY. The above
*  statement is added. The IP address is the one assigned to TCP/IP
*  for VSE. This is the value usually found in the TCP/IP for VSE
*  initialization statement SET IPADDR. The nnnn is the MTU size.
*  Selecting the correct MTU size is a complex subject and is
*  discussed in Chapter 2, "Planning for Installation," and
*  Chapter 12, "Performance."
. . .
START CTC1
*  This statement tells VM TCP/IP to begin processing traffic on the
*  CTC link.
```

Real CTC Adapter or 3088 MCCU

A CTCA connection can be established between two copies of TCP/IP FOR VSE running on two different processors or between TCP/IP FOR VSE and TCP/IP executing under VM, OS/390, or MVS running on two different processors. This connection requires two CTC adapters: one for input and one for output.

If you are trying to define a virtual CTCA connection, you should follow the instructions in the previous section.

Defining to TCP/IP

You can use the following command to define a CTC adapter interface to TCP/IP.

```
DEFINE LINK,ID=LINK01,TYPE=CTCA,DEV=(500,501)
```

Both device addresses are listed in order to control the sequence of use. The first listed device is used for input, and the second listed device is used for output. At the “far end of the pipe,” the devices must attach in reverse order. This means that the “other end” of device 500 must be the second device in the DEFINE LINK specified by the target TCP/IP FOR VSE.

Defining to VSE

The CTC adapter appears as two adjacent device addresses, and follows the rules of CTC adapters. The device definition depends on your VSE release level. The following definition is for VSE/ESA 2.1 and above releases.

```
ADD 500:501,CTCA
```

Defining to VM

The CTC adapter appears as two adjacent device addresses, and follows the rules of CTC adapters. The device definition depends on your VM release level. In either case, you must also ATTACH the device (both addresses) to the VSE virtual machine. The following definition is for VM/ESA 1.2 and above releases.

```
RDEVICE 500-501 TYPE CTCA
```

You can add this statement to the SYSTEM CONFIG file, or you can allow VM/ESA to sense the device automatically.

Defining to IOCP

The CTC Adapter is defined to the IOCP as two adjacent 3088s. The addresses must be sequential and must begin on an even boundary. This is shown in the following example.

```
CHPID    PATH=(21),TYPE=BL
CNTLUNIT CUNUMBR=001,PATH=(21),UNITADD=((00,32)),        X
         UNIT=3088,PROTOCOL=S4,SHARED=N
IODEVICE ADDRESS=(500,32),CUNUMBR=001,UNIT=CTC,          X
         UNITADD=00,STADET=N,TIMEOUT=N
```

This example defines a full series of 32 devices. Although you may not need this many devices now, it allows for future expansion.

OSA Express

The OSA Express adapter provides exceptional performance and can be used to communicate over a network, between virtual machines, and between LPARs.

Defining to TCP/IP

To use the OSA Express adapter in queued direct I/O (QDIO) mode, use the following definition:

```
DEFINE LINK,ID=link_id,TYPE=OSAX,DEV=(cuu1,cuu2),DATAPATH=cuu3, -
       IPADDR=ipaddr,MTU=mtu_size
```

The two device addresses (cuu1, cuu2) must be an even/odd pair. If cuu2 is omitted, cuu1 + 1 is used as the default. These addresses control the sequence of use. The first listed device is used for input, and the second is used for output. The following is an example:

```
DEFINE LINK,ID=OSAX_01,TYPE=OSAX,DEV=(920,921),DATAPATH=922, -
       IPADDR=9.164.155.99,MTU=1500
```

You can also specify the following parameters for this link type. These parameters are not processed by the TCP/IP FOR VSE stack and are used as required by the adapter.

Parameter	Description
ALTIP	Assigns up to nine more IP addresses if z/VSE is a multi-homed host or if the TCP/IP FOR VSE stack serves as a gateway to other stacks.
ROUTER	Allows for capturing traffic addressed to “unknown” hosts and applying routing parameters. Values: Primary; Secondary; None (default).
OSAPORT	Specifies a port if two ports per CHPID are supported (default is 0).
PORTNAME	Specifies the symbolic name of the OSA Express port to be used with this link. This parameter is obsolete for most OSAX adapters.

For more information on these parameters, see DEFINE LINK in the *TCP/IP FOR VSE Command Reference* along with the IBM-provided documentation for your adapter.

To use an OSA Express adapter in non-QDIO mode, you must define the link as you would for a conventional OSA adapter. See the section “Generic LCS Controllers,” page 51, for more information.

Defining to VSE

Consult the IBM-provided documentation for your machine model and VSE level to install and configure your OSA Express adapter.

OSA Express Performance

You must choose parameter values carefully to optimize this adapter's performance. Depending on installation, this adapter supports MTU sizes up to 64KB. For communication over network segments that support large sizes, the larger the MTU size, the better. This is especially true when you are communicating with virtual machines or LPARs because the data moves memory to memory. If the datagrams are to flow over a physical network such as an Ethernet segment, an MTU size larger than 1500 may result in fragmentation, excessive retransmission, and poor performance. See chapter 12, "Performance," for more information on choosing an optimal MTU size.

If you use your OSA Express to communicate in both inter- and intra-machine modes, you should supply DEFINE ROUTE commands to ensure appropriate MTU values for each destination's class.

Cross-Partition Connections

TCP/IP FOR VSE provides a mechanism for connecting multiple TCP/IP partitions. This feature can be useful in some circumstances. For example, you can use this feature to create production and test partitions and to communicate between them. In this section, we explain how to set up cross-partition connections and discuss considerations that you should be aware of.

Although we document this feature, we do not necessarily recommend that you use it. Cross-partition connections are likely to cause extremely poor performance, especially if the two TCP/IP partitions do not have equal priority.

CSI International does not recommend using the cross-partition feature to separate FTP and TN3270 workloads. See “Performance Factors” in chapter 12, “Performance,” for more information on FTP and TN3270 performance.

Connecting Two Partitions

To connect two partitions, you can use one of the following methods:

- If the two TCP/IP FOR VSE partitions execute in separate VSE images under the same copy of VM/ESA, you can use a virtual channel-to-channel adapter. This is the most common method.
- If the two TCP/IP FOR VSE partitions execute in the same VSE image, you can use a TYPE=IPNET connection.

To set up additional TCP/IP partitions, you can use one of the following methods:

- Connect the second TCP/IP FOR VSE partition directly to the network and treat each TCP/IP for VSE partition separately. This is the simplest method.
- Route TCP/IP FOR VSE traffic from one partition to the other before it goes out to your network. You must use this method if you do not have the physical resources to use the first method. This section is concerned with this second method.

Before continuing, be sure to understand the following terms:

- The *primary TCP/IP FOR VSE partition* is the partition that owns the communication adapter to your network.
- The *secondary TCP/IP for VSE partition* is the partition that does not own any physical links. This partition communicates directly and only to the primary TCP/IP FOR VSE partition using a virtual channel-to-channel adapter or a TYPE=IPNET connection.

The discussion in this section also assumes that you already have a primary partition in place. We assume that the primary partition communicates with your network over your physical control unit and that you are adding a secondary partition.

Specifying to TCP/IP

Let us start by looking at an example of how to set up a cross-partition connection. This example demonstrates how to tell TCP/IP FOR VSE that you want a cross-partition connection.

Each copy of TCP/IP FOR VSE operates under a system ID. You assign this ID on the ID= subparameter of the EXEC statement PARM= parameter. This value can also form the final two positions of the name of the default initialization member name. In this section, however, we use the INIT= subparameter to explicitly supply the member name.

In the initialization statements for each TCP/IP FOR VSE partition, you must include a DEFINE LINK command for each separate TCP/IP FOR VSE system with which you need cross-partition communication.

System 00

The following job contains definitions for system 00.

```
// JOB TCPIP0
// LIBDEF *,SEARCH=(PRD2.CONFIG,PRD2.TCPIP)
// SETPFIX LIMIT=400K
// EXEC IPNET,SIZE=IPNET,PARM='ID=00,INIT=IPINIT00'
/&
```

The following TCP/IP definition establishes a connection to the other TCP/IP partition:

```
DEFINE LINK,ID=LINK01,TYPE=IPNET,SYSID=01,STOPPED
```

The above definition is stored in a library member named IPINIT00.L. Note that the STOPPED parameter prevents attempts at link initialization until the START LINKID command is issued. This is especially useful when this TCP/IP partition is to be the first started (or the only one under normal circumstances) and you do not want to generate error messages or go through initialization retry. You can use the RETRY_TIME parameter to govern the period of time that TCP/IP FOR VSE waits between attempts.

System 01

The following job contains definitions for system 01.

```
// JOB TCPIP1
// LIBDEF *,SEARCH=(PRD2.CONFIG,PRD2.TCPIP)
// SETPFIX LIMIT=200K
// EXEC IPNET,SIZE=IPNET,PARM='ID=01,INIT=IPINIT01'
/&
```

The following TCP/IP definition establishes a connection to the other TCP/IP partition:

```
DEFINE LINK,ID=LINK01,TYPE=IPNET,SYSID=00,STOPPED
```

The above definition is stored in a library member named IPINIT01.L.

Connection Considerations

In this section, we discuss items to consider when setting up a cross-partition connection. These considerations are from four perspectives:

- The primary TCP/IP FOR VSE partition
- The secondary TCP/IP FOR VSE partition
- Other TCP/IP FOR VSE hosts
- TCP/IP FOR VSE partitions with multiple links (multi-homing)

Primary Partition

When you set up a cross-partition connection, you need to add the following statements to the primary TCP/IP FOR VSE partition:

```
DEFINE LINK,ID=LINKSEC,TYPE=CTCA/IPNET,DEV=xxx,MTU=mtusize
DEFINE ROUTE,ID=RTESEC,IPADDR=xxx.xxx.xxx.xxx,LINKID=LINKSEC
DEFINE ALTIP,IPADDR=xxx.xxx.xxx.xxx
```

Each statement is explained in the following table.

Command	Explanation
DEFINE LINK	Defines the required link to the secondary partition. See chapter 2, “Planning for Installation,” and chapter 12, “Performance,” for details on selecting an MTU.
DEFINE ROUTE	Defines the required static route definition to the secondary partition. Variable *xxx.xxx.xxx.xxx* is the IP address of the secondary partition.

Command	Explanation
DEFINE ALTIP	Allows the primary partition to answer ARP requests on behalf of the secondary partition. Variable *xxx.xxx.xxx.xxx* is the IP address of the secondary partition. See DEFINE ALTIP in the *TCP/IP FOR VSE Command Reference.*

Secondary Partition

When you set up a cross-partition connection, you need to add the following statements to the secondary TCP/IP FOR VSE partition:

```
DEFINE LINK,ID=LINKPRIM,TYPE=CTCA/IPNET,DEV=xxx,MTU=mtusize
DEFINE ROUTE,ID=RTEALL,IPADDR=0.0.0.0,LINKID=LINKPRIM
```

Each statement is explained in the following table.

Command	Explanation
DEFINE LINK	Defines the required link to the primary partition. See Chapter 2, “Planning for Installation,” for information about MTU size. The MTU size should match the MTU size of the communications link to your network in the primary partition.
DEFINE ROUTE	Tells TCP/IP FOR VSE that you have one physical connection in this TCP/IP FOR VSE partition and that you want all traffic routed through it. It is not necessary to include the GATEWAY= parameter on this DEFINE ROUTE statement.

Other Hosts

You may need to add static route definitions to other hosts on your network. The DEFINE ALTIP statement enables the primary partition to answer ARP requests for the secondary partition, but it does so only for hosts on the same physical network. For other hosts, you must supply the required routing information. For example, on a Windows® host, you may need the following static route command:

```
ROUTE ADD xxx.xxx.xxx.xxx yyy.yyy.yyy.yyy
```

The variables have the following meanings:

- *xxx.xxx.xxx.xxx* is the IP address of the secondary partition
- *yyy.yyy.yyy.yyy* is the IP address of the primary partition

You can accomplish the same thing more easily by adding the correct routing definitions to your gateways and using domain names.

Multi-homing

TCP/IP FOR VSE supports multi-homing, which means that a single stack can be attached directly to multiple networks. Each attachment point (adapter) requires a unique IP address. To configure a multi-homed system, you must use a combination of the following statements:

- IPADDR parameter on the DEFINE LINK statement
- DEFINE ADAPTER statement
- DEFINE MASK statement
- Appropriate DEFINE ROUTE statements.

For example, assume that TCP/IP FOR VSE is part of a class C network with IP address 192.168.0.1 and subnet mask 255.255.255.0 over a Bustech Netshuttle for VSE at physical address 600. TCP/IP FOR VSE is also part of a class A network with IP address 10.0.0.1 and subnet mask 255.0.0.0 over a CLAW interface at physical address 700. You would use the following statements to configure this system:

```
DEFINE LINK,ID=BUSTECH,TYPE=3172,DEV=600
DEFINE ADAPTER,ID=CARD0,LINKID=BUSTECH,NUMBER=0,IPADDR=192.168.0.1
DEFINE LINK,ID=CLAW,TYPE=CLAW,DEV=700,IPADDR=10.0.0.1
DEFINE MASK,ID=MASKC,NETWORK=192.168.0.1,MASK=255.255.255.0
DEFINE MASK,ID=MASKA,NETWlORK=10.0.0.1,MASK=255.0.0.0
DEFINE ROUTE,ID=RBUSTECH,IPADDR=192.168.0.0,LINKID=BUSTECH,ADAPTER=0
DEFINE ROUTE,ID=RCLAW,IPADDR=10.0.0.0,LINKID=CLAW
```

The SET IPADDR statement provides the default IP address if one is not specified on the DEFINE LINK/DEFINE ADAPTER statements. Also, when multi-homing is used, it is good practice to always explicitly code IP addresses on the DEFINE LINK/DEFINE ADAPTER statements.

5

Configuring the Telnet Daemon

Overview

One of the most useful features of TCP/IP FOR VSE is that it supports the TN3270 and TN3270E protocols. This means that any user with a TN3270 client can directly access your 3270-based applications running under VSE. You can use the following TN3270 daemons with TCP/IP FOR VSE:

- A TN3270 daemon that uses your existing Virtual Telecommunications Access Method (VTAM) configuration to route 3270 data streams through VTAM. This daemon requires VTAM. The installation selects the logical unit (LU) name that the TN3270 client is to use based on IP addresses or a port-mapping methodology.
- A TN3270E daemon that uses your existing VTAM configuration to route 3270 data streams through VTAM. This daemon also uses VTAM, but the TN3270E client selects the LU name.

Before you can use TN3270 support, you must do the following:

- Define a VTAM name for each logical session to be supported. Note that these names define virtual terminals. Virtual terminals are terminals that do not physically exist, but they are defined to VTAM and recognized by VTAM. If you are using TN3270E, all possible LU names that might be selected by any client must be defined to VTAM.
- Optionally, create a logon menu panel if more than one application is to be accessible.
- Define the telnet daemons.

Configuration Commands

See the *TCP/IP FOR VSE Command Reference* for information about the commands you can use to configure telnet daemons. These commands are listed in the table below.

Command	Task
DEFINE TELNETD	Define a telnet daemon
DELETE TELNETD	Delete a telnet daemon
QUERY TELNETDS	Query telnet daemons and associated status
SET TELNETD_BUFFERS	Define the number of concurrent buffers for telnet
CONNECT_SEQUENCE	Controls how telnet daemons are assigned to session requests based on IP address
DEFINE MENU	Define a USSMSG10-type menu
DELETE MENU	Delete a USSMSG10-type menu
QUERY MENUS	Query your menus

Restrictions

TCP/IP FOR VSE does not support the following:

- TN3270E printing as defined by RFC 1147. The enhanced printing capabilities in TN3270E are available through the General Print Server (GPS) optional feature. The TN3270 protocol is non-SNA, which means that the ATTN and SYSREQ keys do not work.
- Line-mode telnet daemons, primarily because there are no VSE applications that are line oriented. This means that you *must* have a TN3270 client if you plan to use the TN3270 daemon provided by TCP/IP FOR VSE.

Features

The TCP/IP FOR VSE TN3270 daemon provides many features:

- Support for multiple screen sizes
- Extended highlighting with high intensity, underscored, and blinking fields
- Support for program symbol sets
- Extended color support. Each character can be a different color, regardless of field definition
- Cursor-selectable fields. Some TN3270 clients support this feature with the mouse, giving you point-and-click capability.

Required Resources

System resources are required to support TN3270 sessions. See chapter 12, “Performance,” for more information about the resources required to run TN3270 daemons.

VTAM Setup

Before you start a telnet daemon, you must define a VTAM application ID to be used as a virtual terminal LU name. One ID (LU name) is required for each telnet TN3270 daemon, whether in session or not. Each application name is referenced by a single corresponding TCP/IP FOR VSE DEFINE TELNETD command.

VTAM Definitions

For your convenience, we provide the library member TCPAPPL.B. This member defines virtual terminals that you can use in one or more DEFINE TELNETD statements.

Once the library member is complete, you can either add it to the VTAM startup member ATCCON*xx* or use the VTAM VARY command after VTAM initialization is complete. Issue this command as follows:

```
V NET,ACT,ID=TCPAPPL
```

Example The following screen is a sample VTAM definition:

```
TCPAPPL  VBUILD  TYPE=APPL
TELNLU01 APPL    AUTH=(ACQ),EAS=1
TELNLU02 APPL    AUTH=(ACQ),EAS=1
TELNLU03 APPL    AUTH=(ACQ),EAS=1
TELNLU04 APPL    AUTH=(ACQ),EAS=1
```

In this example, note the following:

- TELNLU01 through TELNLU04 define four virtual terminals. Each virtual terminal is used by a telnet daemon.
- You must add APPL statements for any additional virtual terminals you need.
- The application names are arbitrary. If you are defining numerous telnet daemons, we recommend that you use virtual terminal names with a prefix and numeric suffix, such as TELNLU*xx*, where *xx* is the numeric suffix. The prefix can be from one to six characters, and the suffix can range from 01 to 99. When you specify the application names (virtual terminal names) this way, it enables you to use the COUNT= parameter on the DEFINE TELNETD command. With the COUNT= parameter, you can define up to 99 telnet daemons with a single DEFINE TELNETD command.
- If you are using TN3270E, the TN3270E client may select LU names. You MUST have VTAM definitions corresponding to those LU names or the TN3270E client cannot log on to VTAM applications.

VTAM Considerations

As you define your telnet daemons, you should consider the following VTAM characteristics:

- Each logical session requires a VTAM application ID. The ID appears as an LU name to the target application and must be acceptable to it. There is no difference in the way that it talks to the virtual terminal and the physical 3270 terminal.
- Each telnet daemon creates and opens a VTAM ACB when it is initialized. This implies that VTAM definitions must be active before you execute the DEFINE TELNETD command that creates the virtual terminal. We recommend that you copy sample member TCPAPPL.B from PRD2.TCPIP to PRD2.CONFIG and tailor the member in PRD2.CONFIG. Once you have defined all of your virtual terminals, you can place TCPAPPL in the ATCCON*xx* member of PRD2.CONFIG (or whatever sublibrary you use for VTAM customization).
- When VTAM runs under VSE/ESA Version 2.1 or higher, it requires one megabyte of dataspace storage for initialization and one additional megabyte of dataspace storage for each partition that connects with VTAM. TCP/IP FOR VSE, therefore, increases by one megabyte the amount of dataspace storage that VTAM requires. To increase VTAM's storage, modify the DSPACE parameter on the EXEC card in the VTAM startup procedure. Depending on your system dataspace definitions (which you can display with the QUERY DSPACE console command), you may need to increase the system dataspace size. You can use the SYSDEF DSPACE command to do this.
- If you run multiple TCP/IP partitions, VTAM requires one additional megabyte for each partition that runs telnet daemons. See the IBM publication *VSE Program Directory for VSE/ESA* for a discussion of VTAM dataspace requirements.
- When VTAM runs under VSE/ESA Version 2.1 or higher, you must specify a DSPACE value on the EXEC IPNET JCL statement that starts the TCP/IP FOR VSE partition. We recommend a default value of DSPACE=2M. If you receive messages indicating that VTAM has a dataspace buffer shortage, you can increase the DSPACE value by one megabyte.

Buffer Pools

Each telnet daemon can use either a dedicated buffer or pooled buffers. In dedicated mode, a daemon obtains a 16K buffer when a session is requested. This buffer is used for all I/O operations for the duration of the session. Dedicated buffer pools are more efficient than shared pools.

In pool mode, a 16K buffer is obtained from a buffer pool only when it is needed for I/O. Because of how TCP/IP FOR VSE is designed, input from the terminal is already completely in memory (in other storage) before the I/O buffer is needed.

For outbound traffic, a buffer is obtained when VTAM indicates that data is waiting. The buffer is retained for the length of time required to obtain the data from VTAM and to pass it to the socket interface. In this way, one buffer can serve many daemons.

Base your decision about pool mode use on whether you have enough storage in the TCP/IP FOR VSE partition to support dedicated buffer pools. The buffer pools are all allocated in 31-bit storage.

Defining a Telnet Daemon

Each telnet session requires the definition of one daemon. As soon as it is defined, the daemon uses the specified APPL name (terminal ID) to connect with VTAM. Connection with a target application such as CICS does not occur until a session is actually requested by a user.

TN3270E implements two types of telnet daemons. The *listener* daemon listens on a specified TCP/IP port (for example, PORT 23). When it receives an incoming request, it passes the request to an eligible *effector* daemon. The effector daemon is responsible for managing the session between the TN3270E client and the VTAM application. You must have one listener daemon for each TCP/IP port that you want to use for incoming telnet traffic and you must have one effector daemon for each concurrent session.

TN3270E requires the coordination of resource definitions. The TN3270E client selects the LU name but the installation must still have a virtual terminal of that name defined to VTAM and a TN3270E effecter daemon within TCP/IP FOR VSE that specifies that LU name using the TERMNAME parameter. Many VSE sites use the terminal name for security and it is important to note that while TCP/IP FOR VSE provides client-specified LU names, the specification of the LU name by the client can still be rigorously controlled by the installation.

LOGMODEs

When client software running on a remote platform requests a TN3270 session, the client indicates the 3270 model number (2, 3, 4, or 5) that it wants to emulate. The daemon then selects an appropriate LOGMODE entry from those specified in its definition. TCP/IP FOR VSE defaults to the following LOGMODEs:

Terminal Type	LOGMODE Name
3278-2	S3270
3279-3	D4B32783
3278-4	D4B32784
3278-5	D4B32785

These LOGMODEs are shipped with VSE/ESA in the default member ISTINCLM.Z in PRD1.BASE. Some VSE/ESA installations use the LOGMODE table that VSE/ESA supplies in member IESINCLM in ICCF library 59. If you use the ICCF member, you must specify appropriate values for the LOGMODE3, LOGMODE4, and LOGMODE5 parameters for each telnet daemon that you define.

Note: All LOGMODEs used by TN3270 must be defined as non-SNA.

CICS

Regardless of the connection method, you must define each LU name (that is, the telnet VTAM application ID) to each CICS you want to use. To do this, you can either use CICS Auto Install or code a TCT entry for each LU name.

Auto Install

If you use the Auto Install feature of CICS, each LU is defined when a connection is attempted. The terminal's characteristics are determined from the bind information passed by VTAM. Because bind information is obtained from the LOGMODE used by the telnet daemon, be sure that you are using the correct ones. Specifically, make sure that each LOGMODE entry is appropriate for the 3270 model being emulated and that the LOGMODE specifies a non-SNA local device. If you are having difficulties connecting to CICS, one place to look for information about why CICS might be rejecting the bind is in the *Inspect Message Log* option in the VSE Interactive User Interface (IUI). This display shows the netname and model name of the terminal that CICS is attempting to Auto Install.

To view the characteristics of the model name, you need to use the CEDA facility in CICS and locate the GROUP where your CICS terminal types are defined. In a typical VSE/ESA system, the default group is VSETYPE, and the CICS command you use to view all of the CICS terminal types would be CEDA EXPAND GR(VSETYPE).

TCT

If you are not using Auto Install, you must code a TCT entry or make an RDO definition for each LU name to be supported. Be sure that TRMTYPE=3270 is coded (non-SNA).

Controlling the IP Address

When a TN3270 client (for example, a terminal user) requests a telnet session and does not specify an LU name, the first available daemon is normally assigned. This may be undesirable for several reasons. You may want to enforce certain session properties based on the originating IP address, or you may want to restrict the number of sessions permitted with some applications. In addition, you may have a security policy that is based on CICS terminal identifiers or VTAM NETNAMEs. You can associate a VTAM terminal name with an IP address in one of the following ways:

- By creating daemon pools
- By specifying address patterns
- By allowing the client to specify the VTAM terminal name using TN3270E

Creating Daemon Pools

You can use port numbers to create separate pools of telnet daemons. By convention, telnet requests are sent to port 23. There is, however, no requirement for this, and most telnet client software permits you to specify the target port number. This approach allows you to establish different daemon pools with different characteristics. The drawback is that the end user must know which port to select.

Specifying Address Patterns

You can specify an IP address pattern for each daemon. In this case, the daemon establishes a session with an IP address only if it matches the pattern. The CONNECT_SEQUENCE command setting determines the process used to select a telnet daemon.

IP addresses can match based on a complete specification, a match on subnet, or a match on network number. A daemon with a 0.0.0.0 specification matches any request.

CONNECT_SEQUENCE OFF

When CONNECT_SEQUENCE is set to OFF (the default), the first daemon that matches any test is assigned. The IPADDR= is completely ignored, and any client can connect without any IP address matching.

CONNECT_SEQUENCE ON

When CONNECT_SEQUENCE is set to ON, TCP/IP FOR VSE selects a telnet daemon (and corresponding terminal name) on a best-fit basis using the following algorithm:

1. TCP/IP FOR VSE attempts an exact match of the IP address of the incoming request with the IP address of each telnet daemon (IPADDR= on DEFINE TELNETD).
2. If step 1 fails, TCP/IP FOR VSE attempts to match the network and subnetwork values of the incoming request with a generic network and subnetwork address based on the IPADDR parameter of each telnet daemon.

3. If step 2 fails, TCP/IP FOR VSE attempts to match the network portion of the incoming request with a generic network number based on the IPADDR parameter of each telnet daemon.

4. If step 3 fails, TCP/IP FOR VSE matches the IP address of the incoming request with an IPADDR of 0.0.0.0 on the IPADDR of a telnet daemon. If the IPADDR is omitted on the DEFINE TELNETD command, an IPADDR of 0.0.0.0 is assumed.

Note: The SET MASK command has no effect on the selection process.

Example

For example, assume the following partial definitions exist with CONNECT_SEQUENCE ON:

```
DEFINE TELNETD,ID=AAAA,COUNT=3,  IP=10.108.34.10
DEFINE TELNETD,ID=BBBB,COUNT=40, IP=10.32.0.0
DEFINE TELNETD,ID=CCCC,COUNT=100,IP=10.0.0.0
DEFINE TELNETD,ID=DDDD,COUNT=200,IP=0.0.0.0
```

Incoming client requests would be serviced in the following sequence:

- Clients with IPADDR 10.108.034.10 would connect to ID=AAAA.
- Clients with IPADDR 10.32.*nnn.nnn* would connect to ID=BBBB.
- Clients with IPADDR 10.*nnn.nnn.nnn* would connect to ID=CCCC, other than the 10.108.034.101 and 10.32.*nnn.nnn* addresses that are serviced by ID=AAAA and ID=BBBB.

TCP/IP FOR VSE also considers the total count per daemon:

- If the number of clients exceeds the ID=AAAA count (3), then the client with IPADDR 10.108.034.010 would connect into ID=BBBB.
- If the number of clients exceeds the ID=BBBB count (40), then clients with IPADDR=10.32.*.* would connect into ID=CCCC.
- If the number of clients exceeds the ID=CCCC count (100), then clients would connect into ID=DDDD.

If there were no ID=DDDD defined that allowed any clients to connect to it—it has the default setting of IPADDR=0.0.0.0—the client connection requests that did not match the ID=AAAA, ID=BBBB, or ID=CCCC IPADDR= settings would be rejected.

Recommendations

Specify CONNECT_SEQUENCE ON in the following situations:

- To guarantee that an incoming telnet request is always associated with a given telnet daemon and its associated terminal name.

- To limit the number of telnet daemons available to TCP/IP hosts on a given network and subnetwork.

CONNECT_SEQUENCE ON may result in slightly higher CPU utilization during telnet connection requests.

Note that even after a connection is established, the IP address, port number, user ID, and password of the person trying to log in is passed to the security exit. A rejection could still occur at that point.

Using TN3270E

When the TN3270E client attempts to connect to TCP/IP FOR VSE, it must specify a port (normally port 23) that a TN3270 listener daemon is listening on. The IP address of the TN3270E client must be able to connect to the TN3270E listener, meaning that it cannot be excluded by the IPADDR parameter of the TN3270E listener daemon.

To confirm that your listener daemon is listening on port 23, and to see whether it places restrictions on the IP addresses that can connect to it, you can issue the TCP/IP operator command QUERY TELNETDS. When the connection is established, the TN3270E listener daemon attempts to locate a TN3270E effecter daemon (DEFINE TELNETD,TN3270=E) that is eligible to process the request. The TN3270E effecter daemon is deemed eligible if its TERMNAME matches the terminal name specified by the incoming TN3270E session.

Telnet Menu

Each daemon can connect the end user directly to an application or can present the user with a menu. Menus provide additional flexibility, including the ability to require a user ID and password. The most common TN3270 target application under VSE is CICS.

TCP/IP FOR VSE provides functionality similar to that available with VTAM's USS table and the USSMSG10 display. When you provide a TCP/IP FOR VSE telnet menu, inbound telnet users can choose available applications with a simple command or PF key. The menu facility in TCP/IP FOR VSE provides the following features:

- Easy application selection
- Network solicitation screen
- Message of the day
- Security. This feature permits you to request a user ID and password before you allow access to any applications. This security check supplements the user ID and password that is required to access a specific application.

How to Code

A TCP/IP FOR VSE telnet menu consists of a set of 80-byte records stored as a library member. Each table provides a definition of one 24 × 80 screen image. Each definition includes a list of hot keys and the actions to be taken.

To define telnet menus to TCP/IP FOR VSE, use the DEFINE MENU command. Once defined, the menu can be included in the definition of a telnet daemon. To change a menu, you must (1) make the change in the source member, (2) issue the DELETE MENU command, and then (3) reissue the DEFINE MENU command. You do not have to cycle the telnet daemons. Users see the new menu automatically the next time they connect to the daemon.

Menu Definition Sections

Each telnet menu definition member consists of the following sections:

1. The first, the header section, contains a series of keyword definitions. These definitions include define-action keys and placeholders used in the second section.
2. The second section consists of a 24 × 80 screen image. You must have EXACTLY 24 lines in the second section or the menu definition will fail.

Header Section

The header section of a screen definition consists of a series of keyword definitions. Each definition begins in column one of a new record. Keywords must be in upper case. Blank lines are ignored in this section. Lines beginning with an asterisk (*) are assumed to be comments and are ignored. The last statement in the header section must be IMAGE.

Keyword	**Description**
IMAGE	This keyword indicates the end of the header section. Exactly 24 records must follow the IMAGE keyword. These records are used to generate the screen image.
HI=*char*	The specified character is replaced with a high-intensity attribute in the screen image section.
LO=*char*	The specified character is replaced with a normal-intensity attribute in the screen image section.
VAR=*char*	This character identifies certain predefined dynamic values that can be displayed in the screen section. The values are described in “Variables” on page 83.
INPUT=*char*	This character identifies certain predefined dynamic values that can be displayed and modified in the screen section. The values are described in “Variables” on page 83.
CMDLINE=*char*	This character identifies the start of a command line. You must terminate the command line with another attribute character. The command line may be any length; however, only the first 80 characters are examined during an input operation. Be sure that the command line area does not overlap the area reserved for error messages.
MSGLINE=*n*	This keyword sets the first screen line (1 to 23) of a two-line area where errors are displayed. The default is line 23. This area must not overlap the command line.
CHAR=*c*=*string*	This statement may be coded as many times as required to define a series of *command letters*. Valid values for *c* are A–Z and 0–9. When the user enters the command character, the value of *string* is executed as a command.
PF*nn*=*string*	This statement may be coded as many times as required to equate a series of command strings to PF keys.

Keyword	Description
PA*n*=*string*	This statement may be coded as many times as required to equate a series of command strings to PA keys.
CLEAR=*string*	This statement assigns a command string to the CLEAR key. In practice, you probably want to assign either the REFRESH or EXIT command.
TRIES=*n*	If user IDs and passwords are checked, *n* is the number of failures permitted before the user is disconnected. The value of *n* can range from 1 to 255.

Entering Screen Commands

A user can enter commands in the following three ways:

- By entering them explicitly in the menu command line, assuming that a command line is available.
- By entering a single character in the menu command line, if the character has been equated to a command string.
- By pressing a PF or PA key that has a command string equated to it.

Command strings consist of one or more commands. Commands are separated from each other by one or more blanks. Command operands are enclosed in parentheses and immediately follow the command, without any intervening blanks. Commands are not case sensitive.

The following table lists commands and their descriptions. Only the capitalized portion of a command needs to be entered, although the entire entered string is checked for correctness.

Command	Description
LOGON	Attempts to complete a logon operation. Its execution is always deferred until all other commands in the current command string are processed.

APPLication(*name*)
APPLid(*name*)
	Sets the name of the application to which logon is attempted.

USERid(*name*)
ID(*name*)
	Sets the user ID to the indicated value. This value is validated when the LOGON command is processed.

Command	Description
PASSword(*string*) PSWD(*string*)	Sets the password to the indicated value. This value is validated when the LOGON command is processed.
REFRESH	Causes a redisplay of the screen. This command is normally assigned to the CLEAR key.
DATA	Allows you to pass DATA into the VTAM application. This is analogous to using the DATA keyword on the VTAM LOGON command. The VTAM application must support the use of the DATA parameter.
EXIT, QUIT, or LOGOFF	Terminates the telnet session.

Variables

Variables are keyword values that you can include in a screen definition to display information and/or permit the user to enter replacement values. Whether or not a field is available for change depends upon the escape character that precedes the keyword.

Variables used for display do not include 3270 attribute characters. Those specified as input fields have lead and terminating 3270 attributes automatically generated and the values appear as high intensity (except for the PASSWORD variable, which is not displayed).

There must be sufficient room on a line to completely display the variable and any generated attributes. Variables may not span lines.

Each variable name may be used only once as an output field and once as an input field in a single screen. The following table shows supported variables.

Variable	Input?	Required Characters	Contents
USERID	YES	16	The user ID to be associated with this session
PASSWORD	YES	16	The password associated with the user ID
IPADDR	NO	15	The network address of this session
APPL	YES	8	The application name scheduled for use by the logon function

Variable	**Input?**	**Required Characters**	**Contents**
LOGMODE	NO	8	The LOGMODE to be used by the LOGON function
LUNAME	NO	8	The VTAM logical unit name to be used for this session

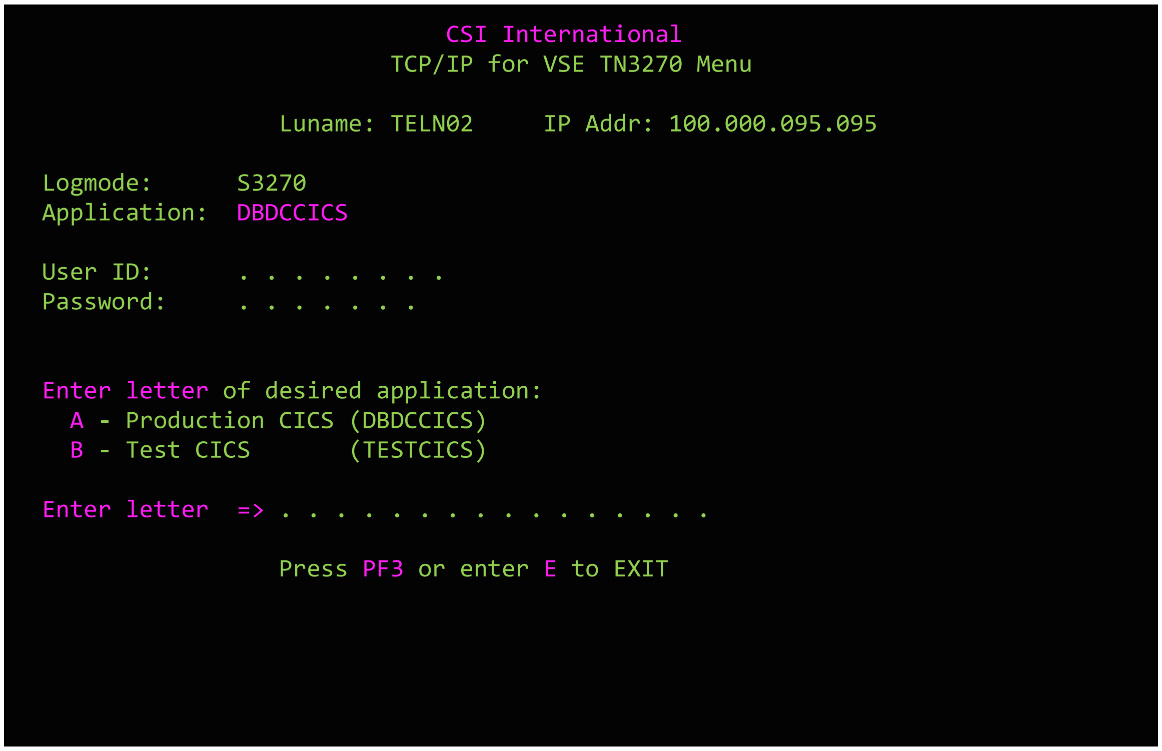
A failed logon attempt, such as one caused by an invalid or inactive application, terminates the telnet session. The failed logon does not, however, produce an error message.

Single-Character-Input Menu Example

The following definition file example shows how single-character selection of allowable values is implemented.

