Hxx4 Secure NetEx/IP™

Release 1.0

Software Reference Manual

MAN-REF-Hxx4-1.0

Revision Record

Revision	Description
01	Manual released.

You may submit written comments to:

Network Executive Software, Inc.
Publications Department
6450 Wedgwood Road N Suite 103
Maple Grove, MN 55311
USA

Comments may also be submitted over the Internet by addressing e-mail to:

support@netex.com

or, by visiting our web site at:

http://www.netex.com

Always include the complete title of the document with your comments.

Preface

This manual describes the Secure NetEx/IP™ software for supported operating systems.

The currently supported operating systems and specific Secure NetEx/IP™ products are:

H214 for the IBM z/OS operating systems on IBM z Systems

H304 for the Unisys OS2200 on Dorado platforms

H804 for the Linux/Oracle Linux operating system on x86 platforms

H624 for the IBM AIX operating systems on IBM Power Systems

"Chapter 1: Introduction", "Chapter 2: Secure NetEx/IP and the ISO Model", "Chapter 3: Secure NetEx/IP Session Services", and "Chapter 4: NetEx Request Block" are intended for all readers. "Chapter 5: C High Level Interface" describes the library of subroutines that are called by the C high-level language programs.

"Appendix A: NRB Error Codes" includes a list and description of the error messages and codes issued by Secure NetEx/IP.

The remaining appendices are installation and configuration instructions for each platform.

Readers are not expected to be familiar with Secure NetEx/IP before using this manual. However, an understanding of programming and using the host operating system is required.

Notice to the Reader

The material contained in this publication is for informational purposes only and is subject to change without notice. Network Executive Software is not responsible for the use of any product options or features that are not described in this publication, and assumes no responsibility for any errors that may appear in this publication. Refer to the revision record (at the beginning of this document) to determine the revision level of this publication.

Network Executive Software does not by publication of the descriptions and technical documentation contained herein, grant a license to make, have made, use, sell, sublicense, or lease any equipment or programs designed or constructed in accordance with this information.

This document may contain references to the trademarks of the following corporations:

Corporation Trademarks and Products

Network Executive Software	**NetEx, Secure NetEx/IP, BFX,**
The Open Group	**UNIX**
IBM Corporation	**IBM, AIX, z System, Power Systems, z/OS, GSKit**
Linus Torvalds	**Linux**
Unisys Corporation	**Dorado, OS2200**
Oracle Corporation	**Oracle**

These references are made for informational purposes only.

The diagnostic tools and programs described in this manual are **not** part of the products described.

Notice to the Customer

Installation information contained in this document is intended for use by experienced System Programmers.

Document Conventions

The following notational conventions are used in this document.

Format	Description
displayed information	Information displayed on a CRT (or printed) is shown in this font.
user entry	*This font* is used to indicate the information to be entered by the user.
UPPERCASE	The exact form of a keyword that is not case-sensitive or is issued in uppercase.
MIXedcase	The exact form of a keyword that is not case-sensitive or is issued in uppercase, with the minimum spelling shown in uppercase.
bold	The exact form of a keyword that is case-sensitive and all or part of it must be issued in lowercase.
lowercase	A user-supplied name or string.
value	Underlined parameters or options are defaults.
<label>	The label of a key appearing on a keyboard. If "label" is in uppercase, it matches the label on the key (for example: <ENTER>). If "label" is in lowercase, it describes the label on the key (for example: <up-arrow>).
<key1><key2>	Two keys to be pressed simultaneously.
No delimiter	Required keyword/parameter.

Glossary

asynchronous: A class of data transmission service whereby all requests for service contend for a pool of dynamically allocated ring bandwidth and response time.

ASCII: Acronym for American National Standard Code for Information Interchange.

BFX: Bulk File Transfer; Network Executive family of file transfer products.

buffer: A contiguous block of memory allocated for temporary storage of information in performing I/O operations. Data is saved in a predetermined format. Data may be written into or read from the buffers.

code conversion: An optional feature in Secure NetEx/IP that dynamically converts the user data from one character set to another (for example, ASCII, EBCDIC, et cetera).

GSKit: The Global Security Kit (GSKit) is a required component for the Secure Socket Layer (SSL) enablement for Secure NetEx/IP on IBM zOS.

header: A collection of control information transmitted at the beginning of a message, segment, datagram, packet, or block of data.

host: A data processing system that is connected to the network and with which devices on the network communicate. In the context of Internet Protocol (IP), a host is any addressable node on the network; an IP router has more than one host address.

Internet Protocol (IP): A protocol suite operating within the Internet as defined by the *Requests For Comment* (RFC). This may also refer to the network layer (level 3) of this protocol stack (the layer concerned with routing datagrams from network to network).

ISO: Acronym for International Standards Organization.

link: (1) A joining of any kind of IP networks. (2) The communications facility used to interconnect two trunks/busses on a network.

Secure NETwork Executive/IP (NetEx/IP): A family of software designed to enable two or more application programs on heterogeneous host systems to privately communicate. Secure NetEx/IP is tailored to each supported operating system, but can communicate with any other supported Secure NetEx/IP, regardless of operating system.

NetEx is a registered trademark of Network Executive Software.

Open Systems Interconnection (OSI): A seven-layer protocol stack defining a model for communications among components (computers, devices, people, etcetera) of a distributed network. OSI was defined by the ISO.

Secure Socket Layer (SSL): A is a standard security technology for establishing an encrypted link between a server and a client

TCP/IP: An acronym for Transmission Control Protocol/Internet Protocol. These communication protocols provide the mechanism for inter-network communications, especially on the Internet. The protocols are hardware-independent. They are described and updated through *Requests For Comment* (RFC). IP corresponds to the OSI network layer 3, TCP to layers 4 and 5.