```
HI=#
LO=$
VAR=@
CMDLINE=?
INPUT=+
PF3=EXIT
PF10=LOGON
CLEAR=REFRESH

CHAR=A=LOGON APPL(DBDCCICS)
CHAR=B=LOGON APPL(TESTCICS)

CHAR=E=EXIT

MSGLINE=23
TRIES=3
IMAGE
                               #CSI International$
                            TCP/IP for VSE TN3270 Menu

                    Luname:@LUNAME     IP Addr:@IPADDR

  Logmode:     @LOGMODE
  Application: @APPL

  User ID:     +USERID
  Password:    +PASSWORD

 #Enter letter$of desired application:
   #A$- Production CICS (DBDCCICS)
   #B$- Test CICS       (TESTCICS)

 #Enter letter =>?                                    $

                    $Press#PF3$or enter#E$ to EXIT
```

The resulting menu screen appears below. The colors in the screen may vary depending on the TN3270 client configuration. In this example, the magenta color represents highlighted text. The pale green lettering represents normal text.



```
                          CSI International
                      TCP/IP for VSE TN3270 Menu

                Luname: TELN02     IP Addr: 100.000.095.095

Logmode:      S3270
Application:  DBDCCICS

User ID:      . . . . . . . . 
Password:     . . . . . . . 


Enter letter of desired application:
  A - Production CICS (DBDCCICS)
  B - Test CICS       (TESTCICS)

Enter letter  => . . . . . . . . . . . . . . . . 

                Press PF3 or enter E to EXIT
```

In the menu above, the user sees the initial screen. The initial LOGMODE and application are taken from the telnet daemon definition. The user is expected to fill in both the user ID and password information. Note that the password field is generated automatically as a non-display field.

In this example, the user could enter the APPL(*name*) directive to select other applications not represented by single characters. For instance, assume that the user enters the following command at the command prompt (=>):

```
LOGON APPLID(MVSTSO)
```

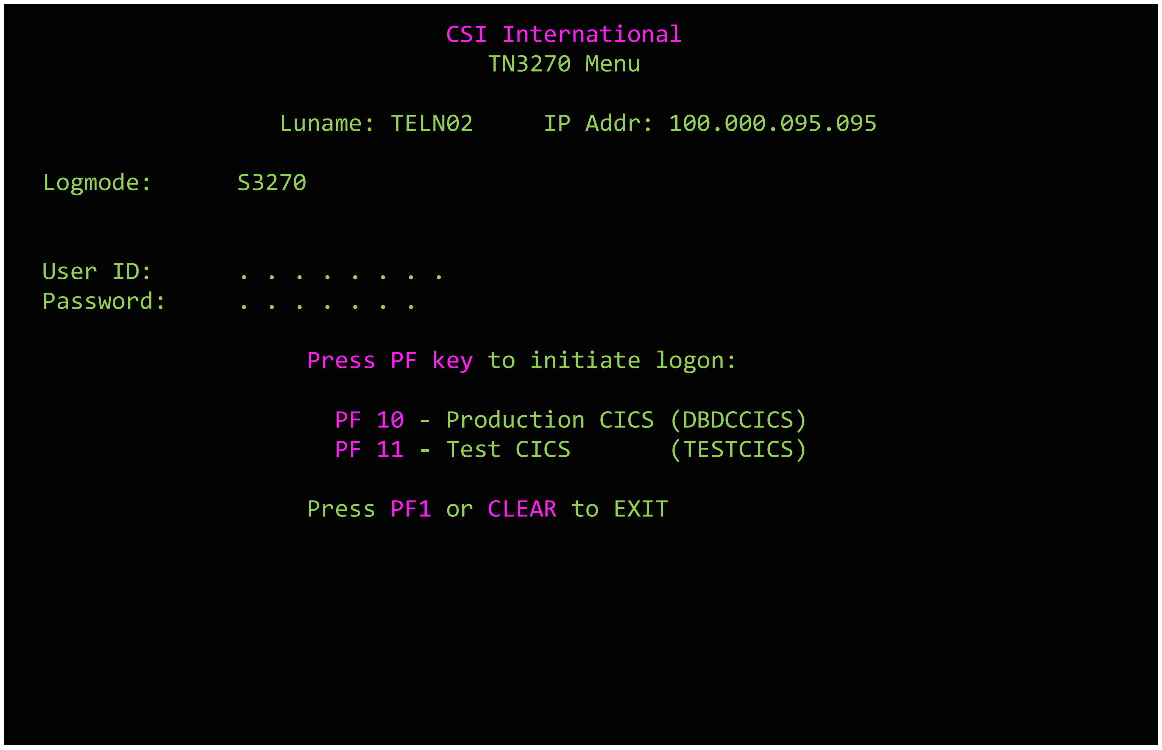
TCP/IP FOR VSE attempts to log the user on to the VTAM application MVSTSO.

PF-Key-Input Menu Example

This definition file example is similar to the first one, but it shows the use of PF keys.

```
HI=#
LO=$
VAR=@
CMDLINE=?
INPUT=+
PF1=EXIT
CLEAR=EXIT

PF10=LOGON APPL(DBDCCICS)
PF11=LOGON APPL(TESTCICS)

PF1=EXIT

MSGLINE=23
TRIES=3
IMAGE
                               #CSI International$
                                   TN3270 Menu

                    Luname:@LUNAME     IP Addr:@IPADDR

  Logmode:     @LOGMODE


  User ID:     +USERID
  Password:    +PASSWORD

                    #Press PF key$to initiate logon:

                      #PF 10$- Production CICS (DBDCCICS)
                      #PF 11$- Test CICS       (TESTCICS)

                    $Press#PF1$or#CLEAR$to EXIT
```

The resulting menu appears below.



```
                                CSI International
                                   TN3270 Menu

                    Luname: TELN02     IP Addr: 100.000.095.095

 Logmode:      S3270


 User ID:      . . . . . . . .
 Password:     . . . . . . .

                      Press PF key to initiate logon:

                        PF 10 - Production CICS (DBDCCICS)
                        PF 11 - Test CICS       (TESTCICS)

                      Press PF1 or CLEAR to EXIT
```

Using PF keys reduces the required keystrokes. Also, note that the lack of a command line prevents the user from selecting unanticipated values.

Library PRD2.TCPIP contains the sample CSIMENU1, which you can tailor to meet your requirements.

6

Configuring FTP Daemons

Overview

The File Transfer Protocol (FTP) enables you to transfer files between VSE and remote systems such as UNIX, z/OS, VM, Microsoft Windows®, AS/400, Linux, and even other VSE systems.

The VSE files you can transfer include

- VSAM (ESDS and KSDS) files
- Sequential disk files
- Tape files
- Librarian members
- VSE/POWER RDR/PUN/LST queue entries
- Hierarchical File System (HFS) files.

For an overall understanding of how the FTP protocol works, see chapter 2, “FTP,” in the *TCP/IP FOR VSE User Guide*.

Planning Considerations

Performing file transfers is simple, but the first thing that needs to be considered is how the FTP transfer is initiated and controlled. If you want to start and control the FTP file transfer from a remote system, then you must first configure a FTP Server (daemon) on the VSE system. Information on configuring and running a FTP server on VSE can be found in the next section, "Introduction to Configuring an FTP Server (Daemon)."

If you want to start and control the FTP transfer from VSE, then you can use any of the FTP clients provided with TCP/IP FOR VSE. All of these clients run on VSE. The remote system must also be enabled and running an FTP server (daemon).

If you only want to use the batch FTP client (// EXEC FTPBATCH program) to send or receive files from a remote system, then you do not need to configure or define a FTP server (daemon) on VSE because the client attaches and executes an FTP daemon automatically during the FTPBATCH job's execution. The FTP daemon attached by FTPBATCH is terminated when the FTPBATCH job step is completed. See the section "FTP as a Client on VSE" in the *TCP/IP FOR VSE User Guide* for more information on starting and controlling FTP transfers from VSE.

As you activate FTP on VSE, you need to consider the following:

- How you want to configure an FTP server (daemon) on VSE:
  - Internal FTP servers are created with the DEFINE FTPD command
  - External FTP servers are created with the // EXEC FTPBATCH program
- Which security features are required, and how security is to be handled. For more information on FTP security, see Chapter 9, "Security."
- How to control data transmission defaults using EXTTYPES.L

We examine each of these issues in the following sections from an installation point of view.

Introduction to Configuring an FTP Server (Daemon)

If you want to start and control the FTP transfer from a remote system, then you must consult the documentation of the remote system for information on the commands and interface that it provides. A common example of a foreign client would be the Microsoft® Windows DOS FTP command, and many PC and UNIX software companies provide easy-to-use graphical FTP client interfaces. The TCP/IP FOR VSE automatic and interactive FTP clients also require an FTP server to be active on VSE.

See the section "FTP as a Server on VSE" in the *TCP/IP FOR VSE User Guide* for information on FTP server commands and replies when using VSE as an FTP server.

Creating an Internal FTP Server

The DEFINE FTPD command is used to create an internal FTP Server (daemon). You must issue one DEFINE FTPD for each port that is opened by remote FTP clients. The default port number is 21.

See DEFINE FTPD in the *TCP/IP FOR VSE Command Reference* for information on the available parameters for this command. The minimum recommended parameters are shown in the following syntax statement.

```
DEFine FTPd,ID=id[,PORT=num|21][,UNIX=YES|NO|BIN]
       [,MAXACTive=n|3][,IDLETIME=ttt|0][,WELCOME=welcome.L]
       [,EXTTYPES=YES|NO][,DYNFILES=YES|NO][,ZEROERROR=YES|NO]
       [,UPPERCASE=NO|YES][,EXTRADATA=FAIL|WARN|IGNORE|ACCEPT]
       [,SITELAST=NO|YES]
```

These parameters are described in the following table:

Parameter	Description
DYNFILES= [YES | NO]	The default (YES) allows users to specify dynamic (autonomous) files, which are files that are not defined to the TCP/IP FOR VSE file system. This means that remote FTP users can transfer a local VSE file directly by specifying its DLBL/TLBL and bypass the TCP/IP FOR VSE file system. For tighter security, you may want to specify DYNFILE=NO on this server (port) to restrict access to only files defined by the DEFINE FILE command. See the *TCP/IP FOR VSE User Guide* for details on autonomous files. **Note**: The LOCAL_DLBL OFF command overrides the default DYNFILE=YES and provides the same restriction for all internal servers. It is run from the global TCP/IP initialization deck. This command is supported for compatibility with earlier releases.

Parameter	**Description**
EXTRADATA= [FAIL | WARN | IGNORE | ACCEPT]	This option controls what is done when an incoming text file has extra data at the end. “Extra data” (in a text file only) is defined as a text string that is not correctly ended with a CR/LF or other valid delimiter. The values for this parameter are as follows: FAIL (the default): The transfer fails and is not stored on VSE. An FTP343W warning message is generated, and a 5*xx* failure code is sent to the client. WARN: A warning message is generated, but the file is stored on VSE. A normal code sent to the client. The undelimited data is discarded. IGNORE: No messages are generated. The undelimited data is discarded. ACCEPT: No messages are generated. The undelimited data is accepted and stored as if it were correctly delimited.
EXTTYPES= [YES | NO]	Specifying NO suppresses using the EXTTYPES.L overrides when transferring data. The default is YES. See also SITELAST= below.
ID=*id*	A 1- to 16-character alphanumeric name to identify this daemon.
IDLETIME= [*ttt* | 0]	Specifies a time-out interval for FTP sessions that are open but where no command has been received. The IDLETIME automatically terminates a session that is left idle for more than the time interval determined by the variable *ttt*. This variable is specified in 300th-second units. For example, setting IDLETIME to 18000 would terminate an inactive FTP session after one minute. The default is zero, which means a remote user who establishes an FTP session and walks away from the PC would leave the session open indefinitely.
MAXACTIVE= [*n* | 3]	This parameter establishes the number of concurrent user sessions supported by the daemon. Once this limit is reached, no new sessions are allowed to be established until other sessions in use terminate. Valid values of *n* range from 1 to 65535. The default value is 3.

Parameter	Description
PORT= [*num* | <u>21</u>]	The TCP/IP port number for this FTP daemon to monitor. The default is 21, but you might want an FTP daemon listening to a different port. For example, assume that you have an FTP daemon listening at port 3021. If the FTP daemon on port 21 has reached the maximum number of active sessions, and you need to transfer a file, you can point your FTP client to port 3021 and establish an FTP connection.
SITELAST= [<u>NO</u> | YES]	Setting this option to YES allows SITE commands to override DEFINE FILE parameter settings (BLKSIZE, CC, CRLF, LRECL, RECFM, TRCC). The default is NO. **Note**: This option has no effect if EXTTYPES=YES and the file type has a matching entry in EXTTYPES.L. In that case, EXTTYPES overrides are always used. See “Controlling Defaults Using EXTTYPES.L” for information on the EXTTYPES transfer values.
UNIX= [YES | BIN | <u>NO</u>]	Specify YES or BIN to start in UNIX mode: • YES to tell the FTP daemon to start each session in UNIX simulation mode. Binary transfers occur in ASCII mode. • BIN is the same as YES, but it is better because it allows binary transfers to occur in binary mode. This is useful when you use a graphical PC client that understands the simulated UNIX directory structure of TCP/IP FOR VSE, but you still want true image/binary transfers to occur. The default is NO.
UPPERCASE= [<u>NO</u> | YES]	All replies are sent in mixed case by default. Specifying YES causes all replies to be sent back to the foreign FTP client in upper case. This may be needed because of translation issues.

Parameter	Description

WELCOME= *member*	The VSE member containing a message that is sent at the start of each new session. This text is added to the 220 system message a remote client receives when it connects into your VSE system. You can add text to identify your company and display a help message. The *member*.L must be catalogued in the LIBDEF chain as part of the TCP/IP FOR VSE initialization. Any text in columns 73 through 80 is ignored.
	
	The following job shows how to catalog a message member, in this case GREETING.L:
	
	`// EXEC LIBR`
	`ACC SUB=lib.sublib`
	`CATALOG GREETING.L REPLACE=YES`
	`*********************************************`
	`WELCOME TO ABC CORPORATION’S VSE SYSTEM.`
	`EMAIL XYZ@ABC.COM FOR HELP ON THIS FTP SERVER`
	`*********************************************`
	`/+`
	`/*`
	
	The example below shows how the message in GREETING would appear to a remote FTP client.
	
	Note: Using FTPD=NO on the ASECURITY command blocks these messages and prevents new FTP sessions. For details, see chapter 9, “Security,” on page 133.
	
	`220-TCP/IP for VSE Internal FTPDAEMN 02.01.07 20160310 02.34`
	`  Copyright (c) 1995,20xx Connectivity Systems Incorporated`
	`*********************************************`
	`WELCOME TO ABC CORPORATION’S VSE SYSTEM.`
	`EMAIL XYZ@ABC.COM FOR HELP ON THIS FTP SERVER`
	`*********************************************`
	`220 Ready for new user`

ZEROERROR= [YES | NO]	Attempting to transfer an empty (null) file is considered an error by default, and a 500-level error message (a fatal error) is generated. Specify NO to allow transferring an empty file.

Monitoring and Controlling an Internal FTP Server

See the *TCP/IP FOR VSE Command Reference* for details on the following commands that allow you to monitor and control an internal FTP server (daemon):

- DEFINE FTPD
- DELETE FTPD
- QUERY FTPD
- QUERY ACTIVE TYPE=FTP

Creating an External FTP Server

If you want to provide FTP services using a partition external to the TCP/IP stack, FTPBATCH can be executed as a full-function, multiple-session server. Using FTPBATCH this way achieves the following advantages:

- File open and file close operations are moved to the external partition. All I/O is accomplished outside of TCP/IP.
- The main TCP/IP stack partition is freed to focus on network activity.
- Recoverability is improved. The external FTP servers can be terminated and restarted without affecting the main stack partition.
- Workloads are prioritized effectively. Multiple CPUs can be exploited using the VSE turbo dispatcher, and the VSE PRTY (priority) and other VSE commands can be used to control and monitor FTP transfers.

Use the following JCL as guide to running an FTPBATCH server.

```
// JOB FTPB0021
// OPTION LOG,PARTDUMP
// OPTION SYSPARM='00'
The Assign's, Dlbls, and Extent's for all defined files
should be taken from TCP/IP startup job
// EXEC FTPBATCH,SIZE=FTPBATCH,PARM='FTPDPORT=21'
SET IDLETIME 36000
/*
/&
```

This example shows a basic configuration. The parameters and SET commands you can use with FTPBATCH to configure a server are described below.

Parameters

The following keyword parameters apply when configuring an FTPBATCH server.

Parameter	Description
ABORT=[YES|NO]	Specifies whether the ABORT command is allowed to prematurely terminate an active transfer. The default is NO.
CASE=UPPER	Specifies that all messages issued to CONSOLE and SYSLST are translated to upper case. (UPPER is the only valid value.)
COUNT=	Synonym for MAXACT=
DATAPORT=*num*	Data port number to be used for the local VSE data connection. Applies only when VSE is in the passive mode.
DEBUG=[ON|OFF]	Sends debugging messages to SYSLST. The default is OFF.
DUMP=[YES|NO]	Specify YES to create a complete dump if the FTPBATCH program abends. Use this parameter when directed by CSI Technical Support. The default is NO.
DYNFILE=[YES|NO]	The default (YES) allows users to specify dynamic (autonomous) files, which are files that are not defined to the TCP/IP FOR VSE file system. This means that remote FTP users can transfer a local VSE file directly by specifying its DLBL/TLBL and bypass the TCP/IP FOR VSE file system. For tighter security, you may want to specify DYNFILE=NO on this server (port) to restrict access to only files defined by the DEFINE FILE command. See the *TCP/IP FOR VSE User Guide* for more information on autonomous files.
FTPDPORT=*nnnn*	Required to create an FTPBATCH server. This causes the FTPBATCH partition to stay up and wait for connection requests from foreign FTP clients. **Note**: You can block autonomous file transfers on this server (port) by using the DYNFILE=NO parameter.

Parameter	**Description**
ID=*sysid*	Specifies the system ID of the TCP/IP FOR VSE partition that is to serve as the local host or client. Remember that you can have more than one copy of TCP/IP FOR VSE running at one time (such as production and test). The default is 00. If your installation uses a TCP/IP FOR VSE ID other than 00, you must specify the *sysid* value in any FTP batch job that you run. Alternatively, you can specify the ID parameter in a `// OPTION SYSPARM='`*`sysid`*`'` job control statement.
LIP=*nnn.nnn.nnn.nnn*	Allows a different local IP address to be used by FTPBATCH for use in VSE systems with multiple local IP addresses.
MAXACT=*nn*	Specifies the maximum number of FTP sessions that can connect to FTPBATCH in server mode. This number ranges from 1 to 28. The default is 16, which normally is adequate to ensure that sessions can connect. Although you can permit up to 28 concurrent sessions, each session, as it becomes active, requires additional system resources. You must plan the size of the FTPBATCH server partition accordingly.
SSL=SERVER	Enables SSL (secure encryption) for the server. Requires activating the SecureFTP optional feature. The Internet Engineering Task Force (IETF) has renamed SSL to TLS, and TLS= should be used instead of SSL=. See TLS= for more details. SSL= is still accepted to support older jobs. SSL= and TLS= are synonyms.
SYSLOG=ON	Specifies that all messages are directed to the console and to SYSLST. (ON is the only valid value.)
TAG=[YES|NO]	Specify YES to add the task ID and timestamp to all SYSLST output. The default is NO.

Parameter	**Description**
TLS=SERVER	Synonym for SSL=SERVER. **Note**: Specifying TLS= instead of SSL= does not affect the protocol version that is used. The SET TLS*xx* command sets the minimum protocol version to be allowed by a server. For more information, see "SecureFTP for VSE" in the *TCP/IP FOR VSE Optional Features Guide*.
TRAN=*translate-table*	Specifies a translate table name for ASCII-to-EBCDIC translation on the data connection. This setting does not override the TRANSLATE parameter on the DEFINE FILE command. **Note 1**: Setting this parameter causes SITE TRANSLATE commands to fail with a "505 Translate value cannot be overridden" error message. **Note 2**: Use SET TELNTRAN to override the translate table setting on the control connection.
UNIX=[YES|BIN|NO]	Controls operation in UNIX simulation mode. Specify YES to tell the FTP daemon to start each session in UNIX simulation mode. Binary transfers occur in ASCII mode. The default is NO. BIN is the same as YES, but it is better because it allows binary transfers to occur in binary mode. This is useful when you want to use the simulated UNIX directory structure of TCP/IP FOR VSE while allowing true image/binary transfers to occur. Forward slashes may be used to separate directory and file names in UNIX mode. For more information on this mode, see chapter 1, "Fundamentals of TCP/IP."
WELCOME=*member*	Name of a VSE library ('.L') member containing site-specific text that is added to the initial 220 message issued by FTPBATCH. See WELCOME= on page 94.
ZEROERR=[YES|NO]	Specifies whether a GET of an empty file should be considered an error (YES) or ignored (NO). The default is NO.

SET Commands

The following SET commands apply when configuring FTPBATCH as a server.

SET Command	**Description**
SET BUFFCNT	Synonym for SET BUFFMAX.
SET BUFFMAX [*nn*|<u>4</u>]	This command is no longer used. If it is specified, it is ignored.
SET BUFFSIZE [*nnnnnn*|<u>65536</u>]	Sets the buffer size to use when sending data buffers to the TCP/IP partition. The default and minimum setting is 65536. The maximum is 131072. For compatibility with older releases, smaller values are allowed and not flagged as an error. They are ignored, however, and 65536 is always the smallest value used.
SET CHAPCNT [*nn*|<u>32</u>]	Used when running FTPBATCH as an external FTP server. Each VSE subtask handles an FTP session, and the VSE CHAP command controls when a session is forced to the lowest priority to allow other sessions to have equal priority. In general, higher values of *nn* lead to fewer changes in priorities (CHAPs). The default is 32.
SET CONSOLE *operand*	Manages how messages are displayed on the VSE system console. The *operand* specifies whether a message type is sent to the console or suppressed. Only one operand from the following list is allowed per command. Each operand's value defaults to the default setting of the TCP/IP partition.

Operand	**Message Type**
ALL | NONE	All messages
WARN | NOWARN	Warning
INFO | NOINFO	Informational
DIAG | NODIAG	Diagnostic
RESP | NORESP	Response
SECURE | NOSECURE	Security
UPPER	All messages are sent in upper case.

SET Command	Description
SET DATAWECB [ON|OFF]	ON causes a wait-after-send on the send buffer until it is successfully acknowledged by the foreign FTP server. The default is OFF. This command can be used to slow down FTP transfers that may be overloading a congested network.
SET DIAGNOSE [ON|NODUMP|OFF| EVENTS|FILEIO]	Controls FTP diagnostics. SET DIAGNOSE ON causes diagnostic messages and dumps to occur. Sometimes the dumps are not necessary and cause timing problems, so if you want the diagnostic messages but no dumps, you can issue a SET DIAGNOSE NODUMP after the SET DIAGNOSE ON to enable diagnostic messages with no dumps.

Value	Effect
ON	Enables diagnostics (messages and dumps) of FTP commands.
NODUMP	Suppresses dumps. Issue SET DIAGNOSE NODUMP after SET DIAGNOSE ON.
OFF	Disables diagnostics of FTP commands. (The default).
EVENTS	Enables diagnostics of significant events during the FTP processing.
FILEIO	Enables dumps during file I/O requests.

SET Command	**Description**
SET EXTRADAT [FAIL|WARN| IGNORE|ACCEPT]	This option controls how extra data at the end of a received text file is handled. "Extra data" is defined as a character string not delimited by a CR, LF, or other valid delimiter. The settings are as follows:

Value	**Effect**
FAIL	The transfer fails and is not stored on VSE. A warning message (FTP343W) is generated, and a 5*xx* failure code is sent to the client. (The default).
WARN	A warning message is generated, but the file is stored on VSE. A 2*xx* normal code is sent to the client. The undelimited data is discarded.
IGNORE	No messages are generated. The undelimited data is discarded.
ACCEPT	No messages are generated. The undelimited data is accepted and stored as if it were correctly delimited.

SET Command	**Description**
SET EXTTYPES [ON|OFF]	Enables or disables using EXTTYPE.L for file transfer overrides. The default is ON (enabled). See also "SET SITELAST" on page 103.
SET FIOWAIT [ON|OFF]	Forces file I/O to be single threaded (ON). The default is OFF.
SET IDLETIME [*nnnnnn*|0]	Sets the wait time before terminating an inactive FTP session when using FTPBATCH as a server. The value *nn* is the number of 300th-second intervals. The default is 0, which causes the server to wait indefinitely.
SET LOGCONSL	Synonym for SET CONSOLE.
SET [MSGXLEFT| MSGXRGHT]	Causes timestamps to be placed to the left or right of messages issued to SYSLST. The default position is to the right.

SET Command	**Description**
SET MSGXLOG [ON|OFF]	SET MSGXLOG ON can be used to echo SYSLST output from FTPBATCH into a sequential disk (SD) file. This can be used for applications that want to ASSGN SYSLST to a disk file for separate processing of output from FTP commands such as a directory listing. This, then, causes an open of an SD file for output, and the FTPBATCH job stream must also contain the ASSGN, DLBL, and EXTENT JCL for the MSGXLOG file. The output file is fixed in unblocked, 121-byte records to allow assigning IJSYSLS to a disk file. The first byte of each record contains the print carriage control and is followed by the 120-byte print line. The default is OFF.
SET PASVPORT *start-number tot-ports*	Sets the starting port number (>4096) and the total ports to be used for the data connection when in passive mode. The highest number in the range must less than 65536. This setting overrides the values set by the PORTRANGE command in the TCP/IP partition.
SET PULSE [ON |OFF]	Specifies whether pulsing is on or off during a GET or a PUT. The default is ON.
SET SENDFAST [ON|OFF]	Causes SENDs to be issued without waiting. When SENDFAST is set to ON and the number of unacknowledged bytes is four times the BUFFSIZE, a wait is issued to allow the foreign FTP server to acknowledge the sent data. The default is OFF. **Note:** The default behavior for sending buffers is to send a single buffer without waiting and fill a second buffer while the prior buffer is waiting for acknowledgement. If the filling of the second buffer completes before the prior buffer is completely acknowledged, a wait is issued for it to complete, and then the second buffer is sent. Alternating between these two buffers is recommended, and use of this command will probably not lead to any significant performance improvement. It could, in fact, lead to network congestion and a closed-window condition on the remote system, causing an overall decrease in performance.

SET Command	**Description**
SET SENDSNOT	Synonym for SET SENDFAST.
SET SENDWACK	Synonym for SET DATAWECB.
SET SITELAST [YES|NO]	Setting this option to YES allows SITE commands to override the following parameters on the DEFINE FILE command: BLKSIZE, CC, CRLF, LRECL, RECFM, and TRCC. The default is NO. **Note**: If SET EXTTYPES ON (the default) is used and the file type has a matching entry in EXTTYPES.L, then this option has no effect and EXTTYPES overrides are always used. See "Transfer Overrides" on page 108 for more information.
SET SSL ...	Sets SSL30 as the minimum SSL/TLS protocol version that clients are allowed to use to connect in to this server. Requires activating TCP/IP FOR VSE's SecureFTP optional feature. See the *TCP/IP FOR VSE Optional Features Guide* for more information on this SET command. **Note**: SET TLS*xx* is recommended over SET SSL. See SET_TLS*xx* on page 104.
SET STAMP [NONE|LEFT|RIGHT]	Specifies how to place timestamps in messages sent to SYSLST. **Value** — **Effect** NONE — Turns off timestamps. LEFT — Puts timestamps to the left of messages. RIGHT — Puts timestamps to the right of messages. (The default)
SET TELNTRAN *table-name*	Sets the translation table name to be used for the control connection. This value overrides the SET TELNET_TRANSLATE setting for the TCP/IP partition. For details, see SET TELNET_TRANSLATE in the *TCP/IP FOR VSE Command Reference*.
SET TERSE [ON|OFF]	ON causes shortened (one-line) 150 and 226 messages to be used for GET and PUT requests. The default is OFF.

SET Command	Description

SET TLS10 ...
SET TLS11 ...
SET TLS12 ...
	Sets TLS 1.0, TLS 1.1, or TLS 1.2 as the minimum SSL/TLS protocol version that clients are allowed to use to connect in to this server. Requires activating TCP/IP FOR VSE's SecureFTP optional feature. See the *TCP/IP FOR VSE Optional Features Guide* for more information.

Controlling the FTPBATCH Partition from the Console

You can issue a MSG *xx*,DATA=*command* from the VSE system console to display status information or shut down the FTPBATCH server. The partition ID *xx* must be specified along with a specific *command*. Commands are described in the following table.

DATA= Command	Description
ABORT ALLDATA	Terminates all active data transfers for any session with an open data connection.
SHUTDOWN	Terminates the FTPBATCH server.
STATUS	Same as STATUS SERVER, or just MSG *xx*.
STATUS DEBUG	Displays debugging information for the FTPBATCH server
STATUS EVENTS	Causes an event report to be generated for diagnosing a problem. Use only when requested by CSI Technical Support.
STATUS SERVER	Displays global information, including the current number of active sessions and maximum sessions allowed. It also displays detailed information about each active FTP session connected into this server.
STATUS SUMMARY	Displays the total amount of data sent and received by this server

Note:

When FTPBATCH initializes, either as a client or an external FTP server, it makes a copy of the TCP/IP stack's DEFINE FILE system within the partition GETVIS of the FTPBATCH job. When it processes files, FTPBATCH only refers to its own file system copy and not that of the TCP/IP stack. Therefore, if you make changes to the TCP/IP stack's file system (a DEFINE/DELETE FILE) and you want these changes to be reflected in the external FTPBATCH partition, you must shut down FTPBATCH and restart it. Upon start up, it will again check the stack and make a fresh copy of the file definition control blocks.

Using FTPBATCH to Create an External FTP Server with RAW Data Space

The external FTPBATCH daemon can also be used to send and receive files directly into a data space. The following sample JCL defines an FTPBATCH server with a RAW data space:

```
* $$ JOB JNM=FTPBDSPS,CLASS=P,DISP=K
* $$ LST CLASS=A
// JOB FTPBDSPS
// OPTION LOG,PARTDUMP
// OPTION SYSPARM='00'
// LIBDEF PHASE,SEARCH=prd2.tcpip
// EXEC FTPBATCH,SIZE=FTPBATCH,PARM='FTPDPORT=3121'
DEFINE DSPACE RAW RAWDSPAC 2000
/*
/&
* $$ EOJ
```

The following command creates a data space the size of 2,000 4K pages, or 8000K.

```
DEFINE DSPACE RAW RAWDSPAC 2000
```

The RAWDSPAC argument allows a single file to be pre-read into a data space before it is sent to a foreign FTP server. The size of any file sent or received into this data space must be 8000K or smaller. You can define any size data space up to the maximum amount of virtual storage available on your VSE system. This FTPBATCH data space server attaches an FTP daemon and listens on port 3121 for an FTP request.

The following line defines the data space to use for the RAW option:

```
DEFINE DSPACE RAW RAWDSPAC 256
```

The size of the data space is the number of 4K pages. In this example, 256 × 4096 = 1,048,576 (or 1 MB). This definition is the same as the following:

```
DEFINE DSPACE RAW RAWDSPAC 1M
```

or

```
DEFINE DSPACE RAW RAWDSPAC 1024K
```

As an example, here is a JCL that sends an SD file into a data space and then to a foreign FTP server:

```
* $$ JOB JNM=FTPBPUTR,CLASS=A,DISP=D
* $$ LST CLASS=F,DEST=(,AFTPDRS)
// JOB FTPBPUTR
// OPTION LOG
// OPTION SYSPARM='00'
// ASSGN SYS007,502
// DLBL SAMTEST,'SDCREATE.TEST.FILE',,SD
// EXTENT SYS007
// LIBDEF PHASE,SEARCH=prd2.tcpip
// EXEC FTPBATCH,SIZE=FTPBATCH
SET SHUTDOWN ON
LOPEN
LUSER userid
LPASS password
OPEN nnn.nnn.nnn.nnn 3121
USER userid
PASS password
ASCII
QUOTE TYPE I
PUT %SAMTEST,SAM,FB,80,800 %RAWDSPAC,DSPACE,RAW,TESTNAME
CLOSE
LCLOSE
*
TIMEWAIT 3  WAIT A FEW SECONDS...
* * Now open the ftpd in the other partition with
* * the data space as the local ftp connection...
LOPEN nnn.nnn.nnn.nnn 3121
LUSER userid
LPASS password
OPEN foreign
USER userid
PASS password
BINARY
PUT %RAWDSPAC,DSPACE,RAW  FTPBPUTR.TXT
LGOTOEOJ
CLOSE
LCLOSE
SHUTDOWN
/*
/&
* $$ EOJ
```

The key lines in this example are as follows:

- `SET SHUTDOWN ON`. This line causes the FTPBATCH job to require a shutdown command to terminate. This allows multiple FTP opens and closes to be issued without causing the job to terminate. SET commands are issued before LOPEN.

- `ASCII`. This line causes the data being sent to the FTPBATCH server to be translated from EBCDIC to ASCII. For example, this option is used when the SD file to be zipped contains EBCDIC-displayable characters and is being sent to a PC.

- `QUOTE TYPE I.` This line tells the FTPBATCH server to assume the data is binary. In this example, it was specified to translate the file from EBCDIC to ASCII. Without a QUOTE TYPE I statement, the FTPBATCH server would translate the received data back to EBCDIC. One of the nice things about the FTPBATCH server is that all data translation occurs before the data is sent, and you can use all the standard FTP translation tables with it.

- `PUT %SAMTEST,SAM,FB,80,800 %RAWDSPAC,DSPACE,RAW,TESTNAME`. This line causes the data to be read from the input SD file. It is received and stored into the data space of the FTPBATCH server.

  Note: These files are specified using autonomous-file syntax. Autonomous files are files that have not been defined to the TCP/IP FOR VSE file system using DEFINE FILE. For more information on using this syntax in FTPBATCH statements, see the *TCP/IP FOR VSE User Guide*, chapter 2, “FTP,” subsection “Autonomous Files.”

- `CLOSE`; `LCLOSE`. These two lines close the connection to the FTPBATCH server and to the current local connection, respectively. The data is stored in the data space owned by the FTPBATCH server.

- `TIMEWAIT 3`. This line causes a 3-second delay and allows the FTP daemon attached in the partition to restart and listen on its port.

- `LOPEN` *`nnn.nnn.nnn.nnn`* `3121`; `LUSER` *`userid`*; `LPASS` *`password`*. These three lines connect to the FTPBDSPS partition as a local FTP server, allowing the file to be read and sent from the data space to the final remote FTP server destination.

- `OPEN` *`foreign`*; `USER` *`userid`*; `PASS` *`password`*. These lines connect to the remote FTP server. This is the first operation attempting to establish a connection to the remote PC or UNIX system, for example. The file on VSE has already been completely read and translated and is now sent to the final destination.

- `BINARY`. This line specifies that the data be treated as binary data. The data space contains the translated data, and without this command it would be translated twice.

- `PUT %RAWDSPAC,DSPACE,RAW FTPBPUTR.TXT`. This line takes the translated copy of the file out of the data space and uses the external file name FTPBPUTR.TXT on the remote FTP server.

- `LGOTOEOJ`. This line tells the server to go to the End of Job.

- `CLOSE`; `LCLOSE`. These lines close the connections.

- `SHUTDOWN`. This line is needed to terminate the job because SET SHUTDOWN ON was previously issued in the JCL.

Controlling Defaults Using EXTTYPES.L

The member EXTTYPES.L contains definitions that are used by FTP to control file transfers. This section describes these definitions.

EXTTYPES.L is also used by the HTTP daemon and the EMAIL client. For information about how the HTTP daemon uses MIME content-type definitions, see chapter 8, “Configuring the HTTP Daemon.”

For information about how these definitions are used with the EMAIL client, see the *TCP/IP FOR VSE User Guide*, chapter 6, “TCP/IP for VSE Email,” subsection “Using Email Client Commands.”

File Types and Translation

There are times when you want TCP/IP FOR VSE to perform EBCDIC-to-ASCII translation and times when you do not. In general, the FTP client commands ASCII and BIN control the translation function:

- ASCII requests translation
- BIN prevents translation.

If you are transferring any kind of text file, you want to perform translation. If you are transferring something that is already in EBCDIC, such as a TCP/IP FOR VSE service pack, you need to prevent translation.

Transfer Overrides

When TCP/IP FOR VSE emulates a UNIX system, the PC system assumes that no matter what the user says, the file must be translated in binary mode. Because VSE is really an EBCDIC system, we need some way of overriding this behavior on the part of the PC. This is where the EXTTYPES.L configuration file comes in. Definitions in EXTTYPES.L can force ASCII-to-EBCDIC translation to occur even when you are running in UNIX Emulation Mode.

When you transfer a file, its type is checked against records in EXTTYPES.L. Each record maps a file type to a transfer type with default values. You control which defaults are used for a file type by specifying one of five transfer types. **It is important to understand that the EXTTYPES.L definitions override a user’s selections, regardless of whether the user is in UNIX Emulation Mode, so use care when modifying this file.**

You can suppress using the EXTTYPES.L overrides by

- Using the SITE EXTTYPES OFF command to disable EXTTYPES for a single session/FTPBATCH job.
- Using the SET EXTTYPES OFF command in an individual FTPBATCH job or in the FTPBATCH.L member. This disables EXTTYPES processing for all FTPBATCH jobs.
- Configuring an FTP server with the EXTTYPES=OFF parameter setting.

If you disable using EXTTYPES.L, or if EXTTYPES=ON but a matching file type is not found in the member, processing proceeds as follows:

- If SITELAST=NO (the default), then parameter settings on the file's DEFINE FILE statement take precedence over SITE commands.
- If SITELAST=YES, then the following SITE commands override the parameter setting on the file's DEFINE FILE statement:

  SITE BLKSIZE

  SITE CC

  SITE CRLF

  SITE LRECL

  SITE RECFM

  SITE TRCC

When EXTTYPES processing is in effect, default values for these parameters are used. The next section describes the values associated with each transfer type to which a file type may be mapped.

EXTTYPES.L Definitions

The EXTTYPES.L member contains comments and data records. Comment lines contain an asterisk in column one and enable you to annotate records in the file. Data records contain three positional fields. These fields, which are separated by at least one blank, are as follows:

1. **File type**. TCP/IP FOR VSE determines whether the file type exactly matches any portion of the public name on VSE. If it does, the transfer type in the second positional field determines how the transfer is performed. For more information on public names, see chapter 1, "Fundamentals of TCP/IP."

   Refer to the EXTTYPES.L member in use on your system to determine which transfer type is specified for each file type.

2. **Transfer type**. The values are BIN, BIN80, STRING, TEXT, and TEXTC, as follows:

   - **BIN**. Indicates that the file contains binary data and is to be transferred without translation.

BIN Transfer Settings
TYPE=Binary, STRUCTURE=File, MODE=Stream
LRECL/BLKSIZE=4096
RECFM=S (string)

- **BIN80**. Indicates that the file is to be transferred without translation and that the data is to be divided into 80-byte chunks. This is useful for putting binary data into VSE/POWER queues.

BIN80 Transfer Settings
TYPE=Binary, STRUCTURE=File, MODE=Stream
LRECL/BLKSIZE=4096 bytes (may be transformed to 80 bytes by the POWER file I/O driver)
RECFM=FB

- **STRING**. Indicates that the TCP/IP FOR VSE FTP daemon is to store the data in a VSE library using RECFM=SV. This transfer type is used only when you are transferring data into a VSE library.

STRING Transfer Settings
TYPE=Non-binary, STRUCTURE=File, MODE=Stream
LRECL/BLKSIZE=4096 bytes (may be transformed to 80 bytes by the POWER file I/O driver)
RECFM=SV (string variable)

- **TEXT**. Indicates that the file is to be transferred using ASCII-to-EBCDIC translation (or the reverse, as needed). Inbound data is text, stored in EBCDIC.

TEXT Transfer Settings
TYPE: Non-binary, STRUCTURE: File, MODE: Stream
LRECL/BLKSIZE: • Non-POWER files: 80 bytes • POWER files: 80 bytes (the POWER I/O driver may override this value for print-queue entries)
RECFM: • Non-POWER files: FB • POWER files: V

- **TEXTC**. Can be used to store records longer than 80 bytes. On upload, it splits the record at column 79 and puts an ‘&’ in column 80 to indicate a split. When the file is downloaded or displayed in the HTTPD daemon, the records are joined together.

The TEXTC defaults are as follows:

TEXTC Transfer Settings
TYPE: Non-binary, STRUCTURE: File, MODE: Stream
LRECL/BLKSIZE: 80 bytes
RECFM: • LIBR files: FB • HFS files: V • All others: VB

3. **MIME content-type header**. The TCP/IP FOR VSE FTP daemon ignores this string, which is used by the HTTP daemon. *MIME* stands for Multipurpose Internet Mail Extensions. This header identifies file formats on the Internet. See chapter 8, "Configuring the HTTP Daemon," for more information.

There is one keyword parameter that you can specify on any record in the EXTTYPES.L file. If you specify the keyword FTP=NO, the record determines whether EBCDIC-to-ASCII translation should be performed only when the file is requested by an HTTP daemon. When the file is requested by an FTP daemon, the record does not determine whether a translation is to be performed.

Modifying EXTTYPES.L

The EXTTYPES.L member is located in the TCP/IP FOR VSE install library and is loaded each time you start an FTP daemon. If you change EXTTYPES.L, you must delete and redefine all of your FTP daemons. You do not need to bring down TCP/IP FOR VSE to do this.

To change EXTTYPES.L, first make a copy and keep this copy in a configuration library. Then, change the copy in the new sublib only, and keep the original as it was distributed in the install library. This protects the modified copy from being overwritten when a new release is installed. To use the modified copy, add the sublib that contains this copy to your library search chain that is read when TCP/IP FOR VSE starts up.

Default Member

The following records are in the EXTTYPES.L member that is supplied with TCP/IP FOR VSE.

File Type ↓ Transfer Type ↓ MIME Content Type ↓

```
AIF     BIN    audio/x-aiff
AIFC    BIN    audio/x-aiff
AIFF    BIN    audio/x-aiff
ASC     TEXT   text/plain
AU      BIN    audio/basic
AVI     BIN    video/x-msvideo
BIN     BIN    application/octet-stream
BINJOB  BIN80                           (FTP only)
BJB     BIN80                           (FTP only)
BMP     BIN    image/bmp
CAB     BIN    application/octet-stream
CLA     BIN    application/java
CLASS   BIN    application/java
CSS     TEXTC  text/css
CSV     TEXT   application/csv
DOC     BIN    application/msword
DUMP    BIN                             (FTP only)
EXE     BIN    application/octet-stream
FLR     TEXT   x-world/x-vrml
GIF     BIN    image/gif
HTM     TEXTC  text/html
HTML    TEXTC  text/html
HTMLB   BIN    text/html
HTMLS   STRING text/html
ICA     TEXT   application/x-ica
ICO     BIN    image/x-icon
JAR     BIN    application/octet-stream
JAVA    BIN    text/plain
JPE     BIN    image/jpeg
JPEG    BIN    image/jpeg
JPG     BIN    image/jpeg
JS      TEXTC  application/x-javascript
LOG     TEXT   text/plain
MID     BIN    audio/midi
MIDI    BIN    audio/midi
MOV     BIN    video/quicktime
MPE     BIN    video/mpeg
MPEG    BIN    video/mpeg
MPG     BIN    video/mpeg
MPGA    BIN    video/mpeg
MP2     BIN    video/mpeg
MP3     BIN    video/mpeg
OBJ     BIN80                           (FTP only)
PDF     BIN    application/pdf
```

(continued next page)

```
PHASE   BIN                              (FTP only)
PNG     BIN    image/png
PRO     TEXT   application/octet-stream
PY      TEXT   application/x-pythonwin_py
QT      BIN    video/quicktime
RA      BIN    audio/x-realaudio
RAM     BIN    audio/x-pn-realaudio
RM      BIN    audio/x-pn-realaudio
RSS     BIN    application/xml
RTF     BIN    text/rtf
RTX     BIN    text/richtext
SHTML   TEXTC  text/html
SND     BIN    audio/basic
SWF     BIN    application/x-shockwave-flash
TAR     BIN    application/x-tar
TEXT    TEXT   text/plain
TIF     BIN    image/tiff
TIFF    BIN    image/tiff
TTF     BIN    application/octet-stream
TXT     TEXT   text/plain
VCF     BIN    application/octet-stream
WAV     BIN    audio/x-wav
WRL     TEXT   x-world/x-vrml
WRZ     TEXT   x-world/x-vrml
XLS     BIN    application/excel
XLT     BIN    application/excel
XML     TEXTC  application/xml
XSL     TEXTC  application/xml
ZIP     BIN    application/zip
```

Transfer Examples

Here are a few examples of how data is translated in an FTP transfer:

- You are transferring a PC file to a VSAM dataset. The VSAM dataset has a public name defined to the TCP/IP FOR VSE file system as VSAM.DON.TEXT. TCP/IP FOR VSE automatically performs an ASCII-to-EBCDIC translation on this file because “TEXT” is the third component of the public name.

- You are attempting to install a TCP/IP FOR VSE service pack. The name of the PC file is TCPIP15E.BJB, and the public name of the file into which you are trying to transfer the file is POWER.RDR.ATCPIP15E.BJB. TCP/IP FOR VSE automatically performs a BIN80 file transfer because that transfer type is specified in the BJB file-type entry in the EXTTYPES.L member.

- You are transferring the text file JOHN.SND into a VSE library with a public name of PRD2.SND. Even though the file you are attempting to transfer is a text member, TCP/IP FOR VSE performs a binary file transfer because the second component of the public name, SND, matches a file type in the table that is mapped to the BIN transfer type.

7

Configuring the Line Printer Daemon

Overview

The Line Printer Daemon (LPD) is a standard protocol that allows z/VSE users with TCP/IP to accept print data from TCP/IP hosts using the Line Printer Requester and Line Printer Daemon (LPR/LPD) protocols. After the data arrives on VSE, it can be printed, stored in a VSE library, or saved in a VSAM file.

Defining the LP Daemon

Use the DEFINE LPD command to initiate a line printer daemon (server). This command has the following syntax:

```
DEFine LPD,PRinter=pname,Queue=pubname [,LIBrary=libname]
       [,SUBlibrary=sublibname] [,TRANslate=name1]
       [,HEXdump=YES|NO ] [,USERid=id,PASSword=password ]
```

LPDs enable remote hosts to send files to VSE/POWER or to VSE-based datasets. The remote host must be capable of transmitting the files using the LPR/LPD protocol (this is not FTP). You must define each virtual printer that is supported.

The LPD parameters are described in the following table:

Parameter	**Description**
PRinter= *pname*	Specifies a 1- to 16-character printer name (case sensitive). This name must be unique and be known to external clients.
Queue = *pubname*	Specifies the location to receive the output routed to this daemon. This location must be a valid public name from the TCP/IP FOR VSE file system. The sections that follow provide additional information. See the section "TCP/IP FOR VSE File System" on page 24 for information on defining public names.
LIBrary= *libname*	Specifies a library to be used for temporary file storage. If you do not specify a library name, incoming data is stored temporarily in memory. This means that the largest file that can be handled is limited to available memory.
SUBlibrary= *sublibname*	Specifies a sublibrary for temporary storage of a file. This parameter is meaningful only if you have specified a value for LIB=. If omitted, TCP/IP FOR VSE uses the default sublibrary TEMP.
TRANslate= *name1*	Specifies the name of the table to be used for ASCII-to-EBCDIC translation.
HEXdump= [YES | NO]	Enables an optional debugging dump. If set to YES, inbound files are converted to hexadecimal dump format. The contents of the control file are included. This can be useful for debugging LPR/LPD errors. If set to NO (the default), files are stored normally.

Parameter	Description
USERid=*id*, PASSword= *password*	The user ID and password are 1- to 16-byte values (default is $LPD). These strings are passed to the user-defined security exit, and these fields can be omitted from the DEFINE LPD if your exit does not need them. Use them to meet in-house LPD security requirements.

The following display shows a sample command:

```
F8-043 IPN300I Enter TCP/IP Command
F8-043
F8-043 def lpd,printer=local,queue='power.lst.a',lib=prd2,sublib=save
F8-043 IPN444I    Print Name: LOCAL
F8-043 IPN445I         Queue: POWER.LST.A
F8-043 IPN446I       Library: PRD2.SAVE
F8-043 IPN300I Enter TCP/IP Command
```

Notes:

- A single LPD can handle any number of simultaneous requests. By default, the listings are buffered in memory until they are complete. When complete, the listing is sent to its final destination. We recommend that you specify a library and sublibrary for temporary storage.
- The sublibrary used for temporary storage of incomplete files should not be used for other purposes and should specify REUSE=IMMEDIATE.
- See the *TCP/IP FOR VSE Command Reference* for information on the following related commands.

Command	Task Performed
DEFINE FILE	Defines a file and associates it with a file I/O driver
DEFINE USER	Creates a user ID and password
DELETE LPD	Terminates a Line Printer Daemon
QUERY FILES	Displays the TCP/IP FOR VSE file system contents
QUERY LPDS	Displays the status of LPDs

Writing to VSE/POWER

Writing to a POWER queue is possible only if the special dataset name POWER is assigned a public name using a DEFINE FILE command. The remainder of this discussion assumes that you have followed the recommended procedure of assigning the public name POWER to the special dataset name POWER.

In the DEFINE LPD command, you must provide a QUEUE specification that consists of public name POWER qualified with the POWER queue and class. For example, you might specify QUEUE='POWER.LST.A'.

When a client LPR establishes a link to the daemon, it may pass a *jobname*. This is defined in the LPR protocol. If *jobname* meets the syntactic requirements of a POWER job name, it is used in that manner. Otherwise, a job name is constructed as LPDFA*nnn*, where *nnn* is the three-digit job number transmitted by the LPR client (transmission of a job number is required by the protocol).

Writing to a VSAM KSDS

QUEUE= must specify a fully qualified public name from the TCP/IP FOR VSE file system (no directory structure is available for a KSDS). Any records sent by the LPR client are passed to VSAM as an INSERT. Except for special purposes, this type of operation has little use.

Writing to a VSAM ESDS

QUEUE= must specify a fully qualified public name from the TCP/IP FOR VSE file system (no directory structure is available for an ESDS). Records sent by the LPR client are appended to the end of the dataset.

Writing to a VSE Library

QUEUE= must specify the public name assigned to a VSE library, qualified with a sublibrary name. For example, if the public name is PRD2 and the desired sublibrary is LST, then code QUEUE=PRD2.LST. When a client LPR establishes a link to the daemon, it may pass a job name. The job name is truncated to eight characters.

The member type is derived from the transmitted origin name (defined in the protocol). If no name is available, or if it is unsuitable for use as a member type, the word LISTING is used.

If a duplicate member exists, it is overwritten.

Supported LPR Subcommands

How LPR is used varies greatly by platform. Most LPR implementations send a control file along with the data. The control file contains information such as what to call the data file once it arrives, how to format it, and so on. In addition, many LPR implementations can send status-oriented commands to the LPD. TCP/IP FOR VSE supports the following control file and status commands.

Command	**Description**
SHORT	Lists the jobs that are currently on the LPD queue. The list includes job name and job number.

Command	**Description**
LONG	Lists the jobs that are currently on the LPD queue in a slightly longer format. The list includes job name, job number, host name, origin name, and other data. The list format is the responsibility of the LPR client.
START	Processes and ignores the START PRINTER directive.
REMOVE	TCP/IP FOR VSE does not support the REMOVE LISTING directive.
CONTROL(ORIGIN)	Records the origin of the print job.
CONTROL(JOBNAME)	Records the name of the print job. The jobname is an eight-character identifier that is appended to the queue name. For example, if the queue name is POWER.LST.Z, the job name might be POWER.LST.Z.MYJOB. TCP/IP FOR VSE then appends the origin (if one was specified) or the word LISTING. If the queue name is POWER, this final part of the job name is truncated. Note that the LPR might specify an origin using a method that you might not recognize. For example, the LPR that is supplied with TCP/IP for VM specifies your user ID as the origin.
CONTROL(HOST)	Records the host for the print job. This information is listed if the LPR client issues the LONG command.
CONTROL(CLASS)	Records the class of the print job. This information is not used.
CONTROL(SOURCE)	Records the source user ID of the print job. This information is not used.
CONTROL(FORMAT)	Formats the print listing. Formatting includes ASCII-to-EBCDIC translation. In addition, if the public file name has a RECFM or LRECL associated with it, that information is used to format the print file. TCP/IP FOR VSE processes the CR, LF, and HT ASCII characters, inserting the appropriate EBCDIC characters into the print stream.

8

Configuring the HTTP Daemon

Overview

TCP/IP FOR VSE's HTTP daemon allows your VSE installation to support its own website that can be accessed by popular web browsers such as Microsoft's Internet Explorer®.

A single HTTP daemon can serve many users. There is no requirement to define multiple HTTP daemons.

Web Server Functions

In general, TCP/IP FOR VSE's web server provides the typical features that you would expect from any web server. These features include, but are not limited to, the following functions:

- Displaying static web pages. A static web page is coded using a standard web page markup language such as HTML and XML. By using facilities such as SSI (Server Side Includes) and special internally defined variables within the web page, the static file that is served to the client can appear to be dynamic. In reality, however, it does not contain variable data. As with any web page, you may have interactive forms embedded in the static text to generate dynamic pages.

- Displaying dynamic web pages that are based on data residing in VSE. The TCP/IP FOR VSE web server contains support for the Common Gateway Interface (CGI) protocol, using Assembler or REXX as the foundation for creating dynamic pages. This means that you can have programs run as a subtask of the TCP/IP FOR VSE HTTP daemon. These programs can create HTML dynamically and pass it back to the browser. Using these techniques, you can create HTML output based on live data that resides directly on your VSE system.

- Running non-mainframe programs stored on VSE such as Java applets, .NET, and JavaScript files. Keep in mind that such items, although stored on VSE, are not VSE programs and are using VSE as a repository to deliver these files to the client. In addition, a web page stored on VSE can refer to applets and graphics that are residing on non-VSE platforms. To HTTPD, these are simply files that are delivered, but to the end-user they may be simple animated graphic displays or complicated, full-blown socket applications.

A single web page can contain one or more of each of these categories.

While you can certainly use VSE to display static documents, the real power of the TCP/IP FOR VSE web server is in its programming interfaces. See the *TCP/IP FOR VSE Programmer's Guide* for information about how to create CGI programs that run with the TCP/IP FOR VSE HTTP daemon.

Defining the HTTP Daemon

DEFINE HTTPD Command

To define and initialize an HTTP daemon, issue the following command:

```
DEFine HTTpd,ID=name,ROOT='pubname'
```

There are many more parameters, but these are the minimum required for defining the HTTP daemon. See the *TCP/IP FOR VSE Command Reference* for a complete listing and description of each parameter.

Note:

If you use the HTTP daemon with security on, you are required to have a special user ID defined to your TCP/IP FOR VSE system, as follows:

```
DEFINE USER,ID=$WEB,PASS=$WEB,WEB=YES
```

See the section “Security” on page 132 for more information on this topic.

Related Commands

See the *TCP/IP FOR VSE Command Reference* for information about the following commands:

- DELETE HTTPD
- QUERY HTTPDS

HTML Files

Web pages consist of HTML code and other objects such as sounds and pictures. When a web client receives a particular web page, it determines what other objects it wants to request and does so, sometimes in parallel with other HTTP requests. It is the job of the HTTP daemon to locate and access the various HTML, graphics, and other files that are being requested as they arrive to VSE.

File Location

Before you define an HTTP daemon, you must specify a default library to contain the HTML files and other web objects. You must specify the public name for this library in the ROOT= parameter on the DEFINE HTTPD command. The library you select *must* be defined to the TCP/IP FOR VSE file system.

The HTTP daemon is responsible for sending files that are requested by web browsers. The requests can come directly from the Uniform Record Locator (URL), or they can come from HTML tags in other HTML documents. In either case, the HTTP daemon needs to make sense of the request and return the requested object, such as a JAVA applet, a sound clip, or a file.

From a VSE perspective, web browsers assume that the entities serving the requested documents use hierarchical file systems. Because this may not be the case, TCP/IP FOR VSE must emulate this type of system to the browser. An administrator setting up a VSE-based website must understand this process in order to name the files appropriately.

File Location Algorithm

When TCP/IP FOR VSE receives a request for a file from a web browser, it must determine the exact name of the file. To do this, the HTTP daemon uses the following algorithm:

1. It starts with the string the user typed into the URL following the IP address of the VSE machine. It appends this name to the root name. It then appends the string ".INDEX.HTML" to the result.

2. The resulting name is processed left to right until a fully qualified name is located. The fully qualified name is assumed to be a library. The next-to-last qualifier is used as the member name, and the last qualifier is used as the member type.

3. Unused qualifiers are considered to be extraneous and are discarded. This means that the appended .INDEX.HTML is ignored if the file name built from the first two steps is a fully qualified file name. If the name built by the first two steps references a public name only, the appended .INDEX.HTML is used to find the complete file name.

4. If the requested file does not exist, the HTTP daemon retries the search without using the root value. It assumes that the requested file is a fully qualified public name. You can prevent this second search by coding CONFINE=YES in the DEFINE HTTPD command.

File Location Examples

Here are some examples. To begin, assume that your HTTP daemon is defined with the following command:

```
DEFINE HTTPD,ID=HTTP1,ROOT='PRD2.HTML',CONFINE=YES
```

In addition, assume that the IP address of your VSE machine is 192.168.0.9 and your web browser is pointed at HTTP://192.168.0.9/.

Example 1

Using the information above, TCP/IP FOR VSE begins to build a fully qualified file name. It begins with the root you specified, which is PRD2.HTML. It appends everything to the right of the final '/'. In this case, there is nothing after the slash, so the name is still PRD2.HTML. TCP/IP FOR VSE then appends .INDEX.HTML and returns the HTML code stored in sublibrary PRD2.HTML, member INDEX.HTML, because the fully qualified file name is PRD2.HTML.INDEX.HTML.

Example 2

For the second example, assume that your web browser is pointed at HTTP://192.168.0.9/MYPAGE.HTML.

TCP/IP FOR VSE starts at the root you specified, which is PRD2.HTML. It appends everything to the right of the final '/', which is MYPAGE.HTML, thus forming the file name PRD2.HTML.MYPAGE.HTML. TCP/IP FOR VSE then appends .INDEX.HTML to the right of this string, forming the name PRD2.HTML.MYPAGE.HTML.INDEX.HTML. But, because the first four qualifiers represent a fully qualified public name, the remaining two qualifiers are discarded. Therefore, the HTML document that is returned resides in PRD2.HTML.MYPAGE.HTML.

Example 3

For the third example, assume that your web browser is pointed at HTTP://192.168.0.9/PRD1.MACLIB.WTO.A.

TCP/IP FOR VSE starts at the root you specified, which is PRD2.HTML. It appends everything to the right of the final '/', yielding a file name of PRD2.HTML.PRD1.MACLIB.WTO.A. TCP/IP FOR VSE appends .INDEX.HTML to the right of this string and attempts to open the file. In fact, TCP/IP FOR VSE attempts to load a member name of PRD1.MACLIB in library PRD2.HTML. In all likelihood, this fails.

If you specified CONFINE=YES, the search stops here, and the web browser receives error 404 FILE NOT FOUND. If you specified CONFINE=NO, the search begins again.

This time, instead of starting at the root, TCP/IP FOR VSE begins with a null string, appends everything to the right of the final '/', and derives a file name of PRD1.MACLIB.WTO.A. As in Example 2, .INDEX.HTML is appended. Because the first four qualifiers form a valid fully qualified public name, the .INDEX.HTML is discarded. The web browser receives a copy of WTO.A, which is in VSE sublibrary PRD1.MACLIB.

Although this scheme may seem a bit convoluted, it is enormously flexible. The ROOT parameter is not required to point to a VSE sublibrary. For example, say that you want to create multiple web applications named Webappl1 and Webappl2. You want both applications to be accessible to your web browser clients, but you do not want to have a main menu with links. You can (1) define a VSE library named HTML with sublibraries Webappl1 and Webappl2, (2) define this library to the TCP/IP FOR VSE file system, and (3) use the command in Example 4 below to define the HTTP daemon.

Example 4

The fourth example uses the following HTTP daemon definition:

```
DEFINE HTTPD,ID=WEBAPPL,ROOT='HTML',CONFINE=YES
```

In this example, assume that your web browser is pointed at HTTP://192.168.0.9/Webappl1 and that the DEFINE command used in our third example is in effect.

TCP/IP FOR VSE starts with the root HTML and appends everything to the right of the '/', forming the public name HTML.WEBAPPL1. TCP/IP FOR VSE appends .INDEX.HTML and attempts to open HTML.WEBAPPL1.INDEX.HTML, which is probably the name you want.

Example 5

The fifth example points out the type of problem that can occur when you specify file names the way we did in Example 4. For this example, assume that your web browser is pointed at HTTP://192.168.0.9/ .

TCP/IP FOR VSE starts with the root HTML and appends everything to the right of the '/', which is nothing. TCP/IP FOR VSE appends .INDEX.HTML, forming a public name of HTML.INDEX.HTML. This name does not mean anything to the TCP/IP FOR VSE file system. Because you specified CONFINE=YES, the search ends with error 404 FILE NOT FOUND.

Non-Standard File Names

As you can see, the standard file location algorithm works well when your HTML document names fit into the standard eight-character by eight-character member name format that is supported by VSE sublibraries. Sometimes you cannot choose the names assigned to different portions of your web applications. For example, you may port a web application from another platform that has names longer than eight characters.

Most web applications have objects (usually non-textual objects) that fall into this category. Because you cannot fit those names into the VSE Librarian structure, your options are to modify the names of the files in the application or use a VSAM-based pseudo-directory structure for your application.

To use a VSAM-based structure, follow these steps:

1. Define a VSE user catalog to hold the components such as HTML, video, and sound for the application. You can use an existing user catalog for this purpose.

2. Add the DLBL for your VSAM catalog to standard labels, to the partition label area, or to the TCP/IP FOR VSE startup JCL. The DLBL must be formatted so that the CAT= indicator points back to itself. For example, if you use a DLBL for IJSYSUC, then you must also have the parameter CAT=IJSYSUC.

3. Define your user catalog to the VSE file system as TYPE=VSAMCAT. For example, the following command defines the user catalog to the TCP/IP FOR VSE file system:

```
DEFINE FILE,PUBLIC='VSAMCAT',DLBL=IJSYSUC,TYPE=VSAMCAT
```

4. Define your HTTP daemon using your VSAMCAT definition as the root. The following definition accomplishes this:

```
DEFINE HTTPD,ID=HTTP1,ROOT='VSAMCAT',CONFINE=NO
```

5. Define all the components of your application such as HTML files, JPG files, and WAVV files as individual VSAM files. For example, if you want your application initiated with member INDEX.HTML, you need to define a VSAM cluster with a name of INDEX.HTML and catalog it to the VSAM catalog that is referenced by the DEFINE FILE statement above.

   Here is an example:

```
// JOB CATALOG INDEX.HTML
// DLBL OUTFILE,'INDEX.HTML',,VSAM,CAT=IJSYSUC
// EXEC IDCAMS
   DELETE -
          INDEX.HTML -
          PURGE
   DEFINE CLUSTER (NAME -
          (INDEX.HTML) -
          NONINDEXED RECFM(F) -
          RECORDSIZE(80 80) -
          RECORDS(100 100) -
          REUSE -
          VOLUMES(SYSWK1)) -
     DATA (NAME -
          (INDEX.HTML.D) REUSE) -
     CATALOG -
          (VSESP.USER.CATALOG)
   REPRO INFILE(SYSIPT,ENV(RECFM(F))) -
          OUTFILE(OUTFILE) REPLACE
/*
/&
```

With text it is straightforward to populate these VSAM files. With binary data, such as in JPG files and sound files, it is more difficult. You need to execute the DEFINE CLUSTER without the REPRO statements. Then you need to use FTP to PUT the data in binary form into the cluster area.

6. There may be some component names that are longer than eight characters. If this is the case, you must break the name into eight-character chunks before you do the definition. Here are examples:

    - TN3270PAINT.CLASS is defined as TN3270PA.INT.CLASS. Note that TCP/IP FOR VSE always respects a dot in the file name and only breaks longer component names into eight-character units.

    - MYREALLYLONGSOUNDFILE.WAVV is defined as MYREALLY.LONGSOUN.DFILE.WAVV

There are two more things you need to know:

- First, VSAM limits the file length to 44 characters (periods included).

- Second, unlike the Librarian method of defining the root directory for the HTTP daemon, the VSAMCAT method does not automatically append INDEX.HTML to the file name. Instead, TCP/IP FOR VSE checks whether the file name your web browser is requesting contains a dot (.). If the name does not contain a dot, TCP/IP FOR VSE appends INDEX.HTML to the name.

These restrictions do not apply to the HFS (Hierarchical File System).

Non-Standard File Examples

The examples in this section assume that an HTTP daemon is defined as follows:

```
DEFINE HTTPD,ID=HTTP1,ROOT='VSAMCAT',CONFINE=YES
```

Example 1

Assume that your web browser is pointed at HTTP://192.168.0.9/.

TCP/IP FOR VSE starts at the root, which is the VSAM catalog, and appends everything to the right of the slash. In this case, nothing is appended because there is nothing to the right of the slash. Next, it appends .INDEX.HTML because the requested file name (again, in this case, nothing) did not contain a dot. TCP/IP FOR VSE attempts to locate a VSAM cluster called INDEX.HTML in the catalog pointed to by the public name VSAMCAT.

Example 2

Assume that your web browser is pointed at HTTP://192.168.0.9/MYPAGE.HTML.

TCP/IP FOR VSE starts at the root, which is the VSAM catalog, and appends everything to the right of the slash. This returns a name of

VSAMCAT.MYPAGE.HTML. Because the requested file name contains a period, TCP/IP FOR VSE does not append .INDEX.HTML and returns the contents of the file contained in the VSAM cluster MYPAGE.HTML in the user catalog pointed to by VSAMCAT.

Member Types and Translation

Each webpage consists of text with embedded HTML tags. There are three distinct levels of HTML with additional extensions supported by Netscape® and Internet Explorer. The HTTP daemon is unaware of page content and does not care about the HTML version. It merely ships the data to the browser at the browser's request.

Member Types

When a request is received from a browser, the document that is returned is assigned a member type. TCP/IP FOR VSE always assumes the member type is denoted by the characters to the right of the final slash or dot in the file name. For example, the file INDEX.HTML is assigned a file type of HTML, whereas the file /USR/HTML/JOHNMILL.JPG is assigned a file type of JPG.

A web browser may fetch any type of member; however, the browser itself relies on the member type to indicate how it should be processed. In general, any member containing HTML-delimited text should have a file type of HTML.

Translation

There is, of course, the issue of EBCDIC-to-ASCII translation. How does an HTTP daemon know whether to convert a given file from EBCDIC to ASCII and, just as important, when NOT to convert a file from EBCDIC to ASCII? The HTTP daemon uses the same approach that FTP uses, namely, the parameter file named EXTTYPES.L.

The member EXTTYPES.L is located in your TCP/IP FOR VSE partition search chain. It is loaded every time you start an HTTP daemon. To change it, you must delete and redefine all of your HTTP daemons, but you do not have to bring down TCP/IP FOR VSE. The two types of records in the file are comments and data records. Comments begin with an asterisk in column one and allow you to annotate records in the file.

Data records contain three fields in positional order. The three fields are delineated by at least one blank. The three fields in EXTTYPES.L are file type, transfer type, and MIME-type header. These fields are as follows:

1. **File type.** TCP/IP FOR VSE determines if the file type matches the file that is requested by the web browser. If it does, it uses the data in the second positional field to determine how to perform the transfer.
2. **Transfer type**. The transfer types used by the HTTP daemon are as follows:
   - BIN (BINARY). The file is to be transferred without translation.
   - TEXT. The file is to be transferred with ASCII-to-EBCDIC translation. The data is text and is stored in EBCDIC.

- TEXTC. This type can be used to store records longer than 80 bytes. On upload, it splits the record at column 79 and puts an '&' in column 80 to indicate a split. When the file is downloaded or displayed in the HTTPD daemon or Entrée, the records are joined together.

3. **MIME content-type header**. The HTTP daemon does not use the MIME type (transfer-type) header but passes it along to the web browser by prefixing it to the data that is passed. In this way, the web browser knows how to process the file. The MIME-type headers that may be used depend on the web browser. If the MIME-type string is missing from a file-type record in the EXTTYPES.L file, the HTTP daemon ignores the line. (Some lines are used only by FTP.)

The following sample record maps the "PDF" file type to the "BIN" transfer type and the "application/pdf" MIME content type.

```
PDF     BIN    application/pdf
```

See "Default Member" on page 112 for a complete listing of the default file types.

HTTP Examples

Here are two examples that show how EXTTYPES.L is used by the HTTP daemon:

- Your web browser attempts to access PRD1.MACLIB.WTO.A. The HTTP daemon determines that the file type is 'A' and performs a lookup. Because a file type of 'A' is not defined to your web browser, a default MIME type of "plain text" is assumed.

- Your web browser attempts to access PRD2.HTML.VSEPICT.JPG. The HTTP daemon determines that the file type is JPG and, using EXTTYPES.L, sees that it needs to transfer this file to the browser in binary with a MIME type of "image/JPEG."

Graphics Files

You can prepare graphics files with any software you want. Typically, this is done on a PC. After you create an image, you must place it in the appropriate VSE library.

Loading

The graphic image is stored in a string member the same way that phases or other binary data are stored. The easiest way to load your graphic data into a member is to use FTP. Be sure that your FTP client transmits the file with TYPE set to BINARY and record format set to 'S'.

Hint: If your PC-based FTP client has no provision for issuing SITE commands, perform the transfer using TCP/IP FOR VSE's CICS or batch FTP client, or simply name the file with an extension defined in EXTTYPES.L.

Displaying Data

To display a page of data, a web browser must fetch the HTML text page and all of the referenced graphic files. To speed up the process, some web browsers open multiple TCP/IP connections and download all of the graphics files simultaneously. Although this technique works well with more primitive platforms, the complexities of VSE limit its usefulness. In fact, once the available bandwidth is filled, opening additional connections slows the process while consuming additional machine resources. For this reason, you may want to set browser options that limit the number of concurrent sessions used by the client software.

In general, you need to limit the number of sessions only if you notice performance problems. Such noticeable problems would be partial data, web pages with pieces missing, or frames with error messages. If you cannot configure your web browser to get around this, then you should define multiple HTTP daemons. Then, if one daemon is busy, the other takes up the load. Usually, three daemons can handle this type of load in such an environment. You should, of course, test an appropriate number for your site if you encounter these types of problems.

CGI

The Common Gateway Interface (CGI) is a provision of the HTTP standard that allows you to display dynamically generated information. Starting with a static web page, HTML tags inform the browser to request execution of a CGI script. (In our case, this is an Assembler or a REXX program.) If user-entered data is provided through the use of HTML form tags, it is also uploaded as a parameter string.

When the request reaches the HTTP daemon, the requested CGI script or phase is fetched and the uploaded parameter string is passed to it for processing. The CGI script is expected to return either a dynamically constructed HTML page or the name of a static page to be fetched by the browser.

See the *TCP/IP FOR VSE Programmer's Guide* for information on how to create CGI scripts.

Server Side Includes

Introduction

The TCP/IP FOR VSE HTTP daemon includes a function known as a Server Side Include (SSI). An SSI enables you to include HTML lines from another file in the current transaction. For example, you might have HTML lines that define your corporate logo. You would not want to include these lines in every document. Instead, you can include them from a common file.

Syntax

The syntax of the Server Side Include directive is

```
<!-INCLUDE filename >
<!-INCLUDE FILE="filename" >
<!-EXEC CGI="cginame?parms">
```

The variables are as follows:

Variable	Description
filename	Specifies the fully qualified public name of any file defined to the TCP/IP FOR VSE file system
cginame	Specifies any CGI defined to TCP/IP FOR VSE using the DEFINE CGI command
parms	Specifies parameters to the CGI

How it Works

When the TCP/IP FOR VSE HTTP daemon sees an include filename directive using either syntax, it immediately opens the referenced file and begins reading additional HTML lines from that location. When the HTTP daemon finishes reading the referenced file, it continues reading the original file. There is no practical limit to the number of included files.

When the HTTP daemon sees an EXEC CGI directive, it immediately calls the referenced CGI and waits for the CGI to finish returning HTML lines. When the CGI finishes returning HTML lines, the HTTP daemon continues reading from the original file.

CGI Variables

There are special variables that are generated by HTTPD for passing back to a CGI to use. Each variable contains a piece of the directory location. For example, if the CGI is invoked from a root of "PRD2.HTML", then &1 will contain "PRD2" and &2 will contain "HTML" that is passed to the CGI.

Name	Meaning
&1, &2, &3, &4, &5, &6	First through sixth piece of the file name being returned to the client

HTTPD Variables

There are special names you can embed in your HTML that are passed back to the client. The HTTP daemon changes the names into an associated value, as follows:

Name	Meaning
&IP	IP address of the HTTP daemon
&FOIP	IP address of the web browser (client)
&LOPORT	Port of the HTTP daemon
&FOPORT	Port of the web browser (client)

URL Commands

There are two special commands that you can pass as part of your URL request. For HTTP, a question mark (‘?’) terminates the access string and allows you to append one or more “GET” parameters. You can use the commands in the table below individually or together as parameters. Use an ampersand (‘&’) to join a second parameter.

Command	Description	Equivalent FTP Command
CSI_PDF=ON|OFF	The file is converted to PDF before delivery.	SITE PDF
CSI_CC=ON|OFF	Carriage control is enabled	SITE CC

For example, to access a POWER print file called “TCPIP,” convert it to a PDF, and use the carriage control of that file, you might issue

```
HTTP://192.168.1.9/POWER/LST/A/TCPIP?CSI_PDF=ON&CSI_CC=ON
```

The CSI_PDF=ON command causes a PDF file instead of a plain text file to be delivered to the user. The CSI_CC=ON command enables carriage control for this print file.

These commands can also be embedded as hidden command fields so that the user just needs to click on a link to initiate the conversion.

The "OFF" value for each command can be used as a convenience. For example, you can make a field on a page called "&CSI_PDF" and have two radio buttons associated with it called "On" and "Off". When the user selects either option and presses a SUBMIT button, it generates a "&CSI_PDF=*value*" that is passed to the HTTPD.

Security

Traditionally, web sites are open and unsecured. Because web-browsing software does not pass user ID or password information, security is difficult to enforce.

Using the CONFINE=YES parameter on the DEFINE HTTPD command limits access to the file specified by the ROOT parameter of the same command. If ROOT specifies a partially qualified name, then access is limited to files lower in the directory tree.

If you have provided a security exit, it is called before each fetch operation. The IP address of the requester is passed, as well as the name of the file that is requested. For a higher level of security, you can code SECURE=YES on the DEFINE HTTPD command. You must also copy three provided HTML documents (PASSWORD.HTML, VIOLATED.HTML, and BLANKING.HTML) to the daemon's root directory or to the directory specified by the LIB= and SUBLIB= parameters on the DEFINE HTTPD command. After you do this, the HTTP daemon performs the following tasks:

- When a new network IP address requests an HTML document, the PASSWORD.HTML document is shipped instead. This document is a form that requests the user ID and password.
- The user ID and password are processed with the level of security currently in effect for TCP/IP FOR VSE.
- If the security check fails, the VIOLATED.HTML document is shipped. The next request displays the PASSWORD.HTML document again.
- If the security check passes, the requested document is shipped and the user's IP address is entered into a table of valid network addresses.
- Additional requests for documents are honored as long as the network address remains in the authorized table.
- If no request is received within the time limit that is defined on the TIMEOUT= parameter of the DEFINE HTTPD command, the expiring address is removed from the table. The next request requires the user to reenter the user ID and password.
- A user can manually expire a network address by linking to the non-existent document LOGOUT or by using the URL HTTP://*ip-addr*/logout, where *ip-addr* is the address of TCP/IP FOR VSE.

The contents of the three HTML documents can be customized as desired. The only restriction is that the form data appearing in PASSWORD.HTML must remain unchanged.

9

Security

Overview

TCP/IP FOR VSE has opened VSE communication paths to virtually every other computer platform in existence. With this new openness, you might find that you also have new security concerns.

Although TCP/IP FOR VSE is not a security product, it does provide numerous functions that secure system resources. In this chapter, we describe the functions we provide and discuss their pros and cons. This information enables you to make informed decisions about security and about which functions you can safely provide to the end user. In general, any security implementation must consider the following issues:

- Defining user IDs and passwords.
- Definitions of the resources you want to protect. These resources can be a functional entity such as an application or a physical entity such as a file.
- Authorizations defining which users can perform which functions and which users can access which resources.

Important! By default, security is off, and any user ID or password is accepted. If you do not know whether security is active, issue the following command immediately:

```
QUERY SECURITY
```

If the response indicates that security is OFF, immediately issue the following command:

```
SECURITY ON  MODE=WARN  AUTO=ON
```

This command activates security in WARN mode, and a message log is created that can help identify the user IDs being used and resources accessed. It still allows all logins and accesses because MODE=WARN is specified.

Remember that issuing the SECURITY OFF command leaves your system completely exposed to hackers and unauthorized users.

Details of securing your TCP/IP FOR VSE system are covered in the following sections.

SECURITY Command

The TCP/IP FOR VSE SECURITY command is used to control and manage TCP/IP FOR VSE security. This command contains one or more options, separated by blanks or commas, that define or modify current security settings. The global command options are described in the following table.

Option	**Description**
ON/OFF	Activates or deactivates global security processing in the TCP/IP partition
BATCH=ON/OFF	Activates or deactivates security processing in external batch partitions (for example, FTPBATCH)
ARP=ON/OFF	Controls ARP request checking
IP=ON/OFF	Controls IP address checking
MODE= WARN/FAIL	Controls whether security failures are allowed with a “warning” or refused with a “failure”
LOGGING= ALL/FAIL/NONE	Controls logging of security requests. The user-supplied Security Exit may set a flag to force logging of specific requests.
DUMP= ALL/FAIL/NONE	Controls dumping of the entire SXBLOK following a failed security request. NONE suppresses dumping; FAIL causes dumping when a security failure occurs; and ALL causes dumping of failures in both FAIL and WARN modes. The SXBLOK security exit block is described in a later section.
LOCK	All security settings are locked to their current values

The command options for controlling a User-Created Security Manager are as follows.

Option	**Description**
EXIT=ON/OFF	Controls loading and activating the User Security manager
PHASE=	Specifies the name of the User Security Manager phase
XDATA=	Specifies a 40-byte character string to be passed to the User Security manager with each call

Option	Description
ASMDATE=	Assembly date of the user security manager (1 to 8 characters)
ASMTIME=	Assembly time of the user security manager (1 to 8 characters)
VERSION=	Version of the user security manager (1 to 8 characters)
LEVEL=	Modification level of the user security manager (1 to 8 characters)

The command options for controlling the CSI-provided Automatic Security Manager are described in the following table.

Option	Description
AUTO=ON/OFF	Controls loading and activating the automatic security manager
ADATA=	Specifies a 40-byte character string to be passed to the automatic security manager with each call.

Defining User IDs

After a SECURITY ON has been issued, the first and easiest security that must be implemented is to activate user identification with passwords. A user ID and a password are required before the system allows access to secured TCP/IP FOR VSE applications such as FTP, Telnet, and HTTPD.

TCP/IP FOR VSE enables you to explicitly or implicitly define user IDs and passwords. You can explicitly create user IDS using the TCP/IP FOR VSE DEFINE USER command, or you can create a ‘user security manager’ to implicitly define users to TCP/IP FOR VSE. The security manager gets control and can permit or deny access based on user IDs and passwords, regardless of whether the user IDs and passwords are defined with DEFINE USER command. See the section “User-Created Security Manager” for more information on defining user IDs implicitly.

Explicitly Defining User IDs

The explicit mechanism used to define user IDs is TCP/IP FOR VSE’s DEFINE USER command. The syntax of this command is as follows.

```
DEFINE USER,ID=id[,PASSWORD=passwd][,DATA='data']
          [,FTP=YES/NO]  Controls FTP access by this user
          [,LPR=YES/NO]  Controls LPR access by this user
          [,WEB=YES/NO]  Controls Web page access this user
          [,TELNET=YES/NO]  Controls Telnet menu access
          [,ROOT='directory']  Restricts the user ID to a
                               specific directory
```

You can issue this command anywhere you can issue TCP/IP FOR VSE operator commands. This may include the console, an include deck, the IPNETCMD command processor, or the TCP/IP FOR VSE initialization deck. This implies the following loopholes in security:

- Anyone who can issue VSE operator commands can define user IDs. **To prohibit the use of the VSE operator console as a mechanism for issuing TCP/IP FOR VSE operator commands, use the SET PASSWORD command in the TCP/IP FOR VSE initialization deck.**
- If you choose to define your user IDs in the TCP/IP FOR VSE initialization deck, anyone with access to the initialization deck also has access to all user IDs and all passwords.
- When the DEFINE USER command is entered from the console, the password specified in the DEFINE USER statement displays on SYSLST. You can suppress the password by starting the command with a ‘+’ character. Any command that starts with a ‘+’ does not display on SYSLST.

Despite these limitations, the DEFINE USER command can provide effective protection for identifying who is accessing TCP/IP FOR VSE resources.

The PASSWORD option can be any value up to 16 bytes, and it is not case sensitive when being checked before calling the security manager, although the user security manager could enforce case sensitivity.

This command supports limiting a user ID to the following functions:

Option	**Description**
FTP=YES/NO	Controls FTP access by this user
LPR=YES/NO	Controls LPR access by this user
WEB=YES/NO	Controls Web page access by this user
TELNET=YES/NO	Controls Telnet menu access by this user

If NONE of the above options is explicitly coded, then ALL functions are permitted. If any of the above options are used, then the default for all un-coded options is “NO.” Here are three examples:

- The user has access to everything:

  ```
  DEFINE USER ID=ABC,PASS=XYZ
  ```

- The user has access to FTP only:

  ```
  DEFINE USER ID=ABC,PASS=XYZ,FTP=YES
  ```

- The user does not have access to anything:

  ```
  DEFINE USER ID=ABC,PASS=XYZ,FTP=NO
  ```

A root directory can be defined for a user with the DEFINE USER command using the ROOT= option. This restricts the user to that directory or lower. For example, to restrict a user ID to only the VSE/POWER LST queue class D, you could enter:

```
DEFINE USER,ID=ABC,PASS=XYZ,.ROOT='/POWER/LST/D',FTP=YES
```

Setting a ROOT of ‘/’ or ‘\’ starts the user in either UNIX or VSE mode, respectively.

Note that using the ROOT= parameter requires that SECURITY ON be active.

FTP Security

For files that are defined to the TCP/IP FOR VSE file system and are accessible through a public name, you must either activate the automatic security manager or code a user security manager to restrict access to the files. If you do not use a security manager, then any FTP user can access such files for both reading and writing.

FTP Autonomous Files

The TCP/IP FOR VSE FTP client and servers also allow files to be transferred without previously defining them to the file system. This type of file transfer is referred to as *autonomous FTP*. The files you transfer in autonomous FTP are specified by a '%' followed by a DLBL/TLBL (DDNAME) or data space name. See the *TCP/IP FOR VSE User Guide* for more information on specifying autonomous files in FTP.

There are two mechanisms for using autonomous FTP:

- You can use the FTP batch client, invoked by // EXEC FTPBATCH, and run the FTP in an external partition.
- You can use an internal or external FTP server, which is a daemon invoked by DEFINE FTPD or // EXEC FTPBATCH, respectively, and a remote FTP client.

Each of these choices has security ramifications. If you run the FTP in the TCP/IP FOR VSE partition, the TCP/IP FOR VSE security exit is called and is passed the DLBL that is being transferred as well as the user ID and password of the local user ID used to run the job. Your security exit must be coded to deal with autonomous FTP.

Your VSE security package may also be involved in this scenario. Any time TCP/IP FOR VSE attempts to open a file, your VSE security package (if you have one) must check whether the TCP/IP FOR VSE partition is authorized to open the file. TCP/IP FOR VSE does not do anything that bypasses your installation's VSE security system.

If, on the other hand, you run your FTP using // EXEC FTPBATCH, the VSE OPEN for the file is performed in the partition running FTPBATCH, and standard VSE security applies. If the partition running FTPBATCH is allowed to open the file, then TCP/IP FOR VSE permits its use, and if the partition running FTPBATCH is not allowed to open the file, FTPBATCH does not permit its use.

By default, all users are allowed to bypass the TCP/IP FOR VSE file system and access local VSE files using autonomous FTP. To restrict remote users' access to only files that have been defined to TCP/IP FOR VSE by the DEFINE FILE command, use the DYNFILE=NO parameter. This parameter can be used when configuring an FTPBATCH server (// EXEC FTPBATCH,PARM='FPTDPORT=*nnn*') or an internal server (DEFINE FTPD). Each FTP server can be configured individually to allow or deny transferring autonomous files on its associated port.

Note:

The LOCAL_DLBL OFF command provides the same restriction for all internal FTP servers and is supported for compatibility with earlier releases. You run this command from the global TCP/IP initialization member.

HTTP Security

The HTTP daemon is also capable of allowing users to read files. TCP/IP FOR VSE provides a function that allows you to force HTTP daemon users to log on with a user ID and a password. Any time a web client attempts to access a file, TCP/IP FOR VSE calls the security exit and does an authorization check. You can use the same logic to authorize HTTP access to files and FTP access to files, although the security exit may also be coded to differentiate between the two protocols.

DEFINE HTTPD Command

The USERID and PASSWORD parameters are included in the DEFINE HTTPD command. These parameters allow you to specify a default user ID and password for "unsecured pages." The defaults for both are "$WEB". This user ID and password must be defined using DEFINE USER with an access attribute of "WEB=YES".

Telnet Security

TCP/IP FOR VSE can provide a level of security for using the telnet facility. Because telnet is simply a conduit to an application, it is expected that the application requires a user ID and password if these are needed.

If you choose to provide a telnet logon menu, you may require a user ID and password field on the menu. (If these fields are omitted, they are not required.) See chapter 5, "Configuring the Telnet Daemon," page 70, for more information about defining a telnet menu.

You can also associate a particular IP address with a specified LU name. This also enables you to deny or restrict telnet access based on the originator's IP address. **Remember that the IP address is a mechanism that is under the client's control.**

If you code your telnet menu to allow access only to selected VTAM applications, other VTAM applications are inaccessible from TCP/IP FOR VSE. If you code your telnet menu to allow a generic logon, then any application is accessible.

Security Exit Points

TCP/IP FOR VSE provides security exit points for comprehensive security control. You may provide a routine that is called at various points in TCP/IP FOR VSE processing, including validation of user IDs and passwords and access to resources by a specific user ID.

Security Managers

A security manager is called from a security exit point, and based upon the security manager's return code, the operation is permitted, warned, or denied. The security exit can also log the accessed resource. The following security managers can be activated with the SECURITY command:

- TCP/IP FOR VSE automatic security manager
- User-created security manager
- IBM-provided security manager
- Other vendor-provided security manager.

Regardless of which security manager is activated, the TCP/IP FOR VSE exit passes control and information about the resource being accessed in a common security exit block (SXBLOK). Although you can create your own "user security manager," you should first take a close look at the automatic security exit provided with TCP/IP FOR VSE. It is the quickest and easiest to implement, and it is maintained and updated for you automatically by CSI International. The SXBLOK fields are described in a later section.

The automatic security manager and one other security manager may be active (user created, IBM provided, or other vendor provided).

Security managers are activated and controlled with the SECURITY command. See the *TCP/IP FOR VSE Command Reference* for details.

Security Request Flow

The flow of a security exit request may be important, depending on which security managers are used and activated. Following is the security flow through the various security managers.

1. A TCP/IP FOR VSE application, such as FTP, creates an SXBLOK control block in the security exit.
2. The User ID/password (if present) is checked against DEFINE USER information. The result is set in SXBLOCK along with a default return code.
3. If specified, "Automatic" processing is performed. The result is set in SXBLOK and overrides any return code set in step 2.
4. If specified, user exit processing is performed. The user exit may consider the result of the preceding steps or it may override it.

Auto Security Manager

Automatic security is activated with the SECURITY AUTO=ON command. “Automatic” security means that many users do not need to create and maintain their own security exits.

ASECURITY Command

The ASECUrity command provides control for system-level resources where no user ID and password have been established. The controls are

- ASECUrity ICMP=YES/NO

  Allows (YES) or prevents (NO) VSE from responding to incoming ICMP PING requests. This is useful for stopping sweeps that are commonly used to find active machines on a TCP/IP network. This control does not affect ping requests that originate on VSE.

- ASECUrity FTPD=YES/NO

  Controls connection requests to the FTP Daemon. NO stops new FTP sessions from being accepted by the VSE FTP Daemon. This parameter may be used to temporarily stop new FTP sessions without deleting your FTP daemons. Already-established FTP sessions are not affected. YES: Commands are processed normally.

  When a foreign client connects to the FTP Daemon, a “220-welcome” message is immediately sent to the foreign client. Executing ASECUrity FTPD=NO prevents sending the 220 message (and any user-defined WELCOME= message) and the connection request is simply terminated. This can be useful for preventing “banner grabbing” to find out where an FTP service is active.

- ASECUrity WEBL=YES/NO

  Web Logon Screen Request. For WEBL=YES: The HTTP daemon maintains a minimal level of access security based on the network address. To do this, the daemon maintains a table of “active” IP addresses. When a request is received from an address not in the table, the daemon automatically displays a page that requests a user ID and password. These values are checked through the standard TCP/IP FOR VSE mechanisms. If valid, the IP address is added to the table and the original request is transmitted. The IP address is removed from the table when explicitly requested (a request made for “BLANKING.HTML”) or when the HTTPD inactivity timer (TIMEOUT=) expires. For WEBL=NO: No automatic security checking is performed.

- ASECUrity SCAN=YES/NO

  HTTPD Scanblock Request. For SCAN=YES: If a remote user attempts to open a connection to the HTTP daemon, even partially (a half open), and then ends the connection without sending any data, TCP/IP FOR VSE tracks this event and blocks the user’s IP address after a predetermined number of such calls. The maximum number of such calls is always 3 unless BLOCKIP=YES and BLOCKCNt is set to either 1 or 2. See also BLOCKIP=. An HTTP908W message is always issued when a half-open connection is attempted. The ACCESS command can be used to reset the block for an IP address.

- ASECUrity BLOCKIP=YES/NO

  This parameter enables IP address blocking after the allowed number of security violations occurs. This number is set by BLOCKCNt=. If BLOCKIP=NO, then the IP address is not blocked after this number of security violations. The default (YES) is to block the address. The ACCESS command can be used to reset the block for an IP address.

- ASECUrity BLOCKCNt=*count*

  This is the value used by BLOCKIP to determine the maximum number of security violations before blocking that IP address. The count can range from 1 to 255. The default is zero (never).

- ASECUrity ARP=YES/NO

  Requires SECURITY ARP=ON to be in effect already. Specifying ARP=NO prevents TCP/IP FOR VSE from responding to inbound ARP requests. This could be useful for stopping all current inbound activity due to an ARP attack. But, its use should only be temporary because it can cause a loss of service for legitimate applications.

- ASECUrity IPAV=YES/NO

  Requires SECURITY IP=ON to be in effect already. Specifying IPAV=NO immediately prevents processing of all incoming IP datagrams. This is a drastic step, but one that might prove useful if you are in the middle of an Internet attack, as in a Denial of Service attempt. This parameter is different from ARP=, which is limited to ARP requests.

- ASECUrity FTPC=YES/NO

  For FTPC=NO: This command is similar to ASECURITY FTPD= except that opening a control session is permitted, but all commands that can be issued prior to user ID/password validation are rejected with “500 Command rejected.” The commands that are not allowed are USER, PASS, ACCT, QUIT, REIN, SYST, HELP, NOOP, PBSZ, PROT, and AUTH. If FTPC=YES, commands are processed normally.

Automatic Security can be used in conjunction with the existing DEFINE USER command to allow or prevent specific user access to data. To take advantage of this feature, you can specify a string of Y/N characters with the DATA= parameter on DEFINE USER to indicate allowed and forbidden actions based on the equates normally passed to the automatic security exit. See the section “Security Exit Block (SXBLOK),” page 149, for more information on these equates.

Here are three examples of user ID definitions:

- “Superman” can do anything:

```
DEFINE USER,ID=SUPERMAN,PASSWORD=LOIS123, -
DATA=YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY
```

- An FTP read-only user can be defined by permitting only the functions for SXTYPASS, SXTYREAD, SXTYCMD, SXTYOPDI, SXTYRDD, SXTYCWDL, SXTYCWD, and SXTYFCMD:

```
DEFINE USER,ID=FTPREADO,PASSWORD=READONLY, -
DATA=YYNNNNNNYNNNNNYYYNNNNNNNYNNNYNNNNNNNNNNN -
ROOT='/POWER/LST/A',FTP=YES
```

- By adding SXTYWRIT, SXTYDEL, SXTYREN, SXTYCRT, SXTYAPPE, SXTYMKD, and SXTYRMD functions, we have a user ID that can also write and “control” files and directories:

```
DEFINE USER,ID=FTPWRITE,PASSWORD=WRITETOO, -
DATA=YYYNNNNNYYYYNYYYYNNNNNNYYYNNYNNNNNNNNNNN -
ROOT='/HFS001/CSIVSEDW',FTP=YES
```

User-Created Security Manager

Implicitly Defining User IDs Defined In the User Security Manager

If you have a security exit in place, one of the parameters passed to the exit is SXPASSVF. This parameter indicates whether the password is valid according to information from DEFINE USER. If a user ID or password is invalid, the user security exit can still decide to accept it. In fact, you can include a user ID/password table in the user security manager and use this table in addition to (or instead of) the user IDs created with the explicit DEFINE USER command. The sample security manager SECEXIT.A implements this approach.

Note that this technique still does not protect user IDs and passwords from prying eyes. For example, someone could still dump out the SECEXIT phase and peruse your user ID/password table. Of course, depending on how complicated you wish to make your exit, you can define your user IDs and passwords in an external table, encrypt the passwords in the table, and load the table when the security exit is initialized. Following this methodology, your user IDs and passwords are less likely to be compromised.

Before leaving the subject of user IDs and passwords, we need to briefly touch on the DATA= parameter of the DEFINE USER command. The only function of the DATA= parameter is to take the specified string and pass it to the user security exit on behalf of the user. Therefore, the security exit can make decisions based on the contents of the DATA= parameter. The sample security exit SECEXIT.A implements a scheme in which a user is granted an increased level of access to data when the DATA= parameter contains the word PRIV. You can use the sample exit as a model for building your own security exit.

Network Security in a User Security Manager

The TCP/IP FOR VSE user security manager provides the capability of a security check based on a hardware address. The user security manager can examine the MAC address of each requester and decide whether to grant the request for access. You can use an exclusion list to prevent access by specific devices. You can use an inclusion list to allow for the possibility of someone connecting an unauthorized device to the cable.

The hardware addresses of many Ethernet cards can be customized. Therefore, network security alone is not an effective defense against hackers or unauthorized users.

Sample User Security Manager Code

Two sample security exits are included in the materials loaded during the installation process. The first is called SECEXIT.A and demonstrates some of the processing you can perform in a security exit. The second is called SECPOWER.A and demonstrates a security exit that is targeted at a very specific function.

General Coding Requirements

The requirements for coding a security exit are as follows:

- It must be written in assembler language
- It must be reentrant
- It is called in the TCP/IP FOR VSE partition
- Registers, upon entry, contain the following data:
  - Register 15: Entry point
  - Register 14: Return address
  - Register 13: Standard save area
  - Register 1: Address of the Security Exit Block (SXBLOK)
- Registers must be restored prior to return.
- Upon return, Register 15 contains the result code, as follows:
  - 0: The requested action is approved
  - 4: The requested action is denied

You must ensure that you release all dynamically acquired storage before you return control to TCP/IP FOR VSE.

Operation

The exit routine is called for each of the following three events:

- Immediately after the exit is loaded into storage and enabled
- Before each operation that is subject to security considerations
- Immediately before deactivation of the exit.

The first call is intended to permit the exit to prepare its environment. This includes connections to external security software and loading tables. For continuity, the exit is provided with two double words of storage (initially set to zero) that are passed to the exit on each subsequent call. The initialization call is synchronous. This means that no additional calls are made to the exit until it is fully initialized.

During its operational phase, the exit is called before each user request that may be subject to security concerns. The type of action being requested is passed to the exit.

If the exit is to be terminated (for example, before being refreshed), a special call is made to permit the exit to clean up its environment. This might include disconnecting from external security software, releasing storage, or other tasks.

To refresh the security exit, follow these steps:

1. Delete the existing exit with the SECURITY EXIT=OFF command.
2. Reassemble and relink the new version of the exit.
3. Define the new exit with the SECURITY PHASE=*nnn* XDATA=*xxx* EXIT=ON command.

Security Exit Block (SXBLOK)

The SXBLOK control block is created for each call to the security exit. It contains complete information about the requested operation. The user-written security exit can check fields in SXBLOK and make decisions based on their settings.

The following table contains a summary of the SXBLOK fields. Remember that returning a zero in register 15 indicates a pass condition at all times, while setting this register to a non-zero value causes the application to assume that the security call has failed (unless otherwise noted). For example, setting the value to 4 or 8 (or any other non-zero value) when validating a user for FTP access results in the FTP client rejecting the user request. In this case, the client would interpret the return code of 4 from the user exit as a rejection request.

Field Name	**Description**
SXID	This is a six-byte field that contains the value “SXBLOK”. If it does not contain this value, then storage has been overlaid or is corrupted. In that case, the security exit should output a warning message, set a bad return code, and exit.
SXVERS	This one-byte field is used for checking stack version compatibility. The exit can be coded to check for certain features based on the following equates: • SXVERS0: The exit is running on the earliest version of the stack that supports a security exit. • SXVERS1: The stack is a later version but is earlier than 1.5E. • SXVERS2: The stack is at version 1.5E or later. Other equates may be added in the future. The equate SXVERSC is always set to the most current version number. You can compare SXVERS to the equate SXVERSC; if it is lower, then it is an earlier version. If it is higher, then you should recommend that the exit be reassembled to be consistent with the rest of the stack.

Field Name	Description
SXTYPE	This one-byte field contains the equate number corresponding to the function requested by the exit. For example, if someone wants to issue a "CD" or a "CWD" command in a client such as FTP, EMAIL, or LPR, then SXTYPE would be set to "9" (the equate SXTYCWD). The valid equate numbers are listed below along with their names and meanings. Some equates are omitted because they either have not been implemented or have been made obsolete by other values. 1: SXTYPASS. A password has been provided by the user, and a request is being made to validate it. Check the settings in SXUSERID, SXPASSWD, and other fields to determine whether you want to permit access or not. 2: SXTYREAD. A request has been made to issue an OPEN for INPUT (READ) for a specific file. Check the values in SXUSERID, SXFNAME, and the file name parts in SXDPIE1 through SXSPIE22. 3: SXTYWRIT. A request has been made to issue an OPEN for OUTPUT (WRITE) for a specific file. Check the values in SXUSERID, SXFNAME, and the file name parts in SXDPIE1 through SXSPIE22. 5: SXTYSTRT. This is set if the security exit is being loaded for the first time. You may choose to load a table into storage or just output a startup message at this point. This occurs when the EXIT is set to ON while the automated exit process is set to OFF. 6: SXTYSHUT. This is the value set when the security exit is being unloaded. You may choose to free some storage allocated at startup time, simply output a message, or do nothing at all. This occurs when the EXIT is set to ON while the automated exit process is set to OFF. 9: SXTYCMD. This contains the FTP SITE command issued by the user. This call encompasses all FTP commands (SITE, CWD, etc.). 10: SXTYDEL. A request has been issued to delete a file from LIBR, POWER, BIM-EDIT, or VSAMCAT. If this file is not set to read-only, you may want to check SXUSERID as well as SXFNAME and the file name parts at SXDPIE1 through SXSPIE22. 11: SXTYREN. A request has been made to rename a LIBR or BIM-EDIT member. If this file is not set to read-only, you may want to check SXUSERID as well as SXFNAME and the file name parts at SXDPIE1 through SXSPIE22. 13: SXTYEXEC. A request has come from the FTPD to execute a REXX program from that daemon.

Field Name	Description
SXTYPE (continued)	14: SXTYAPPE. The FTPD wants to perform an APPEND operation against a file. If this file is not set to read-only, you may want to check SXUSERID as well as SXFNAME and the file name parts at SXDPIE1 through SXSPIE22. 18: SXTYSHEL. The REXECD or RSHD wants to execute a REXX program. 19: SXTYICMP. A remote site wants to ping your system. You can choose to reply. 20: SXTYLOGI. Someone wants to log into one of your servers such as FTPD. Check the USERID and PASSWORD and decide if you want to permit this login to proceed. 22: SXTYWEBL. A remote user has sent a request to the HTTPD. If you require them to login to the service, then a login menu is presented to the client and must be filled in before the request is serviced. 24: SXTYMKD. A request has been made to create a directory. If this file area is not set to read-only, you may want to check SXUSERID as well as SXFNAME and the file name parts at SXDPIE1 through SXSPIE22. 25: SXTYRMD. A request has been made to remove an existing directory. If this area is not set to read-only, you may want to check SXUSERID as well as SXFNAME and the file name parts at SXDPIE1 through SXSPIE22. 26: SXTYCWSL. A request has been made by FTPD to change working directories. The user ID, password, and desired positioning are passed to the user. 27: SXTYSTAU. This is set if the security exit is being loaded for the first time. You may choose to load a table into storage or just output a startup message at this point. This occurs when the automated exit process is set to ON. 28: SXTYSHAU. This is the value set when the security exit is being unloaded. You may choose to free some storage allocated at startup time, output a message, or do nothing at all. This occurs when the automated exit process is set to ON. 29: SXTYFCMD. This value is set when any FTP command is being processed. (This does not apply to the SITE command, which is handled by SXTYCMD.) 30: SXTYCGI. HTTPD is about to issue a CGI call from a specific user. You can confirm and accept or reject this request.
SXPASSVF	This one-byte field is set by the user when an SXTYPASS request occurs. If it is set to SXPASSGD, then your exit approved the password. If the field is set to SXPASSBD, then the password failed.

Field Name	Description
SXVALID1 (Note: SXVALID2 is not used.)	This one-byte field indicates the service type requesting a password check. The equates and the associated services are as follows: • SXV1FTP: FTP client or server • SXV1LPR: LPR client or server • SXV1WEB: HTTPD server • SXV1TEL: TELNETD • SXV1EMAI: Email client.
SXFNAME	This is the one- to seven-character DLBL of the file being processed.
SXDPIE1 − SXDPIE22 SXSPIE1 − SXSPIE22	These fields contain directory and file name information. There are two 22-field groups. Each group is eight bytes long. SXDPIE[1-22] contains the directory information, and SXSPIE[1-22] contains the file name parts. For example, “POWER.LST.A.TEST.00123.000” has “POWER” in SXDPIE1, “LST” in SXDPIE2, and “A” in SXDPIE3. “TEST” is in SXSPIE1, “00123” is in SXSPIE2, and “000” is in SXSPIE3. If the name piece is longer than eight bytes, the first byte has X'00', the next three bytes contain the name’s length, and the next four bytes contain a pointer to the name.
SXWORD	When the “SECURITY PHASE=, DATA=” command is issued, it is stored in a control block where the first double word contains phase information and the next 40 bytes contain the “DATA=” value. You can use this pointer to find the attributes of this general information.
SXUID SXGID SXFILUID SXFILGID	These fields contain values if UID/GID is enabled during the DEFINE NAME and DEFINE FILE processes. These UID and GID settings are provided for UNIX compatibility. If you choose not to define a “UID” or a “GID” in your DEFINE FILE or DEFINE NAME settings, then the associated fields are not set.
SXPERMIS	This one-byte flag indicates whether the DEFINE FILE had Group, User, or “other” read and/or write permissions enabled. This is provided for UNIX compatibility as part of DEFINE FILE and may or may not be set. It is useful when there is a READ/WRITE request on a file.
SXPATH	This 320-byte field contains the CWDL value when validation is requested to check whether a user can go to a particular area.
SXRETCD	This fullword field is set if a previous process that took control before your exit set this value.
SXLOIPAD	This field contains the local IP address.

Field Name	Description
SXLOPORT	This halfword field contains the local port ID of the server that is passing control to the security exit.
SXFLAG1	This one-byte field lets you enable certain options that are returned to the primary security system before returning control to the calling client or server. The primary (non-internal) item of interest here is SXF1NERR (the returned code is not to be seen as an error).
SXFROOT	This field is set if "ROOT=" is defined for the specific user.
SXPROCS	This field contains the identifier of the client or server type issuing the call. It is similar to SXVALID1, but it is for other than PASSWORD.

SXBLOK DSECT

A sample SXBLOK DSECT appears below.

```
SXID     DS       CL6'SXBLOK'     Eyecatcher
SXVERS   DS       X               Security Version Number
SXVERS0   EQU      0              - Version 0
SXVERS1   EQU      1              - Version 1
SXVERS2   EQU      2              - Version 2  (1.5 Level E)
SXVERSC   EQU      2              - Current version
*
SXTYPE   DS       X               Security Request Type
SXTYPASS  EQU      1              - Password Check
SXTYREAD  EQU      2              - Read Check
SXTYWRIT  EQU      3              - Write Check
SXTYUPDT  EQU      4              - Update Check
SXTYSTRT  EQU      5              - Startup Security
SXTYSHUT  EQU      6              - Shutdown Security
SXTYHARD  EQU      7              - Hardware Address Verify
SXTYIP    EQU      8              - IP Address Verify
SXTYCMD   EQU      9              - FTP SITE command check
SXTYDEL   EQU      10             - Delete check
SXTYREN   EQU      11             - Rename check
SXTYCRT   EQU      12             - Create check
SXTYEXEC  EQU      13             - EXEC command check
SXTYAPPE  EQU      14             - APPEND check
SXTYOPDI  EQU      15             - OPDIR check
SXTYRDD   EQU      16             - RDDIR check
SXTYCWD   EQU      17             - CWD Check
SXTYSHEL  EQU      18             - SHELL Check
SXTYICMP  EQU      19             - ICMP check
SXTYLOGI  EQU      20             - Daemon LOGIN request
SXTYRPC   EQU      21             - RPC Request
SXTYWEBL  EQU      22             - Web Logon Screen Request
*
*        The following types were added in Version 2
*
SXTYSCAN  EQU      23             - HTTPD SCANBLOCK request
SXTYMKD   EQU      24             - Make directory
SXTYRMD   EQU      25             - Remove directory
SXTYCWDL  EQU      26             - Last CWD
SXTYSTAU  EQU      27             - Startup-auto-exit
SXTYSHAU  EQU      28             - Shutdown-auto-exit
SXTYFCMD  EQU      29             - FTP command check
SXTYCGI   EQU      30             - CGI call via HTTP
*
SXUSERID DS       CL16            Userid
SXHARDAD  EQU      SXUSERID,6     Hardware Address (MAC)
SXPASSWD DS       CL16            Password
SXDATA   DS       CL40            User data (from DEFINE USER)
SXSECDAT DS       CL40            Data string from the SECURITY cmd
SXPROTO  DS       CL8             Protocol Name
SXIPADDR DS       F               IP Address
SXIPPORT DS       H               Port Number
SXFTYPE  DS       X               Type of file
```

```
SXFTDIRE  EQU      X'01'          - Directory
SXFTLIBR  EQU      X'02'          - Library
SXFTKSDS  EQU      X'03'          - KSDS VSAM File
SXFTRRDS  EQU      X'04'          - RRDS VSAM File
SXFTICCF  EQU      X'05'          - ICCF File
SXFTKDIR  EQU      X'06'          - KSDS VSAM Directory
SXFTPSEU  EQU      X'07'          - Pseudo Entry
SXFTPOWR  EQU      X'08'          - POWER Entry
SXFTTAPE  EQU      X'09'          - Tape Entry
SXFTBIME  EQU      X'0A'          - Bim-edit Entry
SXFTFALC  EQU      X'0B'          - Falcon Entry
SXFTCOND  EQU      X'0C'          - Condor Entry
SXFTVPDS  EQU      X'0D'          - VSEPDS Entry
SXFTCGI   EQU      X'0E'          - CGI Entry
SXFTVOLL  EQU      X'0F'          - Vollie Entry
SXFTCAF   EQU      X'10'          - CAF Entry
SXFTCGIB  EQU      X'11'          - CGI-BAL Entry
SXFTCGIL  EQU      X'12'          - CGI-COBOL Entry
SXFTCGIC  EQU      X'13'          - CGI-C Entry
SXFTCGIR  EQU      X'14'          - CGI-REXX Entry
SXFTCGIP  EQU      X'15'          - CGI-PLI Entry
SXFTCAFF  EQU      X'16'          - CAF-FILE Entry
SXFTCAFP  EQU      X'17'          - CAF-PROG Entry
SXFTESDS  EQU      X'C1'          - ESDS VSAM File
SXFTDIRU  EQU      X'C2'          - Directory for User driver
SXFTDIR   EQU      X'C4'          - Direct Access File
SXFTVCAT  EQU      X'C5'          - VSAM Catalog
SXFTVTOC  EQU      X'C6'          - VTOC Directory
SXFTTCUU  EQU      X'C7'          - Tape unit device
SXFTHFS   EQU      X'C8'          - HFS Entry (Added, Ver 2)
SXFTZIP   EQU      X'E9'          - Zip archive
SXFTGZIP  EQU      X'EA'          - GZip archive
SXFTDSPC  EQU      X'EB'          - Normal(non-zip/gz) dspace
SXFTSEQ   EQU      X'E2'          - Sequential File
*
SXPASSVF DS       X               Password verification status:
SXPASSGD  EQU      X'FF'          - Password Good
SXPASSBD  EQU      X'00'          - Password Failed
         DS       CL4             Reserved
SXFNAME  DS       CL8             DLBL of file
*
SXDPIE   DS       0CL176          Dataset name:
SXDPIE1  DS       CL8             Dataset Name Piece 1
SXDPIE2  DS       CL8             Dataset Name Piece 2
SXDPIE3  DS       CL8             Dataset Name Piece 3
SXDPIE4  DS       CL8             Dataset Name Piece 4
SXDPIE5  DS       CL8             Dataset Name Piece 5
SXDPIE6  DS       CL8             Dataset Name Piece 6
SXDPIE7  DS       CL8             Dataset Name Piece 7
SXDPIE8  DS       CL8             Dataset Name Piece 8
SXDPIE9  DS       CL8             Dataset Name Piece 9
SXDPIE10 DS       CL8             Dataset Name Piece 10
SXDPIE11 DS       CL8             Dataset Name Piece 11
SXDPIE12 DS       CL8             Dataset Name Piece 12
```

```
SXDPIE13 DS       CL8             Dataset Name Piece 13
SXDPIE14 DS       CL8             Dataset Name Piece 14
SXDPIE15 DS       CL8             Dataset Name Piece 15
SXDPIE16 DS       CL8             Dataset Name Piece 16
SXDPIE17 DS       CL8             Dataset Name Piece 17
SXDPIE18 DS       CL8             Dataset Name Piece 18
SXDPIE19 DS       CL8             Dataset Name Piece 19
SXDPIE20 DS       CL8             Dataset Name Piece 20
SXDPIE21 DS       CL8             Dataset Name Piece 21
SXDPIE22 DS       CL8             Dataset Name Piece 22
*
SXSPIE   DS       0CL176          SubDirectory Name:
SXSPIE1  DS       CL8             SubDirectory Name Piece 1
SXSPIE2  DS       CL8             SubDirectory Name Piece 2
SXSPIE3  DS       CL8             SubDirectory Name Piece 3
SXSPIE4  DS       CL8             SubDirectory Name Piece 4
SXSPIE5  DS       CL8             SubDirectory Name Piece 5
SXSPIE6  DS       CL8             SubDirectory Name Piece 6
SXSPIE7  DS       CL8             SubDirectory Name Piece 7
SXSPIE8  DS       CL8             SubDirectory Name Piece 8
SXSPIE9  DS       CL8             SubDirectory Name Piece 9
SXSPIE10 DS       CL8             SubDirectory Name Piece 10
SXSPIE11 DS       CL8             SubDirectory Name Piece 11
SXSPIE12 DS       CL8             SubDirectory Name Piece 12
SXSPIE13 DS       CL8             SubDirectory Name Piece 13
SXSPIE14 DS       CL8             SubDirectory Name Piece 14
SXSPIE15 DS       CL8             SubDirectory Name Piece 15
SXSPIE16 DS       CL8             SubDirectory Name Piece 16
SXSPIE17 DS       CL8             SubDirectory Name Piece 17
SXSPIE18 DS       CL8             SubDirectory Name Piece 18
SXSPIE19 DS       CL8             SubDirectory Name Piece 19
SXSPIE20 DS       CL8             SubDirectory Name Piece 20
SXSPIE21 DS       CL8             SubDirectory Name Piece 21
SXSPIE22 DS       CL8             SubDirectory Name Piece 22
*
*        Note: the SXCMDAD field is provided for compatibility
*              purposes only.  It should be replaced with a
*              direct reference to the SXSITCMD field.
*
SXCMDLEN EQU      SXDPIE1+0,4     Site Command length (fullword)
SXCMDAD  EQU      SXDPIE1+4,4     Addr of Site command text
SXSITCMD EQU      SXDPIE2,168     Site Command text
*
SXWORD   DS       A               Address of User Fullword
SXUID    DS       F               UID value of the user
SXGID    DS       F               GID value of the user
SXFILUID DS       F               UID value of the file
SXFILGID DS       F               GID value of the file
```

```
*
SXPERMIS DS       X               File Permissions flag
SXGWRITE  EQU      X'80'          - Group write on
SXGREAD   EQU      X'40'          - Group read on
SXUWRITE  EQU      X'20'          - User write on
SXUREAD   EQU      X'10'          - User read on
SXOWRITE  EQU      X'08'          - Other write on
SXOREAD   EQU      X'04'          - Other read on
*
SXPATH   EQU      *,320           Current path info
         DS       CL80            Original path field
*
*        End of Version 1 SXBLOK
*
         DS       CL240           SXPATH extension
SXRETCD  DS       F               Return code from previous routines
SXLOIPAD DS       F               Local IP addr
SXLOPORT DS       H               Local Port
*
SXHBLOCK DS       X               Return flags for IP blocking:
SXBLOKIP  EQU      X'01'          - Block IP from everything
SXBLOKLG  EQU      X'02'          - Add block to permanent log
SXBLOKWA  EQU      X'04'          - Add info to attack log
*
SXFLAG1  DS       X               Control flags:
SXF1DMP   EQU      X'80'          - Dump SXBLOK to log
SXF1LOG   EQU      X'40'          - Log SXBLOK to log
SXF1NERR  EQU      X'20'          - Return code is NOT an error
SXF1EXTR  EQU      X'08'          - This request from external partition
SXF1CNTL  EQU      X'04'          - "Control" request
SXF1AUTO  EQU      X'02'
SXF1EXIT  EQU      X'01'           Exit "control" request
*
SXFROOT  DS       CL64             FTP Root path
SX@MSGX  DS       A                Address of message writer
*
SXPROCS  DS       X                Process type:
*
SXPGEN   EQU      0                 - Generic process
SXPFTP   EQU      1                 - FTP
SXPLPR   EQU      2                 - LPR
SXPWEB   EQU      3                 - WEB
SXPTEL   EQU      4                 - Telnet
SXPSMTP  EQU      5                 - SMTP
SXPPOP3  EQU      6                 - POP3
SXPEMAIL EQU      7                 - EMAIL (client)
SXPLPD   EQU      8                 - LPD
*
SXMAILNM DS       CL17            POP3 lib.sublib name
*
*        End of Version 2 SXBLOK
*
SXBLOKLN EQU      *-SXBLOK,*-SXBLOK
```

Vender-Provided Exits

IBM

IBM provides a security exit. The functions and facilities of this security exit are explained in IBM manual *TCP/IP for VSE/ESA — IBM Program Setup and Supplementary Information* (SC33-6601-02). The security exit is called BSSTISX, and it can be used with the Basic Security Manager (BSM) in VSE. The BSSTISX exit provides the following basic functions:

- Validation of user IDs and passwords using information maintained in the VSE Interactive Interface (II) and stored in VSE.CONTROL.FILE.
- File Access Control, which limits file access to users defined as administrators. If you run your VSE system with SEC=YES, you can also have BSSTISX issue RACROUTE requests to determine whether a user is authorized to access a given file.
- VSE/POWER Access Control, which limits access to users defined as administrators or to users with user IDs that match those of the POWER queue entry.

If this security exit is used, it must be activated with the following command:

```
SECURTY ON EXIT=ON PHASE=BSSTISX
```

When using this exit, no user-created or other vendor-provided security exit can be used.

CSI International does not support BSSTISX. Refer questions about this exit and its use to an IBM representative.

Other Vendors

Other VSE system software vendors and consultants have created security exits that can be used with TCP/IP FOR VSE.

If this type of security exit is used, it must be activated using the following command:

```
SECURTY ON EXIT=ON PHASE=vendor-phase-name
```

No user-created exit or other vendor-provided security exit may be used.

Please contact the vendor supplying the exit for information on its implementation.

Security Changes in 1.5E

Significant changes were made to security processing and control in TCP/IP FOR VSE in release 1.5E. These changes are described here for those upgrading from any release prior to 1.5E. Although we have made every attempt to provide backward compatibility, we highly encourage you to examine the new commands and facilities and use them to provide a secure environment for your data processing.

Required Commands

All processes now run under user IDs and passwords, either explicitly or by default. If you make no other changes, you MUST provide the following commands in your initialization deck:

```
DEFINE USER,ID=$WEB,PASSWORD=$WEB,WEB=YES

DEFINE USER,ID=$LPR,PASSWORD=$LPR,LPR=YES

DEFINE USER,ID=$EVENT,PASSWORD=$EVENT,LPR=YES

DEFINE USER,ID=$LPD,PASSWORD=$LPD,LPD=YES

DEFINE USER,ID=$EMAIL,PASS=$EMAIL
```

Note that 1.5D user security exits see the same data in 1.5E and later releases and continue to run and provide the same level of security. Additional calls, with additional data, are made under the later releases. With additional coding, security exits can recognize and control a larger variety of requests.

Enhancements

Following is a list of security enhancements for 1.5E and later releases.

1. Logging: Results of security decisions can now be written to the log (routing code SECURITY). Available modes are All, Failed, and None.
2. All changes to security parameters are logged.
3. Security can be operated in Fail and Warn modes.
4. Overhead has been reduced.
5. “Automatic” security is now available for all files, based upon the values provided with DEFINE USER commands. See the section “Auto Security Manager” for more details.
6. Control and monitoring of security functions are consolidated in the SECURITY and the QUERY SECURITY commands.
7. The user-provided Security Exit may send messages to the security log using an address vector passed in the SXBLOK.
8. Security settings can be locked to prevent tampering.

9. FTPBATCH security no longer relies on loading the user exit into the FTPBATCH partition. This potential security exposure is eliminated by having FTPBATCH pass security calls to the target stack partition using the protected libraries and routines. Logging and control is handled by the stack routines automatically and by a stack-based user exit using the security settings set by the customer.

10. The installation-provided security exit may now contain identifying information that is verified when the phase is loaded.

11. User IDs can now be associated with specific uses. For example, having a valid ID for TN3270 access does not automatically permit FTP access.

12. Security requests passed to the user exit now contain the type of usage requested, for example, FTP or LPR.

13. The VSE/POWER user ID and password can be specified using "SET POWERUSERID=" and "SET POWERPASSWORD=", respectively. The default user ID remains SYSTCPIP, and the default password remains XL8'00'

14. Automation (event) processing now uses a default user ID and password. The $EVENT/$EVENT values may be overridden by DEFINE EVENT. The default ID and password are passed to client processes and are used for security calls unless overridden in a script.

15. LPR processing now sets a default user ID/password of $LPR/$LPR. These values are passed to security processing unless they are overridden explicitly by the user or by a script.

Equivalent Security Commands

Commands defined on pre-1.5E releases are still valid. Existing initialization decks and procedures continue to function as in the earlier releases. Pre-1.5E commands can be replaced, however, with the current SECURITY command, as shown in the following table.

Pre-1.5E Command	**Current Equivalent Command**
DEFINE SECURITY	SECURITY PHASE=*nnn* XDATA=*xxx* EXIT=ON
DELETE SECURITY	SECURITY EXIT=OFF
SECURITYARP	SECURITY ARP=
SET SECURITYARP=	SECURITY ARP=
SECURITYIP	SECURITY IP=
SET SECURITYIP=	SECURITY IP=
SECURITY ONX	SECURITY ON BATCH=ON

For example, the command sequence

```
DEFINE SECURITY,DRIVER=USEREX,DATA='ABCD'
SET SECURITY_ARP=ON
SET SECURITY_IP=ON
SET SECURITY ON
```

can be replaced with

```
SECURITY ON,PHASE=USEREX,XDATA='ABCD',ARP=ON,IP=ON, EXIT=ON
```

QUERY SECURITY Command

The following output is generated by the QUERY SECURITY command:

```
IPN253I << TCP/IP TCP/IP Security Settings >>
IPN750I    Security Processing: Disabled
IPN750I    ARP Checking:        Disabled
IPN750I    IP Address Checking: Disabled
IPN751I    Auto Data:           Undefined
IPN751I    Exit Data:           Undefined
IPN750I    Automatic Security:  Disabled
IPN750I    Security Exit:       Undefined
IPN750I    Batch Security:      Disabled
IPN752I    Security Mode: Fail Log: Fail Dump: Fail
```

This display, generated with both RESPONSE and SECURITY routings, summarizes all security information. The QUERY OPTIONS command does not show the redundant security information. The QUERY ALL command includes the security settings.

Note:

See the *TCP/IP FOR VSE Messages* manual for an explanation of the message fields.

Alternative Security Methods

NETWORK Security with IP Addresses

TCP/IP FOR VSE can control access to FTP and Telnet daemons on VSE by using the IPADDR parameter on the DEFINE FTPD and DEFINE TELNETD commands. The IPADDR parameter restricts the IP addresses that can connect with a given daemon. This parameter's use generally is associated with creating a pool of FTP or telnet daemons that service a subset of users, but it can also be used to prevent certain IP addresses from accessing certain applications. **IP addresses are completely under the control of the end user. Therefore, using IPADDR alone is not an effective defense against hackers or unauthorized users.**

Using Cryptography with SSL/TLS Protocols

You should also consider using cryptography to encrypt data being sent over the Internet when transmitting confidential information. The SSL/TLS protocol is implemented in the HTTPD, FTPD, and TelnetD daemons from CSI International. See the DEFINE commands for each of these daemons, and refer to the *TCP/IP FOR VSE Optional Features Guide* for detailed information on implementing and using secure communications (SSL/TLS).

10

Operation

Overview

For the most part, TCP/IP FOR VSE runs as a server task with no operations intervention. Occasionally, it requires the attention of an operator. This chapter discusses the following TCP/IP FOR VSE operations issues:

- Initialization
- Message management
- Command interfaces
- Shutdown processing
- Restart processing
- Waiting for TCP/IP to be active with the CHECKTCP utility
- TCP/IP event publishing

Initialization

TCP/IP FOR VSE must run in its own partition. The partition can be either static or dynamic. As you set up the TCP/IP FOR VSE partition, you need to answer the following questions:

- How big should I make the partition?
- What priority should I give the partition?

Partition Structure

The TCP/IP FOR VSE partition uses seven real VSE subtasks. The product contains a multi-tasking engine. The engine implements up to 65,536 pseudo tasks and does its own dispatching. These pseudo tasks do not count when you calculate the system task count.

The TCP/IP FOR VSE partition is generally all that is required to run TCP/IP. All daemons run in the partition. Client software, such as the FTP client, runs in the requester's partition and communicates cross-partition with TCP/IP FOR VSE. This is true when the client runs in a VSE batch job and also when the client runs as a CICS transaction. Socket applications from third-party vendors also run in their own partitions.

TCP/IP FOR VSE is a large product with a lot of options. You specify these options with a series of operator commands. These commands generally are stored in a configuration parameter file. This file, including a description of where TCP/IP FOR VSE finds the name of the file, is discussed later in this chapter.

Initialization JCL

To initialize TCP/IP FOR VSE, run the following job:

```
* $$ JOB JNM=TCPIP,CLASS=8,DISP=K
* $$ LST CLASS=A,DISP=D
// JOB TCPIP
// LIBDEF *,SEARCH=(PRD2.CONFIG,PRD2.TCPIP)
// EXEC PROC=DTRICCF
// SETPFIX LIMIT=400K
// EXEC IPNET,SIZE=IPNET,PARM='ID=00,INIT=IPINIT00'
/&
* $$ EOJ
```

As you tailor the TCP/IP FOR VSE startup JCL, you must consider the following issues:

- Search order. The search order for TCP/IP FOR VSE phases, library members, parameter files, and so forth is defined in the LIBDEF parameter. Be sure to specify LIBDEF * so that TCP/IP FOR VSE gets parameters from the libraries defined in the search order and not just phases.

Important:

We strongly recommend that you have one library for configuration data, including your TCP/IP FOR VSE authorization codes, exits, parameter files, library initialization members, and so on. We also recommend that you have a second library that contains text members and phases that you obtained from CSI International. Because our maintenance strategy is based on complete system replacement, it is easy for you to inadvertently overwrite your library initialization member and other important files simply by applying a TCP/IP FOR VSE service pack.

In the above examples, you should place your customized files in PRD2.CONFIG and reserve PRD2.TCPIP for library members and phases that we send you. During operation, you can use the QUERY PROGRAMS command to obtain a list of all loaded phases and the sublibrary from which each was loaded.

- // EXEC PROC=DTRICCF statement. This statement is required only if you plan to use FTP to transfer files out of ICCF.
- // SETPFIX LIMIT=400K statement. The value specified as the PFIX limit determines the amount of storage that can be page fixed by the program running in the partition. TCP/IP FOR VSE requires that storage for IO operations be page fixed. If you do not allocate enough page-fixed storage, TCP/IP FOR VSE cannot run and fails. Because the value specified by // SETPFIX is a limit and not an absolute value, you have nothing to lose by specifying a value larger then TCP/IP FOR VSE could ever need. For this reason, we recommend a value of 400K. Changing the value of the page fix limit does not help to solve most TCP/IP FOR VSE storage-related problems.
- DLBL and EXTENT cards. You may choose to include DLBL and EXTENT cards in the TCP/IP FOR VSE startup JCL. Once you do this, you can make the associated files accessible to the TCP/IP FOR VSE file system using the DEFINE FILE command.
- // EXEC IPNET. This statement initializes the TCP/IP FOR VSE partition. You can specify optional parameters in the PARM field of the // EXEC IPNET card. The parameters you can use are described in the table below.

// EXEC IPNET Parameters

Parameter	Description
ID	Enables you to specify a two-digit identifier for this copy of TCP/IP FOR VSE. If you run multiple copies of TCP/IP FOR VSE in the same VSE image, you must assign a unique two-digit numeric code to each. This code permits the TCP/IP FOR VSE partitions to communicate with each other. The default is 00. If you are running only one copy of TCP/IP FOR VSE, we highly recommend that you assign a SYSID of 00 to it. This is because, by default, all client software provided with TCP/IP FOR VSE attempts to establish communication with the SYSID 00 TCP/IP FOR VSE. If you assign a different value, you need to specify an ID= parameter every time you use a VSE client.
INIT	Enables you to specify the member name to be used as the initial source of TCP/IP FOR VSE's initialization parameters. The coded one- to eight-character member name is searched for as a '.L' book. The default is IPINIT*xx*.L, where *xx* is the two-digit system identifier from the ID= parameter (or its default). For example, if your TCP/IP FOR VSE SYSID is 00, TCP/IP FOR VSE pulls its parameters from IPINIT00.L unless you specify the INIT= parameter.
UPCASE	Translates all console messages to upper case. TCP/IP FOR VSE translates some initialization console messages to upper case, even when this parameter is not specified, because some VSE consoles cannot display lower-case characters.

Message Management

TCP/IP FOR VSE uses the VSE console as its command and control device and, therefore, sends a large number of messages to the console and SYSLST.

Message Routing

Each message is assigned an importance level or type shown in the table below. A single-letter routing code that normally corresponds to this message type is appended to the message identifier. Routing code examples include ‘I’, ‘E’, and ‘D’. For example, the diagnostic message “IPN123D” is assigned the ‘D’ routing code.

Each message’s importance level or type is specified in the *TCP/IP FOR VSE Messages* manual by the first word in the message description, for example, “(Diagnostic).”

Message Type	Description
Critical	Messages that contain information of the highest importance. These messages require immediate attention.
Vital	Slightly less important than “Critical”
Important	Messages that convey information that should be acted upon, but do not require immediate action
Informational	Messages containing general status information
Warning	Messages intended to alert you to an unusual but foreseen situation
Response	The response to an operator command (or command entered by a script or the initialization deck)
Diagnostic	Diagnostic messages, which are usually enabled and disabled by the DIAGNOSE command
Security	Messages that have security implications

Message Logging

Messages are written to the console and to one or more log files. Additional log files can be created by the DEFINE LOG command. Existing logs (and the console) can be controlled by using the MODIFY LOG command. These commands are described below.

Note:

Starting in 1.5F, the MODIFY LOG command has replaced the SET MESSAGE command. For backward compatibility, TCP/IP FOR VSE still supports SET MESSAGE.

DEFINE LOG

Use this command to create a new log file. It has the following syntax:

```
DEFine LOG ID=id,TYPE=PRINTER,LOGICALUnit=lunit
   [,LINELength=num ] [,TIMEstamp=Left|Right|None] [,levels]
```

The arguments are as follows:

Parameter	**Description**
ID	A unique name to identify the log. The recommended value is the same as "LOGICALUNIT".
LINELength	The number of displayable positions on each line. Range is 40 to 132. The default is based on the device. If a line exceeds this length, it is broken at a blank and a continuation character (>) is appended.
LOGICALUnit	The logical device to be used, for example, "SYS021". Can also be specified as "LU".
TIMEstamp	Timestamp placement with respect to each line. Values are "Left" (the default), "Right", and "None."
TYPE	The type of log. "PRINTER" is the only valid value.
levels	Optional list of message levels/types to be added to or subtracted from (with a "-NO" prefix) the types to be logged. The default is to write all messages to the log. The *levels* list is processed from left to right. For example, "ALL,-NODIAG" specifies that all messages except diagnostic messages are written to this log. (This is the default for messages written to the console.) The valid values are as follows: CRITical VITAL WARNing IMPORTant INFOrmational RESPonse DIAGnose SECurity ALL ("-NO" does not apply) NONE ("-NO" does not apply)

MODIFY LOG

Use this command to change an existing log file. The syntax is

```
MODify LOG,ID=id [,LINELength=num]
           [,TIMEstamp=Left|Right|None][,levels]
```

The arguments are described in the table below.

Parameter	Description
ID	The name of the log to change. Specifying “CONSOLE” changes the console logging.
LINELength	The number of displayable positions on each line. The range is 4 to 132 (no effect for CONSOLE). When a line exceeds this length, it is broken at a blank and a continuation character (>) is appended.
TIMEstamp	The timestamp placement. Values are LEFT, RIGHT, and NONE (no effect for CONSOLE).
levels	See the *levels* description for DEFINE LOG.

Note:

To suppress console traffic by message number, see the section “Message Suppression” below.

QUERY LOGS

Use this command to determine the current log status.

Message-Case Translation

TCP/IP FOR VSE can translate all messages to upper case before they are displayed. If you want all messages to display in upper case, issue the following command:

```
UPCASE ON
```

You can issue this command in the TCP/IP FOR VSE initialization deck or from the VSE console. To return to mixed case console output, you can issue the following command:

```
UPCASE OFF
```

Message Suppression

TCP/IP FOR VSE contains a message-suppression facility that allows you to suppress individual messages from the VSE system console. To suppress a message, issue the MESSAGE command in one of two ways:

```
MESsage MSGID=message-identifier,CONSOLE=N
 -or-
MSG MSGID=message-identifier,CONSOLE=N
```

The *message-identifier* is the ID of the message you want to suppress, such as “ABC123.” Omit the message’s importance indictor (routing code), such as I, E, or D, at the end of the message identifier.

Like most TCP/IP FOR VSE commands, you can issue this command in the TCP/IP FOR VSE initialization deck or through the console. See the next section, “Command Interface,” for details on using the system console.

Note:

Do not confuse the MESSAGE command synonym “MSG” with the IBM command “MSG *partition-id*”.

To reverse the effect of the above command statement, you can issue the following command:

```
MESsage MSGID=message-identifier,CONSOLE=Y
```

See the *TCP/IP FOR VSE Command Reference* for information on other keyword parameters you can use with the MESSAGE command.

Command Interface

The TCP/IP FOR VSE console interface allows you to issue commands to the TCP/IP partition in one of two ways:

- Through the VSE console
- Through the IPNETCMD batch program.

TCP/IP FOR VSE contains dozens of commands that enable you to display, define, start, and stop most aspects of your TCP/IP environment. These commands are documented in the *TCP/IP FOR VSE Command Reference*. Most commands that can be entered in the TCP/IP FOR VSE initialization job can also be entered at the VSE console. We discuss each of these mechanisms in the following sections.

Issuing Commands through the VSE Console

You can issue TCP/IP FOR VSE commands on the VSE system console with a prompt or with the DATA= parameter of the VSE attention routine (AR) MSG command.

Using a TCP/IP Prompt

To issue a command from a prompt, first enter the following VSE AR command at the VSE system prompt.

```
MSG xx
```

where *xx* is the ID of the partition in which VSE is running.

The MSG command produces the following TCP/IP FOR VSE command prompt:

```
yy-zzzz IPN300A Enter TCP/IP Command
```

where

- *yy* is the partition ID
- *zzzz* is the reply number that you use to enter your command

Once the TCP/IP FOR VSE command prompt appears, you can issue TCP/IP FOR VSE commands through the console by replying to the message.

For example, assume that the following message is displayed on your console:

```
F8-0090 IPN300A Enter TCP/IP Command
```

You can enter the following reply to display the status of telnet sessions:

```
90 Q TELNETDS
```

To eliminate the TCP/IP FOR VSE command prompt, reply to the message with a null string. You can also prevent your operators from eliminating the command prompt. To maintain the command prompt even when the null string is entered, issue the following command:

```
CONSOLE_HOLD ON
```

Using MSG with the DATA= Parameter

To issue a TCP/IP command from the console with the DATA= parameter, enter the following VSE AR command statement:

```
MSG xx,DATA=tcp-ip_command
```

where *xx* is the ID of the partition in which VSE is running.

This method can be used only if no outstanding console read is active on the VSE system console for the TCP/IP partition. To suppress a currently active console read, you can issue:

```
CONSOLE_HOLD OFF
```

A “IPN482I TCP/IP for VSE reply ID will not be maintained” message should then be displayed. After replying with EOB to this message, you should then be able to issue the “MSG *xx*,DATA= ” command.

For example, you can enter the following command to display the status of FTP sessions (internal FTP daemons) in the TCP/IP FOR VSE partition:

```
MSG xx,DATA=Q FTPD
```

Suppressing Entered Strings

In general, TCP/IP FOR VSE commands that you enter on the console are displayed on SYSLST. You can suppress the display for sensitive commands by beginning the command with a plus sign (+).

The example below shows how to suppress the PASSWORD command from the SYSLST display. Note that sensitive material entered by the operator can still appear on the console and in SYSLOG.

This display and logging is controlled by VSE and cannot be prevented by TCP/IP FOR VSE.

```
DEFINE USER,ID=JOHN, -
+PASSWORD=DECAF
```

Issuing Commands through IPNETCMD

TCP/IP FOR VSE allows a user to issue commands to the TCP/IP FOR VSE partition through a batch job named IPNETCMD. To run IPNETCMD, use the following JCL:

```
* $$ JOB JNM=IPNETCMD,CLASS=0,DISP=D
* $$ LST CLASS=A
//  JOB IPNETCMD
// LIBDEF *,SEARCH=(PRD2.TCPIP)
// EXEC IPNETCMD,PARM='ID=xx'
 TCP/IP for VSE command 1
 TCP/IP for VSE command 2
/*
/&
* $$ EOJ
```

The only parameter you can pass to the IPNETCMD facility is ID=*xx*, where *xx* is the identifier of the TCP/IP FOR VSE partition that you are sending the commands to.

If you use IPNETCMD, note the following:

- If you have enabled command security with the SET PASSWORD command in the initialization deck, the TCP/IP FOR VSE password must be the first command in the job stream.
- All output from your TCP/IP FOR VSE commands is sent to the console.
- The IPNETCMD command processor attempts to issue ALL commands in the input stream. It does not stop if it encounters an invalid command.
- If DOWNCHECK is set to ON (meaning that you want to be prompted for confirmation after a SHUTDOWN command is issued), you cannot issue SHUTDOWN from IPNETCMD.

The return codes from the IPNETCMD batch job are as follows:

Return Code	Description
0	The IPNETCMD job was successful, and all commands entered in the input stream were processed.
8	The IPNETCMD job had at least one command that was not processed successfully. Consult the listing for the job to determine which command failed. You also receive a return code of 8 if the ID= parameter in the EXEC statement has a syntax error. Finally, you receive this return code if the TCP/IP FOR VSE system identifier specified in the ID= parameter is inactive.

Return Code	Description
16	The IPNETCMD command processor encountered a severe internal error. Please contact CSI Technical Support for help.
24	The IPNETCMD command processor encountered a severe internal error while attempting to load an internal task. Check to see if you are running the program in a large enough partition (greater than 2 MB). If you are, call CSI Technical Support for help.

Shutdown Processing

TCP/IP FOR VSE should run until you specifically request termination. You can terminate TCP/IP FOR VSE or application processing using one of the following methods:

- Application (process) shutdown
- Normal shutdown
- Cancel
- Cancel with force

Application Shutdown

To terminate application processing gracefully, issue the QUIESCE ON command. This command causes TCP/IP FOR VSE to reject socket OPEN requests from applications and connection requests from external hosts. Processing of established connections continues uninterrupted. This permits work in progress to complete while new tasks are prevented.

To cancel the QUIESCE ON command, issue QUIESCE OFF. When the OFF option is used, socket OPEN requests and connection requests are honored once again, and normal processing is resumed.

When QUIESCE ON is issued, TCP/IP FOR VSE responds as follows:

- Applications issuing socket OPEN requests are notified of a system shutdown, and incoming connection requests are rejected with a RESET. Depending on the host and the application, it may not be possible to resume processing.
- A periodic console display shows the progress of terminating processes. This display includes a count of active connections by local port number. These messages are shown in the following example:

```
F4 0099 T101: IPN225I TCP/IP quiescing; connection requests are rejected
...
F4 0099 000D: IPI515I TCP/IP Stack QUIESCE progress
F4 0099 000D: IPI516I      1 connections active on port    21
F4 0099 000D: IPI516I      5 connections active on port    23
...
F4 0099 000D: IPI517I TCP/IP processing has been quiesced
```

The messages are displayed periodically until you issue QUIESCE OFF or TCP/IP FOR VSE shuts down.

Normal Shutdown

To terminate TCP/IP FOR VSE normally, issue the SHUTDOWN command as a TCP/IP FOR VSE operator command. The SHUTDOWN process is controlled in that TCP/IP FOR VSE stops all TCP/IP-related activities, link drivers, and processes before terminating. During the shutdown process, messages indicate progress.

The TCP/IP partition should end with a return code of zero. Non-zero return codes are used when the stack cannot issue messages to explain the problem. These codes are issued when stack initialization fails.

The shutdown return codes are explained in the following table:

Return Code	Description
0	Normal completion
4	Unable to locate the messages skeleton member. This source book (MESSAGES.M) is required. If it cannot be located, IPNET terminates without issuing any messages. Check the LIBDEF statement and make sure that the member is in the search chain.
8	A GETSL macro failed while reading the message skeletons. Ensure that the correct version of the MESSAGES.M member is first in the search chain. Check the library for damage. If necessary, reinstall TCP/IP FOR VSE and the latest service pack.
12	Critical GETVIS shortage. Insufficient partition GETVIS area is available to load the message skeletons. Increase the partition size by at least 6 MB.

There are two points to keep in mind regarding shutdown processing:

- TCP/IP FOR VSE prompts you to ensure that you really want to shut down the partition. The message you receive is:

  ```
  IPN205A Are you sure you want to SHUTDOWN YES or NO
  ```

  You can suppress this message with the following command:

  ```
  DOWNCHECK OFF
  ```

- If you run VSE in a virtual machine, make sure that you do not inadvertently issue the SHUTDOWN command to CP instead of to TCP/IP FOR VSE.

Cancel

If the TCP/IP FOR VSE partition becomes non-responsive, you may need to cancel it. To do this, issue the following command:

```
CANCEL xx
```

Variable *xx* is the ID of the partition in which TCP/IP FOR VSE is running. TCP/IP FOR VSE responds by going through a controlled shutdown and terminating normally.

Cancel with Force

If the TCP/IP FOR VSE partition becomes non-responsive and the CANCEL command has no effect, you may need to cancel with force. To do this, issue the following command:

```
CANCEL xx,FORCE
```

Variable *xx* is the ID of the partition in which TCP/IP FOR VSE is running. TCP/IP FOR VSE ends immediately. The TCP/IP FOR VSE error exits do not get control, however, and the product may not terminate cleanly. You should use the FORCE parameter only as a last resort because the results are unpredictable.

Restart Processing

If TCP/IP FOR VSE ends without going through its normal shutdown process, it probably did not have a chance to fully clean up its environment. This can happen if the operator cancels the partition with the FORCE parameter or if the partition abends. If this occurs, you may have problems the next time you start TCP/IP FOR VSE. Sometimes the product initializes and then detects that it was not shut down cleanly. TCP/IP FOR VSE then runs a controlled shutdown and termination. At the next restart, TCP/IP FOR VSE should initialize normally.

The following console listing shows the messages that TCP/IP FOR VSE issues during a restart:

```
F4 0099 IPN100I TCP/IP VERSION 02.01.xx 03/03/16 22.56, INITIALIZING
F4 0099 IPN102I COPYRIGHT 1995,20xx (C) CONNECTIVITY SYSTEMS INC.
F4 0099 IPN209I Service Pack xxxx (APAR PK33472) has been applied. Pack >
F4 0099 status is GA
F4 0099 IPN119E Socket queue previously allocated to this partition
F4 0099 IPN146I TCP/IP Beginning Shutdown
F4 0099 IPN597I Shutdown Stage: 4:  Network termination (max. 10 seconds)
F4 0099 IPN597I Shutdown Stage: 11:  Subtask Terminations
F4 0099 IPN597I Shutdown Stage: 12:  Operating System Interface Removal
F4 0099 IPN597I Shutdown Stage: 13:  Printing Statistics
F4 0099 IPN597I Shutdown Stage: 14:  Terminating Console Logging
F4 0004 EOJ PROD      MAX.RETURN CODE=0000
```

This is a standard process that occurs whenever TCP/IP FOR VSE does not terminate normally. For this process to work, TCP/IP FOR VSE must restart in the same partition in which it ran before. If you try to start TCP/IP FOR VSE in a different partition, it produces an error message and shuts down. This is normal. To recover TCP/IP FOR VSE, you must restart it in the same partition.

CHECKTCP Utility

Some partitions depend on TCP/IP FOR VSE. You must verify that TCP/IP FOR VSE is active before these partitions start. For example, if you use the CICS web interface, you want to delay CICS initialization until after TCP/IP FOR VSE starts. If a socket application relies on TCP/IP FOR VSE, you need to ensure that it does not start before TCP/IP FOR VSE is active.

The CHECKTCP utility helps you accomplish these objectives. You can run CHECKTCP in any job stream that requires the use of TCP/IP FOR VSE. CHECKTCP can wait for TCP/IP FOR VSE to become active (thus delaying the entire job stream), or it can immediately terminate so that you can take action based on a VSE conditional JCL.

The following sample job stream uses CHECKTCP:

```
* $$ JOB JNM=CHECKTCP,CLASS=Y,DISP=D
* $$ LST CLASS=A,DEST=(*,MSCHARE)
// JOB CHECKTCP
// LIBDEF *,SEARCH=PRD2.TCPIP
// EXEC CHECKTCP,SIZE=CHECKTCP,PARM='SYSID=00,WAIT=Y'
/*
/&
* $$ EOJ
```

The CHECKTCP parameters are as follows. Some parameters are not set to a value.

Parameter	**Description**
DEBUG	Enables additional debugging messages
QUIET	Suppresses output to SYSLOG
SYSID=*nn*	Specifies the 2-byte SYSID of the TCP/IP FOR VSE partition that you need to be active. The default is 00.
UPPER	Forces messages to be issued in upper case
WAIT=[Y | N]	Specifies whether CHECKTCP should wait for the stack to become active. If WAIT=Y, CHECKTCP keeps waiting until the stack is ready for SOCKET applications to run. The only way to terminate the wait is to issue a MSG *xx* command to the partition where CHECKTCP is waiting. MSG terminates CHECKTCP with a return code of 12. If WAIT=N, CHECKTCP terminates immediately with RC=4 (stack is not available) or RC=0 (stack is available). The default is N.

The CHECKTCP return codes are as follows:

Return Code	Description
0	The TCP/IP FOR VSE partition specified by the SYSID is active.
4	The TCP/IP FOR VSE partition specified by the SYSID is inactive. It shut down normally in its last run, or it was never active.
8	The TCP/IP FOR VSE partition specified in the SYSID parameter is inactive. It terminated abnormally in its last run and is therefore awaiting restart and recovery processing.
12	The operator entered the command MSG *xx* in the CHECKTCP partition, thus terminating the wait.
16	The TCP/IP FOR VSE partition specified in the SYSID parameter is active but is not responding to socket requests.

Event Publisher

The TCP/IP FOR VSE Event Publication Facility provides simplified interfacing between TCP/IP-related events and automation processing.

Events are published by the PUBLISH macro. This macro, called from inside or outside the TCP/IP partition, uses the standard SOBLOK method to queue an event notification for publication. The target TCP/IP partition processes each of the queued notifications and directs these notifications to other processes. The processes include

- IMODs running under CSI-FAQS/ASO
- On- and off-platform processes with a socket connection to a Publisher daemon.

A unique item number identifies each published item. The item dictionary contains each assigned item number along with its format.

Item Format

Items consist of an item header followed by a string of elements. Each element consists of a

- Length field
- Type field
- Data field

The length field is a halfword containing the offset to the next element, not counting the length field. For example, a length field of x'0000' indicates a null element, with the next element length field immediately following.

The element number is the second field in each element and is a halfword value that uniquely identifies the element.

The last field contains the element's value. This is entirely dependent on the element number.

Processing Rules

The following processing rules apply:

- All items are optional.
- Once an element is assigned a value/format, it will not change.
- No item or element may be used unless it appears in the dictionary.
- Items and elements may be used only as they appear in the dictionary.
- Any code that processes "published" data MUST ignore or accept without error all items and elements as they are defined.

Item Header

Each published item begins with a fixed-format header block containing the following fields:

Field	Format	Description
Length	Unsigned halfword	Length of entire item, not counting the Length field
Version	Unsigned byte	Format version of this item, currently X'01'
	1 byte	Reserved
	1 fullword	Reserved
Time	TOD clock value (DBL word)	TOD value at time of publication (GMT)
Home IP	IPv6 address (16 bytes)	IP address of the stack publishing the item
Offset	Signed fullword	Local offset from GMT for TOD value
Class	Classification flags (16 bits)	
	2 bytes	Reserved
Item	Unsigned fullword	The item number

Item Dictionary

In the following table, elements not included in brackets are always present. Those shown within brackets are optional. Elements shown within the same set of brackets will either be all present or all absent. Elements not specified in the table must be parsed and ignored. The "Status" column indicates the item's current status: "D" indicates draft, "P" indicates provisional, and "S" indicates standard.

Item	Name	Elements	Status
1	FTP File Received	1, 2, 4, 6, 7 [,8] [,9 ,10]	D
2	FTP File Sent	1, 2 [,3], 4, 6, 7 [,8] [,9 ,10]	D
3	FTP File Sent	4, 7, 9, 10, 11, 12	D

Element Dictionary

In the following table, the "Status" column indicates the element's current status: "D" indicates draft, "P" indicates provisional, and "S" indicates standard.

Num	Name	Elements	Status
1	Foreign IP address	Binary, 1 to 16 bytes, lead zeros optional	D
2	Local File Name	EBCDIC, 2 to *n* bytes. Byte one contains the separator character (\ / .), followed by the name.	D
3	Foreign File Name	EBCDIC, 2 to *n* bytes (see element 2)	D
4	Completion and reason codes	Binary, 4 bytes (2 bytes each)	D
5	Record count	Binary, 1 to 8 bytes	D
6	Byte count	Binary, 1 to 8 bytes	D
7	Duration	Binary, 1 to 8 bytes (lead zeros optional) TOD format	D
8	User name	EBCDIC, 1 to *n* bytes.	D
9	Job name	EBCDIC, 1 to *n* bytes	D
10	Job number	EBCDIC, 1 to *n* bytes	D
11	Phase name	EBCDIC, Values: "FTPBATCH", "CLIENT", others	D
12	User-specified token	EBCDIC, 1 to *n* bytes	D

Executing IMODs

You can cause selected publishing events to trigger CSI-FAQS/ASO IMODs. The IMOD can then parse the elements and perform appropriate automation tasks.

To activate IMOD triggering, use the following TCP/IP FOR VSE command:

```
DEFINE PUBLISHER,ID=id,IMODLIST=member
```

Where *member* is a library "L" book with the following format:

```
001 IMOD $FTPFRCV /*         FTP FILE RECEIVED
002 IMOD $FTPFSNT /*         FTP FILE SENT
```

The first field is the item number; the second is the keyword “IMOD”; and the third is the IMOD name. The remainder of the record is treated as a comment. Lines that begin with an asterisk (*) are treated as comments.

Related Commands

See the *TCP/IP FOR VSE Command Reference* for information on the following commands:

Command	Description
DEFINE PUBLISHER	Defines a Publisher daemon
DELETE PUBLISHER	Terminates a Publisher daemon
QUERY PUBLISHER	Displays the status of the Publisher daemon

11

ASCII-to-EBCDIC Translation

Overview

TCP/IP FOR VSE has to work with three distinct file types. They are

- EBCDIC
- ASCII
- Binary, also referred to as image or image mode.

When working with EBCDIC and ASCII files, each byte's absolute value is interpreted as a particular symbol or usage. To interpret each byte correctly, TCP/IP FOR VSE consults a map. For example, in an EBCDIC-based system, the hex value C1 is interpreted as the letter "A."

Binary files are never translated. For binary files, the values of the individual bytes are absolute and have the same meaning regardless of the platform.

The map between a hex value and a symbol is referred to as a *code page*. Code pages are standardized mappings and are defined for a large number of special uses, especially to accommodate symbols that are specific to a nationality. The code pages used by TCP/IP FOR VSE are the ones defined by IBM. Code pages are not code; they merely document that a certain EBCDIC value or a certain ASCII value represents a known symbol, such as the letter "A," a US "$," or the European Euro symbol.

EBCDIC

EBCDIC, the Extended Binary Code for Data Inter-Change, is an eight-bit code developed by IBM for use on their computers. Its 256 characters have a one-to-one correspondence with each possible byte value used in the IBM architecture.

ASCII

ASCII, the American Standard Code for Information Interchange, is a long-standing code that is used by most non-IBM mainframe computers. Although ASCII is a seven-bit code with 128 characters, a non-official extended ASCII code is in general use. The extended ASCII code contains an additional 128 characters.

Translation

When data is transmitted between hosts that operate with different codes, translation must occur. For example, FTP sessions must translate not only the files being transmitted, but also the control information (commands and responses).

Problems

Translations between EBCDIC and ASCII code pages can be tricky. There are a variety of problems that can occur, including the following:

- TCP/IP's default code is ASCII. This means that TCP/IP translates all data to ASCII before it ships it, unless you specifically tell it to translate the data in a different manner. If you FTP a file from one TCP/IP FOR VSE site to another TCP/IP FOR VSE site without specific translation instructions, the file is translated to ASCII, transmitted, and retranslated to EBCDIC. If both ends of the connection do not use the same reversible translation tables, then the received dataset is not a mirror image of the transmitted file. You can solve this problem by telling TCP/IP how to translate the data. In this case, for example, you would tell TCP/IP to use EBCDIC for translation. Note that control information and commands are always translated to ASCII for transmission.

- UNIX clients assume that ASCII and binary are the same. This is because the UNIX people got there first. If IBM had developed TCP/IP, things would be different. EBCDIC and binary would be one and the same. Unfortunately (for us) they didn't and they aren't. This means that TCP/IP FOR VSE sometimes must make adjustments. When a UNIX-based client requests a binary mode transmission, we must decide if it should be real binary or ASCII-binary. We generally make this determination based on the file name extension. For more information on file name extensions, see the description of the EXTTYPES.L parameter file in chapter 6, "Configuring FTP Daemons."

- Some records contain both text and binary (numeric) data. This problem is beyond our control. When you need to transmit this type of file, you need to make your own arrangements.

- Different systems use different record formats. Under VSE, records exist as fixed length, variable length, and string. There are no delimiters. All file types can contain text or binary data. When shipped through TCP/IP, however, there are no fixed-length records and, in general, the remote machine probably won't support fixed-length records.

Also, EBCDIC records always end with a new line (NL) character (X'15'), and ASCII records end with a carriage return line feed pair (X'0D0A'). These characters cannot be embedded in the records.

Binary files are treated as a single string of bytes, and such files have no record structure. Thus, the choice of translation affects the record format of the file. In practice, you can use FTP SITE commands to help recreate the desired record formats.

Translation Options

TCP/IP FOR VSE provides several options for translating between ASCII and EBCDIC. All options use standard translation tables. But, the choice of table for each translation can be made in several ways.

Control Operations

An internal translation table with the name SYSTEM translates commands, command responses, and status information processed by TCP/IP FOR VSE clients and daemons. You have no control over the definition and use of the SYSTEM table.

FTP

Following is the order of precedence for specifying the translation table used for FTP file transmissions. The lowest priority is listed first. As you proceed down the list, each level overrides the previous one.

- The translation table set as the system default table (set by the DEFINE TRANSLATION command)
- SITE command
- TRANSLATE= parameter of the DEFINE FTPD command that started the FTP daemon being used
- TRANSLATE= parameter of the DEFINE FILE command of the file that is being transferred. This means that if you specify a TRANSLATE parameter on the DEFINE FILE command, it cannot be overridden when the user tries to transfer the file.

HTTP

Following is the order of precedence for specifying the translation table used for text files sent to web browsers. The lowest priority is listed first. As you proceed down the list, each level overrides the previous one.

- The translation table set as the system default table (set by the DEFINE TRANSLATION command)
- The TRANSLATE= parameter of the DEFINE HTTPD command
- The TRANSLATE= parameter of the DEFINE FILE command that defines the file from which the page is being fetched.

LPR

Following is the order of precedence for specifying the translation table used for files processed by LPR (and Auto-LPR). The lowest priority is listed first. As you proceed down the list, each level overrides the previous one.

- The translation table set as the system default table (set by the DEFINE TRANSLATION command).
- The TRANSLATE= parameter of the DEFINE FILE command from which the source is taken. For Auto-LPR, this is the file whose public name is POWER.

- The LPR SET TRANSLATE command.

LPD

Following is the order of precedence for specifying the translation table used for files processed by the LPD daemon. The lowest priority is listed first. As you proceed down the list, each level overrides the previous one.

- The translation table set as the system default table (set by the DEFINE TRANSLATION command).
- The TRANSLATE= parameter of the DEFINE LPD command.
- The TRANSLATE= parameter of the DEFINE FILE command that defines the file where the listing is to be placed. This includes the POWER file.

Telnet

TN3270 data streams are not translated because TN3270 clients operate in EBCDIC mode. If characters do not display as desired, check the client software's documentation for information about selecting a different code page.

Line-mode telnet sessions established with external daemons by TCP/IP FOR VSE's CICS-based and batch clients are translated using an internally defined table. You can override the name of this translate table by using the SET TELNET_TRANSLATE console command on TCP/IP FOR VSE.

Translation

TCP/IP FOR VSE accepts standard translation and null translation, which are explained in this section.

Standard Translation

TCP/IP FOR VSE ships with standard translation tables and code pages. There are two types of translation tables, as follows:

- Single byte, sometimes referred to as SBCS. The member IPXLATE.L contains a large number of single-byte translation tables.
- Double byte, sometimes referred to as DBCS. For double-byte translation, we provide code pages in CHINA.L, JAPAN.L, and KOREA.L. Because of their size, the DBCS code page tables are not included in the installation job stream. You can obtain them from CSI International's FTP server, the CSI web page www.csi-international.com/products/zVSE/TCP-IP/TCP-IP.htm, or from an IBM PTF.

We strongly recommend that you do not modify the original members. If you need a different translation table, copy the table that most closely resembles what you need to a new library member, change the new library member, and then add a definition for this new member and table name(s) to your TCP/IP FOR VSE initialization member.

Single-Byte Member

A single-byte translation member contains paired, bi-directional EBCDIC/ASCII translation tables. Each record in the member is an 80-byte text record. The records are structured as follows:

- The first record in the member is a descriptor record that can contain any type of text. When the member is processed, this record is read and discarded.
- The following records consist of one or more translation tables. Each translation table consists of the following records:
  - An identifier record containing a 1- to 16-character entry name.
  - A set of 16 records that defines the EBCDIC-to-ASCII table.
  - A set of 16 records that defines the ASCII-to-EBCDIC table.

The records that define the translation tables consist of assembler-style 16-byte hexadecimal constants. There must be no leading blanks. A comment may be added starting at column 45. The following line contains a sample record:

```
X'000102030405060708090A0B0C0D0E0F'
```

Double-Byte Member

A double-byte member contains code page definitions in which each character is identified by its symbolic name and is mapped to a two-byte hexadecimal value. Each double-byte member contains a series of EBCDIC and ASCII code pages. These code pages are arranged in a columnar fashion. Each code page occupies a set of columns that spans all of the records in the member. The records are structured as follows:

- The first record contains the code page names. The first field is the string ID. This is followed by a series of code page names, separated by one or more blanks. The code page numbers are standardized.
- The second record begins with the string LOW. This is followed by the lowest hexadecimal value defined for each code page. Again, the fields are separated by one or more blanks.
- The third record begins with the string HIGH. This is followed by the highest hexadecimal value defined for each code page. Again, the fields are separated by one or more blanks.
- The fourth record begins with the string FILL. The values that follow are used to fill in all code page positions that are not explicitly defined. Note that the fill value is associated with the target code page.
- The remaining records contain data points. The first field in each record contains the symbolic name assigned to a character. The remaining fields contain the hexadecimal value that maps each code page to the character. If a character is not defined in a code page, the string '****' is used as filler.

  The data point records occur in ascending collating sequence by character name. If a character can be represented by more than one hexadecimal value, the data point record can be repeated as often as required with redundant values replaced by ****. When multiple values are assigned to the same symbol, the value specified last is considered to be the preferred value. Code pages containing duplicate definitions for symbols do not yield reversible translations.

Null Translation

It is sometimes convenient to bypass translation. TCP/IP FOR VSE defines a special translation table called the NULL translate table. When TCP/IP FOR VSE encounters a request to perform translation using a translate table named NULL, the translation routine is completely bypassed. Note that there is no translate table called "NULL," so if you want NULL translation with one or two changes, you need to define a custom translate table.

The CSTRAN Macro

Purpose

You can use the CSTRAN macro to generate translation table pairs and to create the single-byte translation member. Its use is not required, and you can code the '.L' members directly if you want.

The macro provides the following functions:

- Generates any CSI International-provided table by specifying its name.
- Permits you to code a table by providing overrides to a standard table.
- Generates the reverse translation table automatically after you specify a table pair.
- Flags non-unique translations (those where two or more characters translate to the same value).

Syntax

The CSTRAN macro continues to support the previously documented high-level assembler format. But, because there is no benefit over the F-level assembler form, we now restrict our documentation to the single form.

As you code the CSTRAN macro, be especially careful about commas, equal signs, and continuation characters. Many technical support calls on this subject are due to missing punctuation.

The format of the CSTRAN macro is shown below.

```
CSTRAN ['id',]NAME=name,FIRST=table, *
            [,CODEPG=pg][,UNIQUE=YES|NO][,REVERSE=YES|NO],           *
             E2AX=' 0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F', *
             E2A0='hh hh hh hh hh hh hh hh hh hh hh hh hh hh hh hh', *
             E2A1='hh hh hh hh hh hh hh hh hh hh hh hh hh hh hh hh', *
. . .
             E2AF='hh hh hh hh hh hh hh hh hh hh hh hh hh hh hh hh', *
             A2EX=' 0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F', *
             A2E0='hh hh hh hh hh hh hh hh hh hh hh hh hh hh hh hh', *
             A2E1='hh hh hh hh hh hh hh hh hh hh hh hh hh hh hh hh', *
. . .
             A2EF='hh hh hh hh hh hh hh hh hh hh hh hh hh hh hh hh'
```

The CSTRAN parameters have the following meanings:

Parameter	**Description**
id	A 1- to 40-byte identification string that occupies the first record of the translation member. This field is enclosed in single quotes. The field is ignored except for the first occurrence of the CSTRAN macro in the assembly.
NAME	A 1- to 16-byte name to be applied to the generated translation table pair. This name is the “ENTRY=” name referred to by the DEFINE TRANSLATION command.
FIRST	The value for this parameter, *table*, can be either E2A (for EBCDIC to ASCII) or A2E (for ASCII to EBCDIC). It determines which table is processed first. This is an important consideration because the table that is processed second may have the reverse of the first table for its default values. When coding this value, do not enclose it in quotes.
hh	A two-digit hexadecimal field. Each sub-argument keyword specifies values for 16 positions in a translation table. Keywords E2A*n*= (where *n* is a hexadecimal value from 0 through F) define the EBCDIC to ASCII table and keywords A2E*n*= (where *n* is a hexadecimal value from 0 through F) define the ASCII to EBCDIC table. The two-digit hexadecimal values are separated from each other by single spaces, and omitted values are coded as two blanks. In this way, the table is coded as a grid. You can code the keyword parameters in any order and you can omit blank lines; however, coding keywords in order makes reading the table easier.
E2AX A2EX	Two special keywords are available for use as labels for the columns of your tables. The values coded for these two keywords are ignored and are not processed.
UNIQUE= [YES | NO]	This parameter determines whether non-unique translations are flagged as warnings. Non-unique translations are those where two or more characters translate to the same value, meaning that accurate reverse translation of data is not possible. Coding NO suppresses the warning messages. Coding YES (the default) generates warning messages.

Parameter	**Description**
pg	This value specifies code pages to be used to generate the translation tables. The values in the specified code pages provide the starting point for building the tables. Any value that you provide overrides the corresponding code page value. The default value is "CSI." Allowable values are listed in "Code Pages" on page 195.
REVERSE= [YES | NO]	After the macro generates the first translation table, it normally reverses the values and generates the second table. Any values you provide are then used as overrides to these defaults. But, some translation tables are NOT reversible. By coding REVERSE=NO, you force the CSTRAN macro to use its internally defined default tables for both E2A and A2E tables. You must then explicitly provide your overrides for both translation directions.

Macro Operation

The CSTRAN macro processes all specified data before generating the translation tables. You can code the tables in either order (the macro always generates the ASCII-to-EBCDIC table first).

The macro uses the following algorithm:

- You specify the source for the default table values.
- The macro creates the default table for the specified translation direction.
- The macro processes your data specifications as overrides to the default table.
- The macro creates the second table's default values by inverting the first table unless you specify REVERSE=NO, in which case it uses the internal default values.
- The macro processes your data specifications as overrides to the derived (or default) second table. If you provide no data, the second table exactly reverses the first table's action (unless you specify REVERSE=NO).
- If you specify UNIQUE=YES, either by code or by default, the macro may provide a list of characters whose translations are not unique. The list identifies situations where two or more characters translate to the same value and, therefore, the translations cannot be reversed.

Assembly Output

The output from the CSTRAN macro is the .L book used to load translation tables. CSTRAN does not generate object code or TXT decks. Do NOT submit the assembler output to the linkage edit; it will not work.

Activating a Table

To load and use a translation table, use the DEFINE TRANSLATION command. The command syntax for loading a single-byte table follows.

```
DEFine TRANslation,MEMber=mem[,NAMe=name][,ENTry=entryname]

DEFine TRANslation,DEFault=name
```

In this command, *mem* is the member name of the .L book that contains the translation table(s) and *entryname* is the specific translate table that you want to load. If you load a specific translate table, you can assign this table a different name in storage using the NAME parameter. If you omit the ENTRY field, all tables in the specified member are loaded and they are assigned the same name in storage as the *entryname*.

To set the system default translation table, use the DEFAULT parameter either by itself or appended to another DEFINE TRANSLATION.

Once a translation table is loaded, it cannot be deleted. You can, however, redefine the table by loading it again.

Code Pages

The CSTRAN macro constructs tables based upon a number of IBM standard code pages. You specify which code pages are used by means of the CODEPG= parameter. The following table shows each value for CODEPG= and the corresponding code pages generated. All of the following tables are shipped in member IPXLATE.L. You can load them by using the DEFINE TRANSLATION command and the Entry Name from the table.

IPXLATE.L Entry Name	CODEPG=	EBCDIC	ASCII	Comment
ARABIC_01	420864	420	864	420 Arabic 864 PC Data * Enforced Subset Match
ARABIC_02	420864	420	864	420 Arabic 864 PC Data * Customized Round Trip
ARABIC_03	420864	420	864	420 Arabic 864 PC Data * Round Trip Algorithm One
ARABIC_04	4201089	420	1089	420 Arabic 1089 Arabic (ISA 8859-6)
ARABIC_05	4201256	420	1256	420 Arabic 1256 PC Data 1256 MS Windows
BALTIC_01E	B56901RT	1156	901	Multi with Euro (PC) (Round Trip)

IPXLATE.L Entry Name	**CODEPG=**	**EBCDIC**	**ASCII**	**Comment**
BALTIC_02E	B56901ES	1156	901	Multi with Euro Enforced Subset
BALTIC_03E	B565353RT	1156	5353	Multi with Euro MS Windows Round Trip
BALTIC_04E	B656353ES	1156	5353	Multi with Euro MS Windows Enforced Subset
BELGIUM_01	500437	500	437	500 International Latin-1 437 PC Data, PC Base * Customized Round Trip
CHINA_01	037A43	037	1043	037 USA, CANADA − CECP 1043 Traditional Chinese Extended − PC
CHINA_02	8361114	836	1114	836 Simplified Chinese Extended 1114 Republic of China (ROC) − PC
CHINA_03	8361115	836	1115	836 Simplified Chinese Extended 1115 People's Republic of China (PRC) − PC
DN 02E	1142858	1142	858	1142 Denmark, Norway ECECP 858 PC Multilingual with Euro
DN_01	277437	277	1252	277 Denmark, Norway 437 PC Multilingual
DN_02	277850	277	1252	277 Denmark, Norway 850 PC Data MLP 222 Latin-1 Countries
DN_03	2771252	277	1252	277 Denmark, Norway 1252 MS Windows, Latin-1
DN_03E	11425348	1142	5348	1142 Denmark, Norway ECECP (Euro) 1252 MS Windows, Latin-1 (Euro)
DN_03E	11435348	1143	5348	1143 Finland, Sweden ECECP 1252 MS Windows, Latin-1 (Euro)
FRANCE_01	297437	297	437	297 France − CECP 437 PC − Multilingual

IPXLATE.L Entry Name	CODEPG=	EBCDIC	ASCII	Comment
FRANCE_02	297850	297	850	297 France − CECP 850 PC Data MLP 222 Latin-1 Countries
FRANCE_02E	1147858	1147	858	1147 France ECECP 858 PC Multilingual with Euro
FRANCE_03	297A52	297	1252	297 France − CECP 1252 MS Windows, Latin-1
FRANCE_03E	11475348	1147	5348	1147 France ECECP (Euro) 1252 MS Windows, Latin-1 (Euro)
FS_01	278437	278	1252	278 Finland, Sweden 437 PC Multilingual
FS_02	278850	278	1252	278 Finland, Sweden 850 PC Data MLP 222 Latin-1 Countries
FS_02E	1143858	1143	858	1143 Finland, Sweden ECECP 858 PC Multilingual with Euro
FS_03	2781252	278	1252	278 Finland, Sweden 1252 MS Windows, Latin-1
GERMAN_01	273437	273	437	273 Germany, Austria − CECP 437 PC − Multilingual
GERMAN_02	273850	273	850	273 Germany, Austria − CECP 850 PC Data MLP 222 Latin-1 Countries
GERMAN_02E	1141858	1141	858	1141 Austria, Germany ECECP 858 PC Multilingual with Euro
GERMAN_03	273A52	273	1252	273 Germany, Austria − CECP 1252 MS Windows, Latin-1
GERMAN_03E	11415348	1141	5348	1141 Germany, Austria − CECP 1252 MS Windows, Latin-1 (Euro)
INTER_01	5001252	500	1252	500 International Latin-1 1252 MS Windows, Latin-1
INTER_01E	1148858	1148	858	1148 International ECECP (Euro) 1252 MS Windows, Latin-1 (Euro)

IPXLATE.L Entry Name	CODEPG=	EBCDIC	ASCII	Comment
INTER_02	500850	500	850	500 International Latin-1 850 PC Data MLP 222 Latin-1 Countries
INTER_02E	11485348	1148	5348	1148 International ECECP (Euro) 1252 MS Windows, Latin-1 (Euro)
ITALY 02E	1144858	1144	858	1144 Italy ECECP 858 PC Multilingual with Euro
ITALY_01	280437	280	437	280 Italy − CECP 437 PC − Multilingual
ITALY_02	280850	280	850	280 Italy − CECP 850 PC Data MLP 222 Latin-1 Countries
ITALY_03	280A52	280	1252	280 Italy − CECP 1252 MS Windows, Latin-1
ITALY_03E	11445348	1144	5348	1144 ITALY ECECP (Euro) 1252 MS Windows, Latin-1 (Euro)
JAPAN_01	10271041S	1027	1041	1027 Japanese Latin, extended Host 1041 Japanese PC Data, extended * Enforced Subset
JAPAN_02	10271041R	1027	1041	1027 Japanese Latin, extended Host 1041 Japanese PC Data, extended * Round trip algorithm 2
JAPAN_03	1027897	1027	897	1027 Japanese Latin, extended Host 897 Japanese PC Data SB * Customized Enforced Subset
JAPAN_04	1027290	1027	n/a	1027 Japanese Latin, extended Host 290 Japanese Katakana, extended host * Round trip
JAPAN_05	2901041S	290	1041	290 Japanese Katakana, extended host 1041 Japanese PC Data, extended * Enforced Subset

IPXLATE.L Entry Name	CODEPG=	EBCDIC	ASCII	Comment
JAPAN_06	2901041R	290	1041	290 Japanese Katakana, extended host 1041 Japanese PC Data, extended * Round trip algorithm 2
JAPAN_07	290897	290	897	290 Japanese Katakana, extended host 897 Japanese PC Data SB
KOREA_01	8331040	833	1040	833 Korean, extended host SB 1040 Korean Extended − PC * Enforced Subset
KOREA_02	8331088	833	1088	833 Korean, extended host SB 1088 Korean PC Data SB * Enforced Subset Match (default)
KOREA_03	8331126	833	1126	833 Korean, extended host SB 1126 Windows Korean PC Data Single-Byte * Enforced Subset
OS_01	10471252S	1047	1252	1047 Latin-1/ Open Systems 1252 MS Windows, Latin-1 * Enforced Subset
OS_02	10471252R	1047	1252	1047 Latin-1/ Open Systems 1252 MS Windows, Latin-1 * Round Trip
OS_03	1047437S	1047	437	1047 Latin-1/ Open Systems 437 PC Data, PC Base * Enforced Subset
OS_04	1047437R	1047	1252	1047 Latin-1/ Open Systems 437 PC Data, PC Base * Round Trip
OS_05	10471252S	1047	1252	1047 Latin-1/ Open Systems 850 PC Data MLP 222 Latin-1 Countries * Round Trip
SPAIN 02E	1145858	1145	858	1145 SPAIN, Latin America ECECP 858 PC Multilingual with Euro

IPXLATE.L Entry Name	**CODEPG=**	**EBCDIC**	**ASCII**	**Comment**
SPAIN_01	284437	284	437	284 Spain (Latin America) − CECP 437 PC - Multilingual
SPAIN_02	284850	284	850	284 Spain (Latin America) − CECP 850 PC Data MLP 222 Latin-1 Countries
SPAIN_03	284A52	284	1252	284 Spain (Latin America) − CECP 1252 MS Windows, Latin-1
SPAIN_03E	11455348	1145	5348	1145 SPAIN, Latin America ECECP (Euro) 1252 MS Windows, Latin-1 (Euro)
UK_ENG_01	285437	285	437	285 United Kingdom − CECP 437 PC - Multilingual
UK_ENG_02	285850	285	850	285 United Kingdom – CECP 850 PC Data MLP 222 Latin-1 Countries
UK_ENG_02E	1146858	1146	858	1146 UK ECECP (Euro)
UK_ENG_03	285A52	285	1252	285 United Kingdom − CECP 1252 MS Windows, Latin-1
UK_ENG_03E	11465348	1146	5348	1146 UK ECECP (Euro) 858 PC Multilingual with Euro.
US_ENG_01	037437	037	437	037 USA, CANADA − CECP 437 PC - Multilingual
US_ENG_02	037850	037	850	037 USA, CANADA − CECP 850 PC Data MLP 222 Latin-1 Countries
US_ENG_02E	0378501140858	0371140	850858	037 USA, CANADA − CECP 850 PC Data MLP 222 Latin-1 Countries 1140 USA, Canada – ECECP 858 PC - Multilingual with Euro
US_ENG_03	037A52	037	1252	037 USA, CANADA − CECP 1252 MS Windows, Latin-1
US_ENG_03E	1140858	1140	5348	1140 USA, Canada – ECECP 1252 MS Windows, Latin-1 (Euro)

Coding Example

Source Job

The following job stream generates a user-defined table named USER1. This table is identical to the CSI-provided US_ENG_03 table except that it provides equivalents for the ASCII DC3, GS, and US characters.

```
* $$ JOB JNM=CREATE,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
* $$ PUN CLASS=0,DISP=I
// JOB CREATE
// LIBDEF *,SEARCH=PRD2.TCPIP
// OPTION DECK
// EXEC ASSEMBLY
         PUNCH    '* $$ JOB JNM=CATALOG,CLASS=A,DISP=D'
         PUNCH    '* $$ LST CLASS=A,DISP=D'
         PUNCH    '// JOB CATALOG'
         PUNCH    '// EXEC LIBR,PARM=''ACC SUB=PRD2.TCPIPCFG'''
         PUNCH    'CATALOG USER1.L  REPLACE=YES'
         CSTRAN NAME=USER1,FIRST=A2E,CODEPG=0371252,                   *
               A2EX=' 0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f', *
               A2E0='                                               ', 0
               A2E1='         3B                            1D    1F', 1
               A2E2='                                               ', 2
               A2E3='                                               ', 3
               A2E4='                                               ', 4
               A2E5='                                               ', 5
               A2E6='                                               ', 6
               A2E7='                                               ', 7
               A2E8='                                               ', 8
               A2E9='                                               ', 9
               A2EA='                                               ', a
               A2EB='                                               ', b
               A2EC='                                               ', c
               A2ED='                                               ', d
               A2EE='                                               ', e
               A2EF='                                             FF'
         PUNCH    '/+'
         PUNCH    '/*'
         PUNCH    '/&&'
         PUNCH    '* $$ EOJ'
         END
/*
/&
* $$ EOJ
```

This example causes two jobs to execute. The first, CREATE, executes the assembler and processes the CSTRAN macro specifications. The assembler also processes the PUNCH statements to wrap the CSTRAN output in JCL and control statements to generate the job CATALOG. This second job actually stores the translation table in the designated library.

We decided to specify the ASCII-to-EBCDIC values and allow the macro to generate the corresponding EBCDIC to ASCII values automatically. Also, we coded the entire ASCII-to-EBCDIC table, although we could have omitted those lines without overrides.

The macro permits including the header keyword A2EX=. This helps place your definitions in a grid. We also chose to code the *row value* as the continuation character (column 72).

We did not code any E2A*n*= keywords as we wanted the reverse table to be auto generated.

Generated Table

The following display shows the contents of USER1.L, which was generated by the preceding example. A label was added to the first data line.

```
DESCRIPTIVE COMMENT.
USER1
X'000102039C09867F978D8E0B0C0D0E0F'         This label begins at column 45
X'101112009D8508871819928F1C1D1E1F'
X'80818283840A171B88898A8B8C050607'
X'909116939495960498999A9B14159E1A'
X'20A0E2E4E0E1E3E5E7F1A22E3C282B7C'
X'26E9EAEBE8EDEEEFECDF21242A293BAC'
X'2D2FC2C4C0C1C3C5C7D1A62C255F3E3F'
X'F8C9CACBC8CDCECFCC603A2340273D22'
X'D8616263646566676869ABBBF0FDFEB1'
X'B06A6B6C6D6E6F707172AABAE6B8C6A4'
X'B57E737475767778797AA1BFD0DDDEAE'
X'5EA3A5B7A9A7B6BCBDBE5B5DAFA8B4D7'
X'7B414243444546474849ADF4F6F2F3F5'
X'7D4A4B4C4D4E4F505152B9FBFCF9FA00'
X'5CF7535455565758595AB2D4D6D2D3D5'
X'30313233343536373839B3DBDCD9DAFF'
X'00010203372D2E2F1605250B0C0D0E0F'
X'1011123B3C3D322618193F271C1D1E1F'
X'405A7F7B5B6C507D4D5D5C4E6B604B61'
X'F0F1F2F3F4F5F6F7F8F97A5E4C7E6E6F'
X'7CC1C2C3C4C5C6C7C8C9D1D2D3D4D5D6'
X'D7D8D9E2E3E4E5E6E7E8E9BAE0BBB06D'
X'79818283848586878889919293949596'
X'979899A2A3A4A5A6A7A8A9C04FD0A107'
X'202122232415061728292A2B2C090A1B'
X'30311A333435360838393A3B04143EFF'
X'41AA4AB19FB26AB5BDB49A8A5FCAAFBC'
X'908FEAFABEA0B6B39DDA9B8BB7B8B9AB'
X'646562666363679E687471727378757677'
X'AC69EDEEEBEFECBF80FDFEFBFCADAE59'
X'4445424643479C485451525358555657'
X'8C49CDCECBCFCCE170DDDEDBDC8D8EFF'
```

Double-Byte Character Sets

Where Used

The Chinese, Japanese, and Korean alphabets are too large to be represented by an eight-bit byte. To support these alphabets, double-byte character sets (DBCS) were developed.

In DBCS, the most-used characters are assigned one-byte values. Other characters are assigned two-byte (16-bit) representations. Two escape characters, shift-out (SO) and shift-in (SI), usually (but not always) control switching between the two character sets.

DBCS Data Structure: EBCDIC

On the mainframe, EBCDIC DBCS records generally begin in single-byte mode. An SO character marks the start of double-byte mode. An SI character returns to single-byte mode. Although a record can end while in double-byte mode, transmittal to an ASCII platform usually requires adding carriage-return and line feed characters as a record delimiter. Because these are single-byte characters, an SI may be needed if a record ends in double-byte mode.

DBCS Data Structure: ASCII

On an ASCII platform, ASCII DBCS data is structured quite differently from its mainframe counterpart. First, datasets are merely a stream of bytes and have no actual record orientation. Normally, however, a delimiting character is used to break the data into logical records.

ASCII DBCS data does not normally contain SO or SI characters. Each DBCS character is identified by the value of its first (high-order) byte. To visualize how this works, first consider a single-byte character set (SBCS) code page. There are 256 possible values, each representing one character. In a DBCS character set, each of the values may still represent a single character OR they may represent a one-byte escape character with the following byte used for an additional 256 characters. In this way, a DBCS character set may contain from 256 to 65,536 characters.

Conversion Problems

Converting DBCS data streams between ASCII and EBCDIC presents a problem: what do we do with the SO/SI delimiters? By default, we remove them when translating to ASCII and add them back when translating to EBCDIC. Unfortunately, this means that there is no longer a one-to-one correspondence in the position of data bytes in the input and output files.

To retain this property, we allow for adding placeholders in the ASCII data stream. The following three possible placeholder characters have been defined:

- EBCDIC SO and SI. In this mode, the EBCDIC SO (X'0E') and SI (X'0F') characters are retained and are included in the ASCII dataset.
- ASCII SO and SI. In this mode, the ASCII equivalents of SO (X'1E') and SI (X'1F') take the place of their EBCDIC equivalents.

- Blanks. Finally, we can remove the EBCDIC SO and SI characters and replace them with an ASCII space character (X'20'), thus preserving the data character spacing.

When translating such a dataset back to EBCDIC, we still detect DBCS characters by their first-byte values. But, we replace the leading and trailing placeholders with SO and SI rather than lengthening the data.

While translating data in placeholder mode, we validate the presence of the appropriate placeholder in the input data stream. An improper placeholder results in the termination of data transfer.

Other Considerations

Mixing single-byte and double-byte characters means that byte-addressed accessing of data is not possible. For example, if we were to extract byte 983 from a data stream, we could not easily determine if it was a single-byte character or the second half of a double-byte character. This means that all such extractions must scan backwards to a SO or SI character (EBCDIC) or to a known single-byte character (ASCII) and then proceed forward to the desired location. Also note that it becomes even more difficult to position a data stream by character number than by byte, due to the mixture of single-byte and double-byte characters.

To make life a little easier, there are some byte values that never occur as either the first or second byte in a double-byte character. Some of these are as follows:

- SO (X'0E')
- SI (X'0F')
- NL (X'15')
- CR (X'0D')
- LF (X'0A')

Single-Byte Translation Tables

Single-byte translation is controlled by a standard translation table pair (EBCDIC to ASCII, ASCII to EBCDIC). TCP/IP FOR VSE includes suitable single-byte tables in the IPXLATE.L book. You are free to customize these tables or to add additional tables as needed. But, you should use great care in doing so.

Double-Byte Translation Tables and Code Pages

Each pair of double-byte translation tables requires 256K bytes of storage. They are generated from code pages supplied by CSI International in cooperation with IBM.

Each code page consists of a list of named symbols and their associated numeric values. Because some symbols may be represented by more than one numeric value, provision is made to translate all acceptable values for a given symbol to the target value. During reverse translation, the character converts to its preferred value.

To generate a DBCS translation table, you specify an EBCDIC and an ASCII code page. The symbols are matched automatically and are used to build the translation table. You assign a name to the resulting double-byte translation table, which also pairs it with a single-byte translation table. In this manner, only a single name is required to specify a translation table pair.

Code Page Source Files

TCP/IP FOR VSE ships three code page source files named CHINA.L, JAPAN.L, and KOREA.L. Each file contains the code pages relevant to the associated nationality.

Each code page file consists of 80-byte records, each containing positional fields. The fields are blank-delimited. Multiple blanks are compressed to single blanks during processing. The records are structured as follows:

- The first record contains the name of each code page. The data associated with each code page continues in a vertical structure beneath the code page name.
- The second record contains the lowest value assigned in the code page.
- The third record contains the highest value assigned in the code page.
- The fourth record contains the default (or fill) character to be used in each table position that is not explicitly defined.
- The remaining records contain the individual code points. The first field is the symbol name. The remaining fields define the two-byte hexadecimal values. If a symbol is undefined in a particular code page, its value is specified as ****. The code point records are sorted by symbol name in ascending order. If more than one value is required for a given symbol, the symbol name is repeated on multiple, consecutive records. In code pages that do not have additional values, the **** placeholder is coded. Where multiple values are supplied, the value coded last is considered to be the preferred value and is the value assigned when translating to the code page in question.

Obtaining the Double-Byte Code Page Files

Three code page files are provided for DBCS support:

- CHINA.L
- JAPAN.L
- KOREA.L

These files are in member DBCS01 and may be obtained from CSI International (TCP/IP for VSE web page; click on “Chinese, Japanese, Korean language support” to access DBCS01.BJB) or IBM. You need to obtain and load the appropriate member into your VSE library.

Activating DBCS Support

When TCP/IP FOR VSE begins execution, table US_ENG_03 is loaded from member IPXLATE.L and is made the default translation table. This table provides a single-byte base suitable for most common translation needs. You may want to load additional single-byte tables for your own use, and you need to load one or more single-byte tables to associate with double-byte tables.

You can find a comprehensive list of all SBCS translation tables earlier in this chapter.

To create and load a DBCS translation table, you must use the DEFINE TRANSLATION command. To illustrate, we use the following example. We are creating a Japanese double-byte translation table. The single-byte translation is handled by SBCS table JAPAN_09. The double-byte table converts between EBCDIC code page 300 and ASCII code page 301.

First, we load and name the single-byte table:

```
DEFINE TRANSLATION,TYPE=SINGLE,MEMBER=IPXLATE, -
ENTRY=JAPAN_09,NAME=JAPAN_DBCS
```

Next, we load the double-byte table:

```
DEFINE TRANSLATION,TYPE=DOUBLE,MEMBER=JAPAN, -
NAME=JAPAN_DBCS,EBCDIC=300,ASCII=301
```

Finally (and optionally), we make this table the default:

```
DEFINE TRANSLATION,DEFAULT=JAPAN_DBCS
```

It is not always appropriate to make a double-byte translation table the default for the entire system. Remember that you can always use the FTP SITE command to explicitly select a double-byte translation table.

Using DBCS Translation

There are several ways to specify a translation table. The most general is to make a particular table the system default. To do this, use the DEFINE TRANSLATION command, as follows:

```
DEFINE TRANSLATION,DEFAULT=name
```

This may or may not be appropriate for your installation. FTP users can override the above specification with the SITE command.

To force a translation table on FTP users, you can code the TRANSLATE= parameter in the DEFINE FTPD command, as follows:

```
DEFINE FTPD,TRANSLATE=name,...
```

A translation table specified in this way *cannot* be overridden with a SITE command.

Translation tables can also be associated with datasets, as shown here:

```
DEFINE FILE,PUBLIC='xxx.yyy.zzz',TRANSLATE=name
```

This specification takes precedence over that coded with the DEFINE FTPD command, and it *cannot* be overridden with a SITE command. The same physical file can be defined multiple times with different names and different translation tables.

DBCS Behavior with FTP

EBCDIC data streams always use SO/SI to bracket DBCS character strings. There is no restriction on the value of the first byte of a DBCS character, except that it cannot be SI or SO, CR, LF, or NL. Other values may also be reserved, such as X'00' and X'FF'.

ASCII data streams never use SO/SI but instead rely on the value of the first byte of the DBCS pair. Thus, there must be a “hole” in the SBCS code page to accommodate each range of 256 DBCS characters.

During standard translation, SO/SI characters must be added or removed as appropriate.

An optional protocol permits the SO/SI characters to be retained as placeholders in the ASCII stream. When translated in this manner, the EBCDIC and ASCII data streams have the same length (except for the ASCII end-of-record delimiters).

Note that these placeholders are not used when translating back to EBCDIC. Instead, we rely on the first-byte values; however, the appropriate placeholder *must* be present or an invalid data stream condition exists and the transfer stops.

Note:

When transferring a DBCS data file from a PC to VSE, the destination file must be able to contain records larger than the specified LRECL for the file. This is needed because SO/SI characters are added for each record containing DBCS characters. When CRLF=OFF, the ASCII byte stream is split into records using the specified LRECL value. The data is then translated to EBCDIC using SITE SOSI CONVERT, which inserts 2 bytes for each DBCS string included in the record. Therefore, the destination file on VSE must be defined with a record length of LRECL + 2 or, if multiple (*n*) DBCS strings are included in SBCS, with a record length of LRECL + (*n* × 2).

SITE Command

SO/SI handling is controlled by the SITE command. The following chart shows specific SITE commands and their meanings:

Command	Description
SITE SOSI CONVERT	This is the default and indicates that the data stream is converted and that SO/SI characters are added or removed as appropriate.
SITE SOSI KEEP	SO/SI characters are retained as placeholders in the ASCII stream.
SITE SOSI XLATE	SO/SI characters are retained as placeholders in the ASCII stream but are translated to their ASCII equivalents of X'1E' and X'1F'.
SITE SOSI BLANK	SO/SI characters are retained as placeholders in the ASCII stream but are translated to ASCII spaces of X'20'.
SITE SOSI NONE	SO/SI characters are not present in the EBCDIC data. Every pair of bytes is treated as DBCS. This mode is referred to as the graphic character set.

Enabling DBCS Support

TCP/IP FOR VSE release 1.5 and higher enables DBCS support for FTP and HTTP.

The TN3270 protocol passes data without translation and permits using DBCS data streams, provided that your TN3270 client is DBCS enabled.

12

Performance

Overview

In this chapter, we focus on TCP/IP FOR VSE performance and resource utilization. We cover the following topics:

- Concepts you must understand to tune your system effectively
- Performance statistics and where to find them
- Performance enhancement
- Port queuing, also known as connection queuing.

As with any performance discussion, our suggestions are guidelines, and your results will vary. Truly maximizing TCP/IP FOR VSE performance requires an in-depth knowledge of a variety of areas as well as your particular system. This chapter provides basic information you need to get started.

TCP/IP Concepts

Before you can tune your TCP/IP FOR VSE system, you need to understand some TCP/IP concepts. This section covers the following concepts and why they affect performance:

- Data flow
- MTU size
- TCP segment size
- TCP window size
- Retransmissions

Data Flow

Because TCP/IP FOR VSE's basic function is to communicate by moving data, it is important to understand how data flows through TCP/IP FOR VSE. You may have heard TCP/IP implementations referred to as the *stack*. The idea behind this term is that a chunk of data flows through many levels of processing before reaching its ultimate destination. They typically begin at the highest level (the application) and move to the lowest level (in our case, the link drivers), with a number of levels in between. These levels, as a unit, are referred to as the *stack*.

An important aspect of the stack's architecture is that each layer is concerned only with its own processing. For example, the IP layer does not care whether the datagram it is packaging for the protocol layer uses TCP, UDP, or another protocol. The IP layer's sole interest is to package the protocol segments into datagrams and then pass them to the transport layer. The way the transport layer moves datagrams across the network is not relevant to the IP layer.

The TCP/IP FOR VSE stack is described in the following table:

Stack Level	Description
Application	The highest level in the stack is the application level. This level includes clients, daemons, and all user-written programs. In the stack architecture, each application sees the rest of the stack as simply a doorway to another application. See the *TCP/IP FOR VSE Programmer's Guide* for more information about application programming interfaces.

Stack Level	Description
Protocol	The second highest level in the stack is the protocol level. The protocol level lies between the application and the IP level. Although there is no limit to the number of protocols that can be made available, for our purposes we will consider only the User Data Protocol (UDP) and the Transmission Control Protocol (TCP). UDP processing is straightforward. Data from the application is packaged with identifying information and then passed to the IP layer for further processing. When the remote stack's protocol layer receives a UDP packet, it interprets the header in formation and passes the data on to the appropriate application. UDP makes no guarantee that the data will arrive in the same sequence it was sent or even that it will arrive at all. TCP processing, on the other hand, incorporates a series of controls that ensure that all data received from an application will reach the remote application in exactly the same order it was sent without duplication or omission.
IP	The third highest level in the stack is the IP level. It is the IP layer's responsibility to add the appropriate header information to complete the datagram before passing it to the transport layer. This header primarily contains the IP addresses of both hosts. If necessary, the IP layer will fragment a datagram into multiple pieces that do not exceed the maximum permitted size. Conversely, on the receiving end the IP layer will reassemble fragments into a complete datagram before passing it to the protocol layer.
Transport	The lowest level in the stack is the transport level. It is composed of the link drivers that handle the physical I/O to the control units. The transport layer is concerned with moving datagrams onto and off of the physical network. In the case of a VSE host, the transport mechanism may consist of Ethernet, a token ring, CTCA, or other method. Depending upon the network's architecture, a single datagram may traverse several transport layers during its travels.

This table shows how data flows from the highest level to the lowest level. This is the process used when TCP/IP FOR VSE sends data to other hosts. Of course, the process is reversed for inbound packets, where data flows up the stack instead of down.

MTU Size

The Maximum Transmission Unit (MTU) size is the number of bytes that can be sent over the physical media without requiring fragmentation. This value is generally fixed by hardware, as follows:

Adapter	Default	Minimum	Maximum
Ethernet	1500	576	1500
Token Ring	1500	576	Depends on speed of the ring: 4 Mbit/sec ~ 4000 16 Mbit/sec ~ 8000
FDDI	1500	576	2000

The maximum MTU size for an Ethernet connection is 1500, which means that datagrams as large as 1500 bytes can fit into a single Ethernet frame. In general, larger MTU sizes are preferable as long as they do not exceed the maximum for the physical device. Larger MTU sizes permit more data to be transmitted in fewer transmission units.

The following detrimental effects can occur if you choose an inappropriate MTU size:

- If you use a smaller MTU size, such as the minimum of 576 bytes, you still have to send the same amount of data. However, it takes more transmission units to accommodate the data. More transmission units mean more I/Os for the link driver to handle, more acknowledgements for TCP to handle, more processing in general, and therefore more network overhead. That is, performance suffers.
- If you use a larger MTU size than the complete physical path can handle, IP packets are broken into fragments of a size that is acceptable to the most restrictive part of the network. Fragmentation is bad for the following reasons:
  - Some TCP/IP implementations do not accept fragmented packets, and the packets are discarded. As retransmissions are attempted, performance suffers.
  - Fragmentation requires more work for the IP layer on the other side of the connection because (even if it supports fragmentation) it needs to gather all pieces of the datagram, which are transmitted separately, and reassemble them before it passes them to the TCP level. This causes unacceptable delays for the application on the other side of the connection. Again, performance suffers.

TCP/IP FOR VSE does not allow you to set an MTU size that exceeds the capability of the physical device. Also, the MTU size is meaningful only for outbound datagrams.

An appropriate MTU size should be specified on each DEFINE LINK and DEFINE ADAPTER statement. This value will serve as a ceiling for all datagrams being sent through that link. In addition, a reduced MTU size may be specified on each DEFINE ROUTE statement.

For example, a VSE system communicates with VM by a CTCA link. The VM TCP/IP stack is then connected to the outside world by an Ethernet link. To obtain the best performance, we would set the MTU size for the CTCA link to its maximum value of 64K. To prevent fragmentation, however, we would include DEFINE ROUTE statements for all destinations reachable by the Ethernet link. On these statements, we would override the MTU value with a more reasonable value of 1500.

The DISCOVER client may be used to help determine the optimal MTU size. For details, see the *TCP/IP FOR VSE User Guide*, chapter 5, "Ping, Traceroute, DISCOVER Clients."

TCP Segment Size

A *TCP segment* is the largest unit of TCP data that the remote host will except in a single datagram. In practice, this unit cannot be larger than the MTU size minus 40 bytes. The 40 bytes is required because it is the combined length of the TCP and IP headers. When a TCP connection is established, each host informs the other of the maximum segment size that it will accept. To prevent fragmentation, the largest maximum segment size (MSS) that TCP/IP FOR VSE will request or honor is 40 bytes less than the MTU size.

By default, TCP/IP FOR VSE always requests an MSS value of MTU -40. If the remote host requests a larger value, it will be reduced to MTU -40. You can use the SET MAXIMUM_SEGMENT command to establish an explicit default value, or you can code the MSS parameter on your DEFINE ROUTE statements.

TCP Window Size

The *TCP window* is the quantity of data the receiving TCP accepts without waiting for an acknowledgement. The TCP window is a simple concept but one that is frequently misunderstood. Before discussing the TCP window in detail, we first need to understand some basic concepts.

The most important aspect of TCP is data reliability. Every bite in the TCP connection from beginning to end must be accounted for and be received in the same order that it was sent. To achieve this, the TCP protocol assigns a number to each byte in a connection. The range of numbers assigned to a particular connection is referred to as its sequence set. The TCP header always contains the sequence number of the first data byte in the packet as well as the number of bytes in the packet. Even if the data length is zero, the sequence number must still be present so that the packet can be properly placed within the sequence set. This strict numbering of transmitted data ensures that every character in the data stream will be received exactly as intended without duplication or omission.

Because it is always possible for an individual datagram to be corrupted or lost somewhere on the network, a method was devised that allows the sending stack to ensure receipt of each packet by the target stack. This is accomplished by including an acknowledgment field in the TCP header. When a TCP datagram is constructed, the acknowledgment field is set to the sequence number of the next expected byte to be received. If the incoming acknowledgment is lower than the highest numbered data byte sent, then the sending stack can precisely determine which datagram was lost and can begin retransmission at that point.

A problem arises when the sending stack delivers data more quickly than the receiving stack can process it. As each datagram is acknowledged, more datagrams are sent. While it is completely valid under the protocol for the receiving stack to simply throw away excess datagrams without acknowledging them, this could result in a flood of duplicate datagrams being retransmitted until an acknowledgment is sent. To prevent this, an additional field called the receive window was added to the TCP header.

The receive window works like this: When the receiving stack sends an acknowledgment, it sends two values. The ACK field contains the sequence number of the next expected data byte. The window field contains a numeric value which, when added to the ACK value, yields a value that is one greater than the highest sequence number that may be sent. If data arrives that is "outside the window," it is discarded and the current ACK and window values are sent in response.

There are several rules related to windows. Perhaps the most important rule is that the ending sequence number may never decrease in value. Another way of stating this is that once a stack has advertised a window for a certain amount of data, this may be viewed as a commitment. The stack may not change its mind and reduce the endpoint of the window. Any reference to a stack "closing the window" is incorrect. The intended meaning of such a statement is that the stack allowed the window to close by not extending the value of the ending byte.

Other rules concern the avoidance of Silly Window Syndrome (SWS). SWS occurs when the receiving stack advertises a window that is just a few bytes in size, and the sending stack immediately sends a small datagram to fill it. This behavior is so damaging that both sending and receiving stacks are expected to observe certain behaviors and to avoid others.

One behavior to avoid is the "shrinking window." Once a maximum size has been established for the window, it must always be returned to that size after it has been allowed to close. By the same token, the sending stack should not resume transmission until the window has regained its full size. Of course, there are exceptions. The guiding principle is to minimize the number of packets required to transmit the data.

The optimum size for the receive window varies depending on the platform. A device with limited memory may choose a window size that is only large enough to contain a few data packets. VSE, on the other hand, works best with large window values (~64K). Before selecting a small window size, remember that the optimum size for a datagram is the MSS.

To manage the TCP window size, you need to know how to set it and display its current value, both on the VSE side and the partner TCP side.

The following table summarizes this information for some popular platforms:

Platform	**Setting the Window**	**Displaying the Window**
TCP/IP	DEFINE ROUTE command	DIAGNOSE PERFORM QUERY CONNECTION QUERY ROUTE
AIX 4.3	NO (network option) recv_space AIX command	NO (network option) command reports the parameter
OS/400	CHGTCPA command (change TCP attributes)	CHGTCPA command also displays the attribute
Windows	Registry Entry (Windows 7): HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services\Tcpip\...	REGEDIT command

Retransmissions

A TCP/IP packet is retransmitted when it is not acknowledged. It does not matter why the transmission failed. When TCP/IP FOR VSE discovers that it needs to retransmit a packet, it enters *retransmission mode* for the connection. While in retransmission mode, TCP/IP FOR VSE does not transmit any more packets on the connection until *all* previously transmitted packets have been acknowledged.

If you use DIAGNOSE PERFORM, you can see how many times TCP/IP FOR VSE entered the retransmission mode during a TCP connection, as well the total number of retransmitted blocks and the total wall clock time in retransmission mode.

Remember that TCP/IP FOR VSE can control retransmissions only on outbound connections. Each TCP implementation has its own controls.

Some of these controls are described in the following table:

Platform	Retransmission Control
TCP/IP FOR VSE	As described above
AIX 4.3	NO (network option) *rto_length*, *rto_limit*, *rto_low*, and *rto_high* commands
OS/400	Cannot be set, but it can be displayed for TCP connections

To analyze retransmissions and help diagnose performance problems, TCP/IP FOR VSE provides a diagnostics display. To enable it, enter the following command:

```
DIAGNOSE PERFORM
```

When this command is issued, you receive output at the end of each TCP connection. The output looks similar to the following display:

```
IPT340D Connection Summary: F: 192.168.1.66/21 L: 192.168.1.161/4103 State: Closed BSD: 0
IPT345D  Open by IPNAFTPC in F4 Ident: C5506D3320A7B000
IPT341D  Start: 13:45:21 End:  13:45:22 Duration: 0.577 IPT343D  Route: LOCAL; MTU: 1,500;
Send MSS: 1,460 (1,460); Recv MSS: 1,460; Buffer: 65,534/1,825
IPT347D  Variable Retrans; Init: 1,000/1,000; Limit: 500/2,000; Delay: 500; Retries: 50
IPT349D  Pulse: Enabled; Interval: 60 sec; Count: 0
IPT342D  Send IS: 502A3321 FIN: 502A3380 Data: 94
IPT344D  Send: 18 blocks, 96 bytes. Retrans: 0 blocks, 0 bytes. Eff: 11%
IPT346D  Retransmit mode 0 times. Cost: 0 seconds.
IPT358D  Maximum Send window: 65,535; Closed: 0; Time closed: 0.000
IPT348D  Retran Start: 1,000; Times: 0; Current: Off; Blocks: 0; Cost: 0 sec
IPT352D  Roundtrip: Min: 2 ms; Max: 11 ms; Last: 6 ms
IPT342D  Recv IS: 4C000C52 FIN: 4C000D77 Data: 292
IPT344D  Recv: 9 blocks, 294 bytes. Duplicate: 0 blocks, 0 bytes. Eff: 44%
IPT358D  Maximum Recv window: 65,534; Closed: 0; Time closed: 0.000
IPT356I  Total sockets: RECV: 6 (292); SEND: 4 (94); STAT: 5; CLOSE: 1 ABORT: 0 CNTL: 0
```

Each message field is described in the *TCP/IP FOR VSE Messages*.

Statistics Command

To obtain performance statistics, issue the following command:

```
QUERY STATS
```

The following display shows sample output:

```
IPN253I (( TCP/IP Operational Statistics ))
IPN516I  FTP Daemons             = 3
IPN516I  - Current Active        = 1
IPN516I  - Maximum Active        = 1
IPN516I  - Current Buffers       = 0
IPN516I  - Maximum Active Buffers = 4
IPN516I  Telnet Daemons          = 33
IPN516I  - Current Active        = 0
IPN516I  - Maximum Active        = 0
IPN516I  - Current Buffers       = 0
IPN516I  - Maximum Active Buffers = 0
IPN516I  LP Daemons              = 3
IPN516I  HTTP Daemons            = 1
IPN516I  FTP Sessions            = 2
IPN516I  Telnet Sessions         = 0
IPN516I  LPR Requests            = 0
IPN516I  HTTP Requests           = 0
IPN516I  TCP Inbound Rejections  = 0
IPN516I  FTP Files Sent          = 1
IPN516I  FTP Files Received      = 1
IPN516I  FTP File Bytes Sent     = 230.5 K
IPN516I  FTP File Bytes Received  = 230.5 K
IPN516I  Telnet Bytes Sent       = 0
IPN516I  Telnet Bytes Received   = 0
IPN516I  TCP Bytes Sent          = 234.8 K
IPN516I  TCP Bytes Received      = 230.7 K
IPN516I  UDP Bytes Sent          = 0
IPN516I  UDP Bytes Received      = 0
IPN516I  IP Bytes Sent           = 250.1 K
IPN516I  IP Bytes Received       = 245.4 K
IPN516I  Storage Cushion Releases  = 0
IPN516I  Received Blocks         = 0
IPN516I  - Inbound Datagrams     = 0
IPN516I  -   Non-IP              = 0
IPN516I  -   Misrouted IP        = 0
IPN516I  -   Arps                = 0
IPN516I  -   Arp Requests        = 0
IPN516I  Transmitted Blocks      = 0
IPN516I  - Outbound Datagrams    = 0
IPN516I  -   Arp Requests        = 0
IPN516I  -   Arp Replies         = 0
```

Each measurement is described in the following table:

Measurement	Description
FTP Daemons	Number of FTP daemons active in the system. FTP daemons consume storage, so you should not define more than you need for concurrent FTP activity
FTP Current Active	Number of FTP daemons active at the time you issue the command

Measurement	Description
FTP Maximum Active	Highest number of concurrently active FTP daemons since TCP/IP FOR VSE initialization. Use this value to adjust the number of FTP daemons you define.
FTP Current Buffers	Number of FTP transfer buffers in use at the time you issue the command
FTP Maximum Active Buffers	Highest number of FTP transfer buffers that were ever in use at one time. Use this value to adjust the number of FTP transfer buffers.
Telnet Daemons	Number of telnet daemons active in the system. Telnet daemons consume storage and CPU resources, so you should not define more than you need for concurrent telnet activity.
Telnet Current Active	Number of telnet daemons active at the time you issue the command. Note that a telnet daemon is active even if the user is merely staring at the TCP/IP FOR VSE telnet menu.
Telnet Maximum Active	Highest number of concurrently active telnet daemons since TCP/IP FOR VSE initialization. Use this value to adjust the number of telnet daemons you define.
Telnet Current Buffers	Number of telnet transfer buffers in use at the time you issue the command. This value refers only to shared buffers. Telnet buffers using dedicated pools are not counted.
Telnet Maximum Active Buffers	Highest number of telnet transfer buffers that were ever in use at one time. Use this value to adjust the number of telnet transfer buffers. Only telnet daemons using the shared pool are counted.
LP Daemons	Number of defined line printer daemons
HTTP Daemons	Number of defined HTTP daemons
FTP Sessions	Total number of FTP sessions since TCP/IP FOR VSE initialization
Telnet Sessions	Total number of telnet sessions since TCP/IP FOR VSE initialization
LPR Requests	Total number of LPR requests since TCP/IP FOR VSE initialization

Measurement	Description
HTTP Requests	Total number of web requests since TCP/IP FOR VSE initialization
TCP Inbound Rejections	Number of times a request was rejected because TCP/IP FOR VSE could not find an application listening to the requested port
FTP Files Sent	Total number of files sent from VSE to a remote system
FTP Files Received	Total number of files received by VSE from a remote system
FTP File Bytes Sent	Total number of bytes sent by outbound FTP requests
FTP File Bytes Received	Total number of bytes received by inbound FTP requests
Telnet Bytes Sent	Total number of bytes sent by outbound TN3270 requests (such as screen updates)
Telnet Bytes Received	Total number of bytes received by inbound TN3270 requests (such as data entered in 3270 sessions)
TCP Bytes Sent	Total number of TCP bytes sent since TCP/IP FOR VSE initialization
TCP Bytes Received	Total number of TCP bytes received since TCP/IP FOR VSE initialization
UDP Bytes Sent	Total number of UDP bytes sent since TCP/IP FOR VSE initialization
UDP Bytes Received	Total number of UDP bytes received since TCP/IP FOR VSE initialization
IP Bytes Sent	The total number of bytes sent since TCP/IP FOR VSE initialization
IP Bytes Received	The total number of bytes received since TCP/IP FOR VSE initialization
Storage Cushion Releases	Number of times TCP/IP FOR VSE released storage because it detected a short-on-storage condition. If this value is not zero, consider running TCP/IP FOR VSE in a larger partition. For more information, see the next section, “Performance Factors.”

Measurement	**Description**
Received Blocks	Valid for LCS interfaces only. Total number of blocks received from the LCS device.
Received Inbound Datagrams	Valid for LCS interfaces only. Total number of datagrams received from the LCS device. Note that you might have multiple datagrams in a block.
Received Inbound Datagrams – Non-IP	Valid for LCS interfaces only. Total number of datagrams TCP/IP FOR VSE received that were not related to IP traffic. This value should be zero. If it is not, consider filtering and eliminating this traffic at the control unit layer. Examples of non-IP datagrams include IPX traffic and Microsoft Networking (NetBEUI) traffic. To determine the type of traffic, run a TCP/IP FOR VSE trace. For more information, see the TRAFFIC and the DEFINE TRACE commands in the *TCP/IP FOR VSE Command Reference*.
Received Inbound Datagrams – Misrouted IP	Valid for LCS interfaces only. Total number of datagrams received by TCP/IP FOR VSE that were destined for another IP address. If you are not running TCP/IP FOR VSE with GATEWAY ON, this value should be zero.
Received Inbound Datagrams – ARPs	Valid for LCS interfaces only. Total number of ARPs received by TCP/IP FOR VSE.
Received Inbound Datagrams – ARP requests	Valid for LCS interfaces only. Total number of ARP requests received by TCP/IP FOR VSE.
Transmitted Blocks	Valid for LCS interfaces only. Total number of blocks transmitted over the LCS interface.
Transmitted Blocks – Outbound Datagrams	Valid for LCS interfaces only. Total number of datagrams transmitted over the LCS interface. Note that multiple datagrams can be contained in a single block.
Transmitted Blocks – ARP requests	Total number of ARPs that TCP/IP FOR VSE sent out over the LCS interface.
Transmitted Blocks – ARP replies	Total number of ARP requests that were replied to.

Performance Factors

In the previous sections of this chapter we described TCP/IP FOR VSE concepts and performance measurements and how these relate to system performance. In this section, we explain how to improve performance by maximizing storage utilization. We also describe how to improve FTP and TN3270 performance.

Storage Utilization

TCP/IP FOR VSE uses both 24-bit and 31-bit partition GETVIS in its normal processing. This section helps you estimate the *minimum* amount of storage required for the TCP/IP partition. As the TCP/IP FOR VSE partition becomes more active, it gradually uses more and more storage. In general, once TCP/IP FOR VSE obtains storage, it does not free it unless it encounters a short-on-storage condition.

TCP/IP FOR VSE does not directly allocate storage in the general subpool. All TCP/IP FOR VSE GETVIS allocations are tagged with unique subpool IDs. The following table shows a few of the subpools that TCP/IP FOR VSE uses:

Subpool	Related Use
LDBLOK	Program loading
TKBLOK	Task blocks within the TCP/IP FOR VSE partition
IBBK*xxxx*	IBBLOKs, which contain IP data as it flows through the partition. The variable *xxxx* represents the size
TNBLOK	TN3270 daemon

TCP/IP FOR VSE allocates only the storage that it needs. We recommend that you begin with a minimum partition size of 32 MB. You should then use the QUERY STORAGE command to examine how the storage is being used.

If you are having storage problems and need a detailed explanation of how your storage is being used, contact CSI Technical Support for assistance.

FTP Performance

The basic goal of the FTP mechanism in TCP/IP FOR VSE is to move your data as fast as possible, and we do this without regard to CPU requirements. You can, however, control the performance of your FTP sessions to some extent.

The following factors affect FTP transfer performance:

- Fixed-block sequential files transmit the fastest, followed by VSAM files and then by librarian members. VSE/POWER file transfers and non-blocked sequential files are slowest. When defining VSAM files, a larger number of buffers on the IDCAMS DEFINE statement will result in better performance over a smaller number of buffers.

- Choice of control unit counts.
- Path counts. Do not expect high speed transfers if you are using a slow network or transmitting data over the Internet. Connection-related parameters such as MTU, MSS, and window size can also greatly affect the speed and efficiency of transmission.
- Dispatching priority counts. TCP is a time-dependent protocol. Running your stack at a low priority will result in increased overhead and connection failures.

See the *TCP/IP FOR VSE User Guide* for more information on the FTP protocol and options that may affect FTP performance.

FTPBATCH Performance

FTPBATCH is used to send and receive files from an external partition, and it provides the following performance features:

- All open/close and I/O processing occurs in the FTPBATCH partition, freeing up the TCP/IP stack partition to focus on the network processing.
- Multiple processors can be exploited when using the VSE turbo dispatcher.
- The VSE PRTY command can be used to control its priority and the sharing of CPU resources with other partitions.
- FTPBATCH dynamically attaches its own FTP daemon, so no daemon is needed in the TCP/IP partition for the FTPBATCH job.

Sometimes it is just as important to slow FTP down. For example, assume that you are transmitting a 10GB file in an FTPBATCH job and you do not really care if it takes 10 minutes or 30 minutes. To keep FTP from consuming all of your CPU resources and saturating the network with data, you can slow it down. But first, it might be useful to understand how FTPBATCH normally processes the sending of data to a foreign FTP server.

Normally, and with the default settings, FTPBATCH will work with two transfer buffers to overlap sending data and waiting for data to be acknowledged by the foreign stack. The process is as follows:

1. Fill transfer buffer 1
2. Send transfer buffer 1 to the foreign FTP server
3. Fill transfer buffer 2 (network busy with transfer buffer 1)
4. Send transfer buffer 2 to the foreign FTP server

5. Wait for transfer buffer 1 to be acknowledged by the foreign FTP server

These steps are repeated until all the data has been sent and acknowledged. The goal is to keep the pipe (the network) full. This can cause the FTP processing to saturate the network with data and slow down other interactive IP applications like HTTP and Telnet.

To slow down FTPBATCH, you can add the following command to the SYSIPT JCL immediately after the // EXEC FTPBATCH line:

```
SET SENDWACK ON
```

This command causes all data in the sent buffer to be acknowledged before sending the next transfer buffer. The following behavior results:

1. Fill transfer buffer 1.
2. Send transfer buffer 1 to the foreign server.
3. Wait for transfer buffer 1 to be acknowledged by the foreign FTP server.

These steps are repeated until all the data has been sent and acknowledged.

TN3270 Performance

The following issues apply to TN3270 performance.

Buffer Pools

Each telnet daemon can use either a dedicated buffer or pooled buffers. In dedicated mode, a daemon obtains a 16K buffer when a session is initiated. This buffer is used for all I/O operations throughout the session.

In pool mode, a 16K buffer is obtained from a buffer pool only when it is needed for I/O. Input from the terminal is already completely in memory (in IBBLOK storage) before the I/O buffer is needed. For outbound traffic, a buffer is obtained when VTAM indicates that data is waiting. The buffer is retained for the length of time required to obtain the data from VTAM and to pass it to the socket interface. Thus, one buffer can serve many daemons.

Base your decision about pool mode use on whether you have enough storage in the TCP/IP FOR VSE partition to support dedicated buffer pools. The buffer pools are all allocated in 31-bit storage.

Telnet Daemons

For every telnet daemon you define, TCP/IP FOR VSE allocates and manages internal control blocks that represent the daemon.

Some TN3270 clients automatically reconnect when a session is terminated. This option offers convenience and transparency to the end user, but it may also cause a looping condition if an unforeseen state occurs.

If you read the chapter on configuring TN3270 sessions, you know that TCP/IP FOR VSE offers a number of options for associating IP addresses with LU names. One such option is to use the CONNECT_SEQUENCE command. If you use CONNECT_SEQUENCE=ON, you will experience slightly higher CPU utilization during new session requests.

See chapter 5, "Configuring the Telnet Daemon," page 70, for more information.

Port Queuing

Connection queuing is an important issue that affects application performance. In this section, we describe the TCP/IP FOR VSE port queuing facility and the commands you use to manage and monitor it.

Background

Standard TCP/IP processing is simple. Each connection consists of a series of interactions between two applications that occur in the following order:

1. Application A issues a Passive OPEN (listen).
2. Application B issues an Active OPEN.
3. The applications issue other SENDs and RECEIVEs as appropriate.
4. Each application issues a CLOSE.

In general, the application that issues the passive OPEN (listen) is considered the server and the application issuing the active OPEN is the client. An example is a web server, which issues its listen on the well-known port 80. When a browser (the client) requests a web page, it issues an active OPEN to port 80 and sends its request. Once the server returns a response or rejects the request because of security settings, the connection is closed and the server reissues its listen to await the next client’s request.

In practice, it is difficult to manage connection requests that occur when the server is already processing another request (that is, when there is no listen in effect). This occurs, for example, when two clients try to connect to a server when only one OPEN port is available. The first client to connect will succeed, and the other will fail. The rules that govern the stack are quite explicit and require it to forcefully reject (RESET) any connection request that cannot be paired immediately with an existing listen connection.

To overcome this problem, servers often are configured to maintain multiple listen connections on the same local port. As each passive OPEN completes, the server immediately replaces it with another OPEN. Connection processing can then overlap to any degree allowed. For example, CSI International’s HTTP daemon, which does not implement port queuing, opens several sockets with the same port number. It scans each one until an ECB is posted, indicating that one of the sockets has connected to port 80. At this point, the primary HTTP task spawns a subtask and passes the socket descriptor to it. The subtask then begins to do a SOCKET RECEIVE using this socket descriptor. The main HTTP task, having multiple sockets in wait mode, will allocate another socket, add it to the chain of available sockets, and repeat the process of scanning, waiting, and passing a socket descriptor to a subtask.

Unfortunately, this type of programming is difficult, and it still permits some requests to be dropped because of processing delays at the application level. It is even more difficult to change the number of connection requests accepted (the queuing depth) without modifying the program's code. The port queuing facility addresses these problems.

Queuing Strategy

To use the port queuing facility, you must first determine the port number on which connection requests are to be queued. For example, a web-server application would choose the well-known port 80. Once the port is designated as eligible for queuing, incoming connection requests that cannot be paired with existing listen connections are assigned automatically to a blank listen connection provided by the stack. This connection is negotiated by proxy with a closed window. When the server eventually issues a listen, the next opened-by-proxy connection is assigned, the window is opened to its normal value, and processing continues in the expected manner.

PORTQUEUE Command

Port queuing is enabled, modified, and disabled by the PORTQUEUE command. This command may be executed only after TCP/IP FOR VSE has fully started. To run it from the TCP/IP FOR VSE initialization member, use the command "INCLUDE *lib_member*,DELAY" and place the PORTQUEUE command in *lib_member*. Here is the syntax:

```
PORTQueue PORT=num[,TIMEOUT=sec][,DEPTH=nn]
```

The parameters are described in the following table.

Parameter	Description
PORT=*num*	The port number for which queuing is to be controlled. Valid values range from 1 to 65536. There is no default.
TIMEOUT=*sec*	The amount of time a queued connection is held waiting for the server to reissue a listen. The connection is reset when the wait time exceeds this value. Valid values range from 1 to 60 seconds. If TIMEOUT is not specified, then the value is left unchanged.
DEPTH=*nn*	The maximum number of connections that may be queued at any time. Once this number is reached, additional incoming requests are refused. Valid values range from 0 (no queuing) to 100. If DEPTH is not specified, then the value is not changed. In this case, a value must be set initially to enable port queuing. Any new value overrides the existing setting. To disable port queuing, set DEPTH=0.

Note:

To use port queuing with BSD socket applications, QUEDMAX must be set to 0 (the default) in $SOCKOPT. For more information, see the *TCP/IP FOR VSE Programmer's Guide*, Appendix A: "$SOCKOPT Options Phase."

Query Command

Use the QUERY PORTQUEUE operator command to display queuing settings, as follows:

```
Query PORTQUEUE
```

This command can be issued at any time to display the current queuing parameter settings. In addition, it shows the following information for each port:

- Connection requests queued
- Connection requests accepted
- Connection requests terminated by the requestor
- Connection requests terminated as stale (timed out).

The command also shows the following statistics. This information is useful for adjusting the queuing parameters to achieve best performance.

- Average wait time for successful requests
- Average time before requestor disconnect
- Maximum achieved queuing depth.

Appendix A: Technical Support

Statement of Intent

CSI International wants to set the standard for technical support. To this end, we provide support 24 hours a day, 365 days a year. You can obtain support in a variety of ways, and we are constantly upgrading both our product and our documentation. We welcome all comments and suggestions.

You can send comments on this manual or other TCP/IP FOR VSE documents to documentation@csi-international.com.

Obtaining Support

Where to Go

If you obtained TCP/IP FOR VSE from IBM, request technical support through your normal IBM support channel.

If you are a customer of CSI International in North America, contact CSI International directly. Our technical support number is 800-795-4914.

If you are outside the U.S., obtain first-level support from the agent serving your locale. If you cannot determine how to contact your agent, contact CSI International directly at +1-740-420-5400.

Availability

Our technical support line is answered 24 hours a day, 7 days a week, including holidays.

You are free to call us any time. Our normal work hours are Monday through Friday, 9:00 AM to 5:00 PM Eastern Time. We ask that you present non-critical problems and questions during these times.

Email inquiries are handled promptly.

If you are inquiring about a problem, include as much information as possible—what you were doing, how the system responded, whether any ABENDs occurred, suspicious console messages, and so forth. Please be specific, and always include your telephone number in case we have questions.

How to Contact Us

You can use any of the following methods to contact us.

Email	support@csi-international.com
Internet	http://www.csi-international.com
Telephone	800-795-4914 (U.S. customers); +1-740-420-5400

How to Report a Problem

When you contact technical support, be sure to provide the following:

- Your CSI account number and your company's name
- Your name
- Your telephone number and email address
- Your z/VSE release level
- Your TCP/IP FOR VSE release and service level
- A description of the problem or question

If your question cannot be answered immediately, you will be assigned a problem number. You will need this number if you send correspondence.

Severe or high-impact problems take precedence over minor problems or installation and configuration questions. Although we do not expect to be overloaded with calls at any particular time, this policy assures the best possible support for all customers.

If you need to send documentation, use the following procedure:

1. Go to the z/VSE Problem Report web page at www.csi-international.com/problemreport_vse.htm.
2. Follow the instructions for filling in a problem report, and note the problem number.
3. Send the documentation using the information provided.

Appendix B: Quickstart Guide

System Setup

This appendix outlines the steps to implement a typical TCP/IP FOR VSE installation. It covers initial planning decisions you need to make, the configuration commands you use, and configuration examples.

Control Units

The first major decision you need to make when implementing TCP/IP FOR VSE is control unit selection. We recommend the following units:

- IBM's Open Systems Adapter (OSA)
- Third-party devices that support the LAN Channel Station (LCS) or 3172 protocol
- IBM's OSA Express, also known as "the QDIO Mode of an OSA2"

TCP/IP Network

Next, you must set up a TCP/IP network. If you already have a network in place, ask your network administrator for an IP address, a subnet mask, and appropriate routing information. You need to add this information to your TCP/IP FOR VSE configuration file as outlined in this guide.

If you do not have a TCP/IP network in place, then you have more planning to do. There are many books available on setting up TCP/IP configurations, and you may find a good book helpful at this time. If you have a VSE mainframe and a few PCs that you plan to begin your TCP/IP network with, we recommend using a class C network and IP addresses 192.168.0.*x*, where *x* is a value from 001 to 255, and a subnet mask of 255.255.255.0. You need to consider the physical cabling of your network, for example, each PC needs an Ethernet card with a cable to an Ethernet hub. You also need to assign an IP address to each PC.

You will need additional information to install TCP/IP on other entities in the network, such as a printer or a UNIX machine.

Configuration Commands

Regardless of the device or network you use, some configuration commands are almost universal. The following statements show typical configuration commands that most installations need.

File System

Configure files using the following commands. Each statement is numbered for reference.

```
1 DEFINE FILE,TYPE=LIBRARY,DLBL=IJSYSRS,PUBLIC='IJSYSRS',READONLY
2 DEFINE FILE,TYPE=LIBRARY,DLBL=PRD1,PUBLIC='PRD1',READONLY
3 DEFINE FILE,TYPE=LIBRARY,DLBL=PRD2,PUBLIC='PRD2'
4 DEFINE FILE,TYPE=POWER,PUBLIC='POWER'
```

Each statement is explained below.

1. This statement defines the IJSYSRS library as part of the TCP/IP FOR VSE file system. This file is defined by the DLBL statement IJSYSRS. Users at remote sites access this library by the name IJSYSRS. FTP clients cannot write to this library.
2. This statement defines the PRD1 library as part of the TCP/IP FOR VSE file system. This file is defined by the DLBL statement PRD1. Users at remote sites access this library by the name PRD1. FTP clients cannot write to this library.
3. This statement defines the PRD2 library as part of the TCP/IP FOR VSE file system. This file is defined by the DLBL statement PRD2. Users at remote sites access this library by the name PRD2. FTP clients can write to this library.
4. This statement defines a pseudo file to the TCP/IP FOR VSE file system. Any files written to file name POWER are passed to VSE/POWER. Users should specify the file name with additional qualifiers, such as POWER.LST.A.

Daemons

Configure daemons using these commands:

```
1 DEFINE FTPD,ID=FTPD,COUNT=3
2 DEFINE LPD,PRINTER=VSELST,QUEUE='POWER.LST.A', -
         LIBRARY=PRD2,SUBLIB=TEMPPRT
3 DEFINE TELNETD,ID=TELNET,TARGET=DBDCCICS, -
         TCPAPPL=TCP,COUNT=5
```

Each statement is explained below.

1. This statement defines one File Transfer Protocol (FTP) daemon capable of supporting three concurrent user sessions. This means that three file transfer sessions can operate concurrently.

2. This statement defines a line printer daemon. The queue name that remote users specify when they want to send data to this printer is VSELST. The data is sent to VSE/POWER because this is the public name of a POWER file, as defined in the DEFINE FILE section. Further, the POWER file is qualified to be the LST queue, class A. While the file is being transmitted, it is temporarily stored in the library with the file system public name of PRD2 in subfile TEMPPRT. Note that the DEFINE FILE statement that created the public name cannot have READONLY specified in its definition.

3. This statement generates five telnet daemons. They are named TELNET01, TELNET02, TELNET03, TELNET04, and TELNET05. These names are only used internally and their values can be anything. Each daemon is assigned a VTAM application ID: TCP01, TCP02, TCP03, TCP04, and TCP05. These IDs need to be defined to VTAM. All of these daemons will attempt to connect to the VTAM application DBDCCICS. All of these daemons will monitor TCP/IP port 23, which is the default for telnet.

Addresses

Define addresses using these commands:

```
1 SET IPADDR=192.168.000.001
2 SET MASK=255.255.255.0
```

Each statement is explained below.

1. The IP address must be assigned and set using this parameter. If you are not familiar with TCP/IP and are not connecting to another network or internet, then we suggest that you start by assigning TCP/IP FOR VSE a class C network address (for example, 192.168.0.1). Avoid network and host numbers of zero or all binary ones. These are generally reserved for special purposes.

2. If you use address masking, you must select a value that conforms to the class of IP address assigned in statement two (SET IPADDR). The mask indicates the portion of the IP address that will NOT be used as the host number. To be useful, the mask must include the full network number plus at least one bit. Do not forget to leave room for the host number. A mask of 255.255.255.255 allows for 16,777,215 subnetworks but no hosts. Our suggestion is to define a single subnet of at most 254 hosts in the range of 192.168.000.001 through 192.168.000.254. Of course, if you already have a TCP/IP network in place, you should conform to your existing addressing structure.

   Using a subnet mask is entirely optional. These masks are not communicated outside the local stack. They are used solely as shorthand notation when coding a routing table. See “Using Subnetworking,” page 240, for an example.

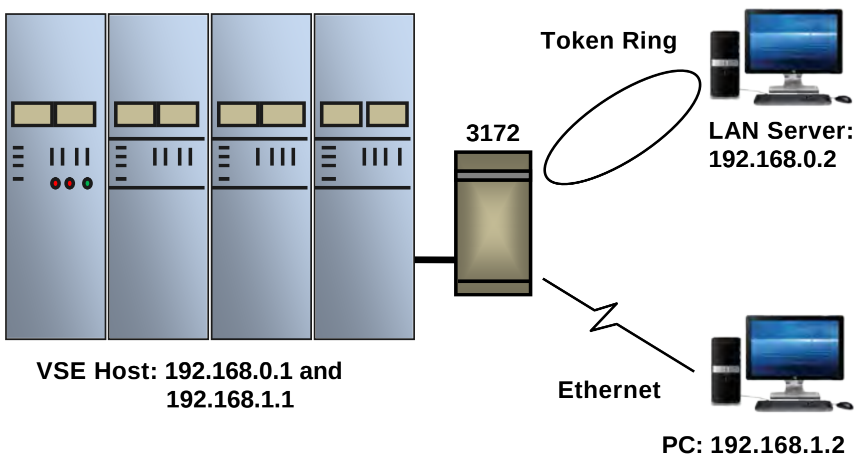
Network-Dependent Configuration

There are a wide number of possible network configurations. This section presents several examples. You should be able to pattern your system after one of these configurations and adjust it to fit.

As a suggestion, try configuring a simple subset of your network. After it is working and you understand the principles, it is easy to expand your configuration to include additional hardware.

Connection Using a 3172 Device

A “3172” device, which includes 8232, LCS, OSA, and OSA2, may contain multiple adapters, each representing a separate and unique network connection. The example below shows a 3172 that contains two adapters, an Ethernet, and a token ring using card slots 0 and 1, respectively. Each adapter is connected to a network with a single PC.



This configuration can be defined using the following commands.

```
1 DEFINE LINK,ID=LINK01,DEV=380,MTU=1500
2 DEFINE ADAPTER,LINKID=LINK01,TYPE=ETHERNET, -
         NUMBER=0,IPADDR=192.168.0.1
3 DEFINE ADAPTER,LINKID=LINK01,TYPE=TOKENRING, -
         NUMBER=1,IPADDR=192.168.1.1
4 DEFINE ROUTE,ID=ROUTE01,LINKID=LINK01,ADAPTER=0, -
         IPADDR=192.168.0.2
5 DEFINE ROUTE,ID=ROUTE02,LINKID=LINK01,ADAPTER=1, -
         IPADDR=192.168.1.2
```

Each statement is explained below.

1. This statement defines link LINK01. It is a 3172 (OSA) controller.
2. This statement defines an Ethernet adapter within the 3172. The LINKID=LINK01 parameter links this statement to the preceding DEFINE LINK statement.

3. This statement defines a token ring adapter within the 3172. The LINKID=LINK01 parameter links this statement to the preceding DEFINE LINK statement.

4. This statement defines route ROUTE01. It specifies the PC's network address of 192.168.0.2 and binds it to a DEFINE LINK (LINK01) statement. The ADAPTER=0 parameter identifies the specific adapter within the 3172.

5. This statement defines route ROUTE02. It specifies the PC's network address of 192.168.1.2 and binds it to a DEFINE LINK (LINK01) statement. The ADAPTER=1 parameter identifies the specific adapter within the 3172.

Using Multiple Devices

This example can easily be extended to multiple devices on each of the two networks by using "zero host" notation. Statements from the previous example could be modified as follows:

```
4 DEFINE ROUTE,ID=ROUTE01,LINKID=LINK01,ADAPTER=0, -
        IPADDR=192.168.0.0
5 DEFINE ROUTE,ID=ROUTE02,LINKID=LINK01,ADAPTER=1, -
        IPADDR=192.168.1.0
```

These statements are explained below.

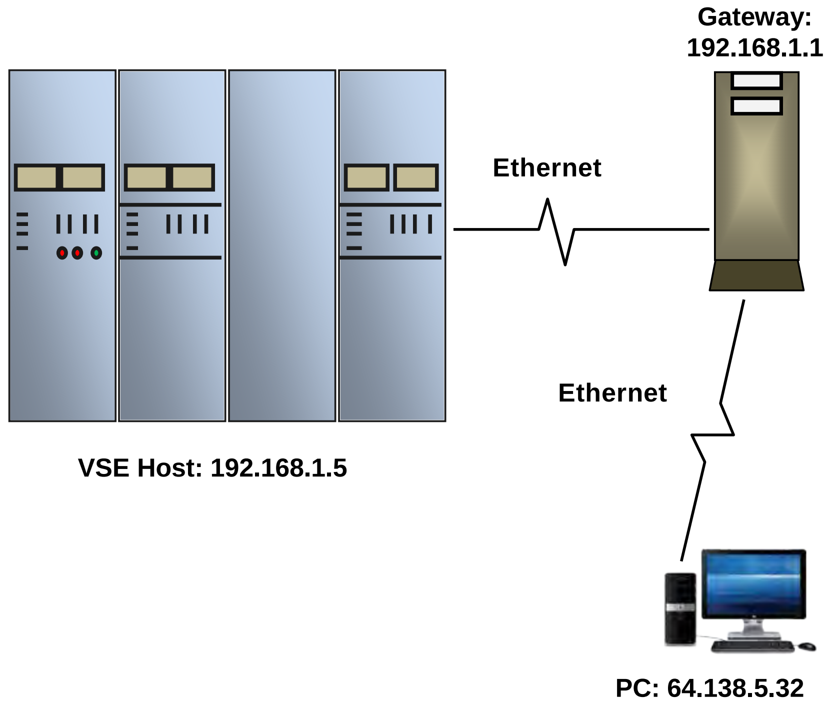
4. Because 192.168.0 and 192.168.1 are both class A networks, only a single byte is available for the host number. Because host number 0 cannot be assigned to a device, we use the value as a generic, representing any host. Thus, coding the zero-host IP address will cause this DEFINE ROUTE to match traffic to any host on the network.

5. This DEFINE ROUTE operates the same as statement 4 but is for network 192.168.1.

You can also use SET MASK and DEFINE MASK to subdivide the host number into a subnet and host. Although subnetwork numbers are defined only for use by the local stack, they can provide useful shortcuts when coding a routing table to support many devices.

Connection Using a Gateway

Few shops will want VSE to only communicate with devices appearing on the local network. To route traffic to any device that is *not* on the local network, for example, over the Internet, it must be routed to a local device called a *gateway*, or router. This is shown in the example below. How the gateway handles further routing is not our concern; we simply need to send it datagrams and it will ensure that they are delivered.



This configuration can be defined using the following commands.

```
1 DEFINE LINK,ID=LINK01,...,IPADDR=192.168.1.5
2 DEFINE ROUTE,ID=LOCAL,LINKID=LINK01,IPADDR=192.168.1.0
3 DEFINE ROUTE,ID=REMOTE,LINKID=LINK01,IPADDR=0.0.0.0, -
        GATEWAY=192.168.1.1
```

These statements are explained below.

1. This statement defines a generic link. The IP address 191.168.1.5 is used to identify all traffic to and from VSE over this link.

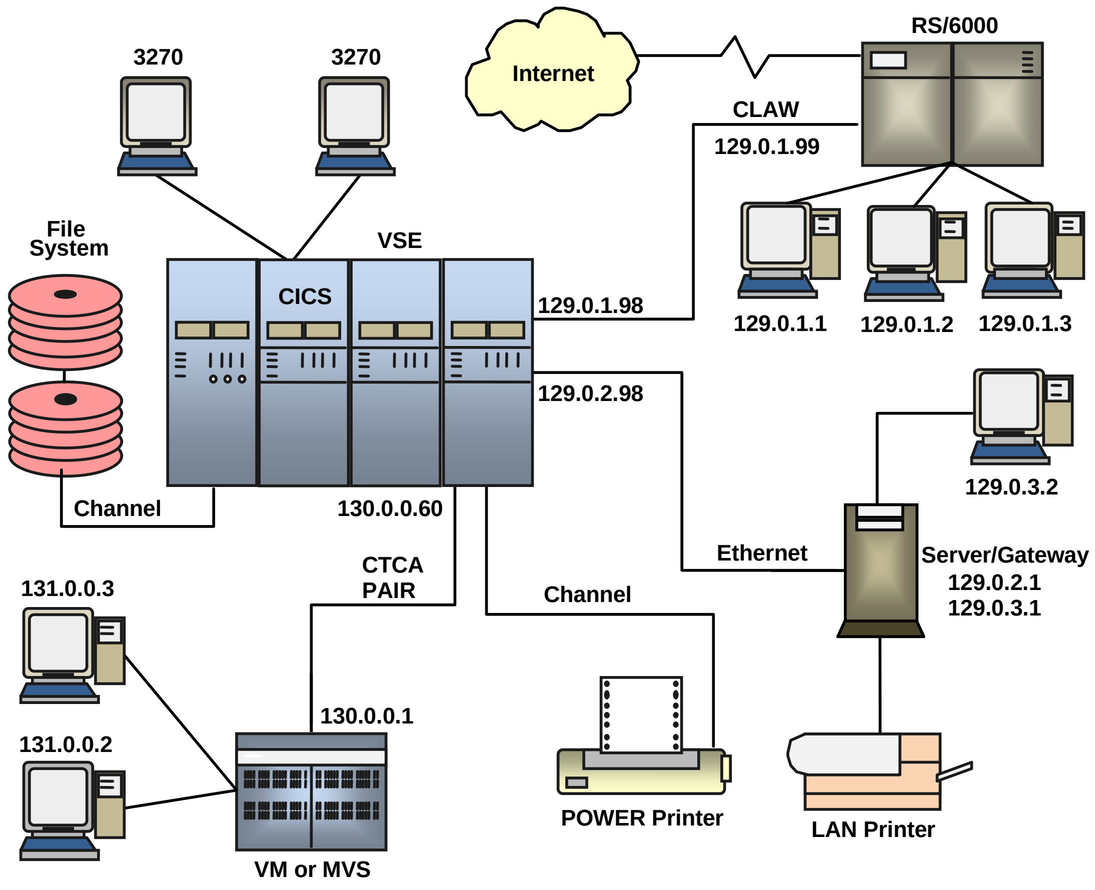
2. This DEFINE ROUTE uses zero-host notation to cause all traffic destined for devices attached to the local network (192.168.1, hosts 1 through 255) to be sent directly to the intended device.

3. Any datagram that does not match the pattern for the local network will match the generic zero-network/zero-host pattern. These are then passed to the gateway device at 192.168.1.1 for further routing.

If this gateway were connected to the Internet, the statements in this routing table would be sufficient to reach any device in the world.

Complex System Example

The following system is a more complex example.



In this system, the mainframe-based VSE host connects to a variety of hardware and software. Here is a description of the components:

- Two physical 3270 terminals are connected to CICS under VSE using an existing SNA network. These terminals can access other network applications using the TCP/IP FOR VSE telnet client.
- Two PCs are connected to a VM (or MVS) host, which in turn is connected to TCP/IP FOR VSE by a CTCA. The VM's TCP/IP is connected to the PCs on network 131. When VM talks to VSE over the CTCA, VSE will see it on network 130. VSE can reach the two PCs by using the VM's stack as a gateway.
- Network access is provided for a VSE/POWER-supported printer and DASD files.
- A PC and a printer are attached to a LAN-based server/gateway. The printer does not have an IP address as it is driven by LPD software in the server. The gateway function passes data between the VSE's Ethernet and the PC's Ethernet.

- A channel-attached RS/6000 processor handles traffic addressed to itself and to the PCs attached to its own local Ethernet. Because the CLAW is a “direct pipe” to the RS/6000, there is no ambiguity when the VSE end and the PCs share a common network number.

The next two sections describe how to configure TCP/IP FOR VSE to support this system.

Basic Command Set

The following command set is the most straightforward way of defining the system described above. Refinements to this command set are described in the next section, “Using Subnetworking.”

```
1  DEFINE FILE,TYPE=LIBRARY,FNAME=IJSYSRS,PUBLIC='IJSYSRS',READONLY
2  DEFINE FILE,TYPE=LIBRARY,FNAME=PRD1,PUBLIC='PRD1',READONLY
3  DEFINE FILE,TYPE=LIBRARY,FNAME=PRD2,PUBLIC='PRD2'
4  DEFINE FILE,TYPE=POWER,PUBLIC='POWER'
5  DEFINE FTPD,ID=FTPD,COUNT=3
6  DEFINE LPD,PRINTER=VSELST,QUEUE='POWER.LST.A', -
          LIBRARY=PRD2,SUBLIB=TEMPPRT
7  DEFINE TELNETD,ID=TELNET,TARGET=DBDCCICS,TCPAPPL=TCP,COUNT=5
8  SET IPADDR=130.0.0.60
9  DEFINE LINK,ID=VM,TYPE=CTCA,DEV=024,MTU=1500,IPADDR=130.0.0.60
10 DEFINE LINK,ID=LAN,TYPE=ETHERNET,DEV=220,MTU=1500, -
          IPADDR=129.0.2.98
11 DEFINE LINK,ID=UNIX,TYPE=CLAW,DEV=040,MTU=1500,IPADDR=129.0.1.98
12 DEFINE ROUTE,ID=RTVM1,LINKID=VM,IPADDR=130.0.0.1
13 DEFINE ROUTE,ID=RTVM2,LINKID=VM,IPADDR=131.0.0.3
14 DEFINE ROUTE,ID=RTVM3,LINKID=VM,IPADDR=131.0.0.2
15 DEFINE ROUTE,ID=RTUX1,LINKID=UNIX,IPADDR=129.0.1.99
16 DEFINE ROUTE,ID=RTUX2,LINKID=UNIX,IPADDR=129.0.1.1
17 DEFINE ROUTE,ID=RTUX3,LINKID=UNIX,IPADDR=129.0.1.2
18 DEFINE ROUTE,ID=RTUX4,LINKID=UNIX,IPADDR=129.0.1.3
19 DEFINE ROUTE,ID=RTLAN1,LINKID=LAN,IPADDR=129.0.2.1
20 DEFINE ROUTE,ID=RTLAN2,LINKID=LAN,IPADDR=129.0.3.2, -
          GATEWAY=129.0.2.1
21 DEFINE ROUTE,ID=RTNET,LINKID=UNIX,IPADDR=0.0.0.0
```

The file system defined in statements 1 through 4 is available to all clients connecting through TCP/IP, regardless of routing.

Each statement in this command set is explained below.

1. This statement defines library IJSYSRS as part of the TCP/IP FOR VSE file system. This file is defined by DLBL statement IJSYSRS. Users at remote sites access this library by the name IJSYSRS. FTP clients cannot write to this library.

2. This statement defines library PRD1 as part of the TCP/IP FOR VSE file system. This file is defined by DLBL statement PRD1. Users at remote sites access this library by the name PRD1. FTP clients cannot write to this library.

3. This statement defines library PRD2 as part of the TCP/IP FOR VSE file system. This file is defined by DLBL statement PRD2. Users at remote sites access this library by the name PRD2. FTP clients can write to this library.

4. This statement defines a pseudo file to the TCP/IP FOR VSE file system. Any files written to file name POWER are passed to VSE/POWER. Users specify the file name with additional qualifiers, such as POWER.LST.A.

5. This statement defines a File Transfer Protocol (FTP) daemon capable of supporting three concurrent user sessions. This means that three file-transfer sessions can operate concurrently.

6. This statement defines a line printer daemon. The public name that remote users specify to send data to this virtual printer is VSELST. The data is sent to VSE/POWER because this is the public name of a POWER file, as defined in the DEFINE FILE section. Further, the POWER file is qualified to be the LST queue, class A.

   While the file is being transmitted, it is temporarily stored in the library with the file system public name of PRD2, in subfile TEMPPRT. Note that the DEFINE FILE statement that created the public name may have READONLY in its definition. If so, then this parameter pertains only to FTP access.

7. This statement generates five telnet daemons. They are named TELNET01, TELNET02, TELNET03, TELNET04, and TELNET05. These names are only used internally and their values can be anything. Each daemon is assigned a VTAM application ID of TCP01, TCP02, TCP03, TCP04, and TCP05. These IDs need to be defined to VTAM. All of these daemons attempt to connect with the VTAM application DBDCCICS. All of these daemons monitor TCP/IP port 23, the default for telnet, for access requests.

8. A default IP address must be assigned and set by this parameter. Because our example shows multiple network connections requiring multiple IP addresses (multi-homing), we recommend that you include a unique IP address for each link or adapter and not rely on the default value.

Statements 9 through 11, described below, define the links that TCP/IP FOR VSE uses to connect to other TCP/IP implementations.

9. This statement defines a CTC link to the VM (or MVS) system. The MTU size of 1500 was chosen to be compatible with VM and all other devices that we intend to reach from VM.

10. This statement defines a connection to a local Ethernet that is populated by a single server/gateway. In practice, multiple servers, gateways, and other devices will share the same Ethernet segment. The MTU size of 1500 is the standard maximum for an Ethernet network.

11. This statement defines a channel connection to an RS/6000 UNIX processor using a CLAW interface. In addition to the PCs shown, a variety of other workstations and printers may be present and addressable. In this example, the RS/6000 is also connected to the Internet in an unspecified way.

    Note: If the VSE stack needs to be visible to the Internet, the IP address used for the link *must* be an address provided by your Internet service provider (ISP). This requirement does not apply to the rare circumstance of connecting through a gateway that provides network address translation (NAT).

Statements 12 through 21, described below, define the routes that are used to reach the various devices. In this example, each device requires a ROUTE statement. The next section, "Using Subnetworking," explains how to use subnetworking as a shortcut to these steps.

12. Arbitrary route name RTVM1 is assigned to direct traffic destined for the VM stack to the CTCA link.

13. Route name RTVM2 directs traffic for the specified device to the CTCA link connected to the VM stack. Once VM receives the traffic, it uses its own routing tables to forward the messages to their ultimate destination.

14. Route name RTVM3 is used similarly to RTVM2 but for IP address 131.0.0.2.

15. Arbitrary route name RTUX1 selects traffic for RS/6000-based applications and sends it through the CLAW interface.

16. Route name RTUX2 causes anything addressed to the specified address to be passed to the RS/6000 via the CLAW. Once there, the RS/6000 continues the routing process and sends the data on its way. Note that we have not included a "GATEWAY=" parameter because the RS/6000 is the only destination directly reachable using the CLAW, and that stack must inspect all traffic.

17. Route name RTUX3 is similar to RTUX2.

18. Route name RTUX4 is similar to RTUX2.

19. Route name RTLAN1 is selected for messages destined only for the device with IP address 129.0.2.1. These messages are directed to the Ethernet adapter and then directly to the device located on the local Ethernet segment. Some of these messages may involve using the attached printer.

20. Route name RTLAN2 is selected for messages destined only for the device with IP address 129.0.3.2. These messages are directed to the Ethernet adapter and then passed to the device identified by the "GATEWAY=" parameter. The gateway is then responsible for sending the messages on to their ultimate destination.

21. The final route name, RTNET, is assigned to the CLAW device. The address 0.0.0.0 is a special form that matches any and all addresses not otherwise present in the routing table. Messages handled by this statement are sent to the RS/6000 using the CLAW. Once there, the RS/6000 routes them (presumably) to other networks.

Using Subnetworking

You can use subnetworking to configure routing with fewer commands. The following numbered commands replace statements 12 through 21 in the basic command set above.

```
12 SET MASK=255.255.255.0
13 DEFINE ROUTE,ID=RTVM1,LINKID=VM,IPADDR=130.0.0.1, -
          MTU=32768
14 DEFINE ROUTE,ID=RTVMX,LINKID=VM,IPADDR=131.0.0.0
15 DEFINE ROUTE,ID=RTUX1,LINKID=UNIX,IPADDR=129.0.1.99, -
          MTU=32768
16 DEFINE ROUTE,ID=RTLAN1,LINKID=LAN,IPADDR=129.0.2.0
17 DEFINE ROUTE,ID=RTLANX,LINKID=LAN,IPADDR=129.0.3.0, -
          GATEWAY=129.0.2.1
18 DEFINE ROUTE,ID=ALL,LINKID=UNIX,IPADDR=0.0.0.0
```

Each statement in this command set is explained below.

12. The SET MASK statement establishes a mask value that is applied to each network address to obtain the subnetwork number. When applying the mask, the network number portion of the address is exempt, although by convention the mask includes it. The masked portion of the host number is considered to be the subnetwork number with the unmasked portion used as the host. However you define and use the mask, remember that it is known only to the VSE stack and is used solely for interpreting the route table.

13. Route RTVM1 specifically matches the full address of the VM host. We include this route statement so that we can override the MTU address and specify the maximum that the destination supports.

14. Route RTVMX is defined as all devices with a network number of 131. This is because the subnet number of 0 matches all subnets of 130, and a host number of 0 matches all hosts of the specified network. We do not override the MTU size because the destinations may require traversing network segments that do not support a larger size.

15. Route RTUX1 is a specific match to reach applications and/or servers that run on the RS/6000. This allows us to selectively increase the MTU size to the maximum for the link.

16. Route RTLAN1 handles routing to all devices on the local Ethernet segment of the LAN adapter, which is subnet 3 of network 129.

17. Traffic destined for any address of the form 129.0.3.*xxx* is sent to the specified gateway, which should then route it to the correct device.

18. Finally, any address that does not match one of the preceding routes always matches the final generic address of 0.0.0.0. In this case, anything that we have not provided for (and there are millions of addresses we have not considered) is sent to the RX/6000 for additional routing.

Selecting Other TCP/IP Software

Depending on the TCP/IP hosts in your network, you may need to obtain partner TCP/IP applications. The available applications vary by platform. The products described below have been tested with TCP/IP FOR VSE.

The appearance of a product in the list does not imply an endorsement. Also, the fact that a product is not listed does not mean that it does not work or has not been tested.

Microsoft Windows®

The following table lists Windows components.

Application	Description / Where to Find It
TCP/IP Stack	Included.
FTP Client (line mode)	Included.
FTP Client (graphical)	WS_FTP at https://www.ipswitch.com/secure-information-and-file-transfer/wsftp-client.
FTP Daemon	This is needed to run batch jobs on VSE that FTP files to the Windows platform. FTP daemons are available from full-function TCP/IP suite products, such as Hummingbird, or a product such as WFTPD.
TN3270 Client	This is needed to connect to TCP/IP FOR VSE's TN3270 daemon. You can obtain full-product TN3270 clients such as PASSPORT or shareware TN3270 clients such as TN3270 Plus from www.sdisw.com. Another choice is QWS3270 from www.jollygiant.com.
Telnet Daemon	This is needed to issue DOS commands from a VSE batch job, a VSE REXX program, or CICS. Several telnet daemons are available as shareware.
LPR Client and Line Printer Daemon (LPD)	These are available from full-function TCP/IP suite products or as shareware.

UNIX

The following table lists UNIX components. There are several varieties of UNIX.

Application	Description / Where to Find
TCP/IP Stack	Included.
FTP Client (line mode)	Included.
FTP Client (graphical)	Depends on the UNIX variant you are using.

Application	Description / Where to Find
FTP Daemon	This is needed to run batch jobs on a VSE that uses FTP to send files to the UNIX platform. All UNIX machines have FTP daemons.
TN3270 Client	Depends on the UNIX variant you are using. AIX and Linux both have TN3270 clients.
Telnet Daemon	This is needed to issue UNIX commands or shell scripts from a VSE batch job, a VSE REXX program, or CICS. All UNIX machines have telnet daemons.
LPR Client and Line Printer Daemon (LPD)	Included.

AS/400 with OS/400 Version 4.3 (iSeries)

The following table lists AS/400 components.

Application	Description / Where to Find
TCP/IP Stack	Included.
FTP Client (line mode)	Included for most OS/400 file types.
FTP Client (graphical)	Not available.
FTP Daemon	Included for most OS/400 file types.
TN3270 Client	Included.
Telnet Daemon	Included in both full screen and NVT (network virtual terminal) mode. Unfortunately, NVT mode is very limited and does not allow you to interface easily with the OS/400 command language.
LPR Client and Line Printer Daemon (LPD)	Included.

For more information on IBM products, see the web page https://www.ibm.com/support/knowledgecenter/.