Contents

Figures

Tables

Chapter 1: Introduction

Network Executive Software's Secure NetEx/IP™ allows two or more application programs (which may be on different host computers) to privately communicate with each other at multi-megabit speeds. The Secure NetEx/IP family of software consists of different versions of Secure NetEx/IP for use with different operating systems, such as the versions for use with the various UNIX, zOS or OS2200 operating system hosts. All of these versions provide a common high-level interface to simplify programming requirements. Secure NetEx/IP utility programs are also available, such as the Secure Bulk File Transfer (BFX™).

Secure NetEx/IP Characteristics

Secure NetEx/IP centralizes network considerations for IP networks, into a single piece of software. The following sections describe the characteristics of the Secure NetEx/IP software.

- External interface
- Internal interaction
- Secure NetEx/IP connections
- Design flow efficiency and flexibility
- Basic I/O flow

External Interface

The Secure NetEx/IP external interface for the application programmer is common for all versions of Secure NetEx/IP. Secure NetEx/IP provides the same requests for use in the programs that call NetEx/IP. *However, Secure NetEx/IP is not interoperable with NetEx/IP.* The calling programs may be written in C or other high-level languages. Secure NetEx/IP for zOS also has an assembler program interface. Secure NetEx/IP programs written in high-level languages may be transported from one host to another, with some changes to account for different word sizes and other machine architecture variations.

Internal Interaction

The internal operations of all supported versions of Secure NetEx/IP are consistent and allow the different versions to interact freely. Thus, any program using Secure NetEx/IP may communicate with any other program on the network that is also using Secure NetEx/IP. When a Secure NetEx application initiates a session (offer/connect), Secure NetEx/IP utilizes secure open system services of the TCP stack to establish the Secure NetEx/IP session and privately transfer data between Secure NetEx/IP nodes. The default port used in the Secure NetEx/IP network is TCP 3919. Configured ephemeral ports are also used for the data transfer.

Note: Secure NetEx/IP's listen port (TCP 3919) and the configured ephemeral ports must be allowed through firewalls between Secure NetEx/IP nodes.

To facilitate communication between hosts of different manufacture, Secure NetEx/IP supports code conversion. Secure NetEx/IP software can perform code conversion.

Secure NetEx/IP Connections

To communicate using Secure NetEx/IP, two calling programs first form a private connection using a handshake protocol. Secure NetEx/IP then allows this pair of programs to communicate.

Secure NetEx/IP can establish multiple private connections at one time, and can allow one program to have multiple private connections simultaneously.

Secure NetEx/IP also supports private communications within a single host. A calling program may connect to another calling program in the same host and exchange information just as if network communications were taking place.

Design Efficiency and Flexibility

The Secure NetEx/IP design enables many applications on the same processor to share the use of the network facility. Programs calling Secure NetEx/IP can be written without regard to the other programs calling Secure NetEx/IP.

Once Secure NetEx/IP accepts data from the caller, Secure NetEx/IP must deliver the data to its destination. The Secure NetEx/IP on each host handles code conversion, flow control, and error recovery.

Basic I/O Flow

Figure 1 shows the basic I/O flow between two programs using Secure NetEx/IP. The calling program communicates with Secure NetEx/IP through the Secure NetEx/IP user interface. Secure NetEx/IP then uses the available system services and network hardware to communicate with the calling program on the other processor.

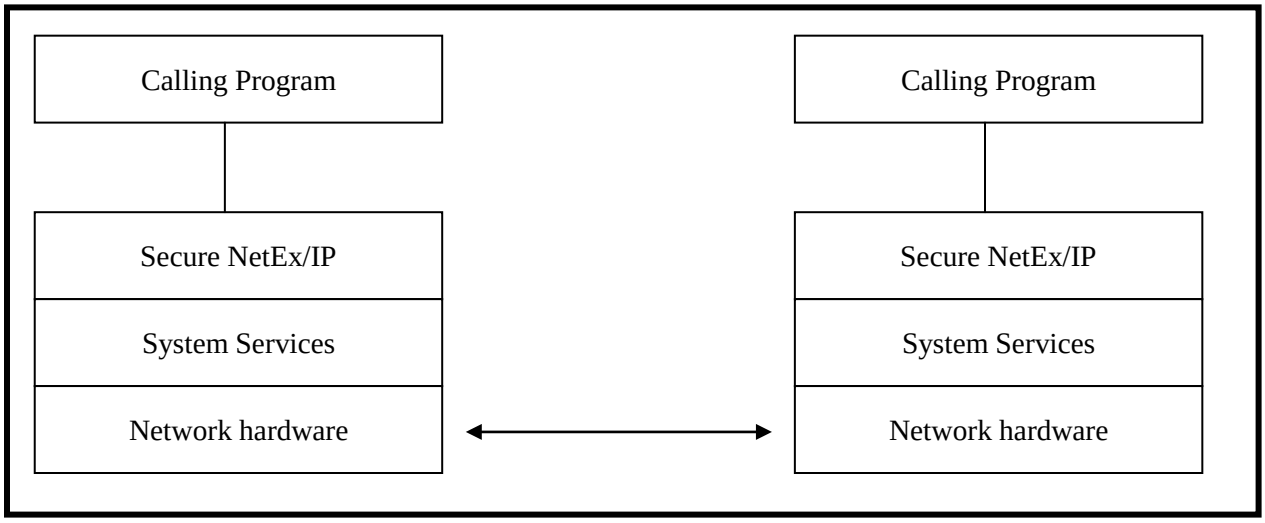


Figure 1. Basic I/O Flow

Secure NetEx/IP

Secure NetEx/IP consists of the user interface or API (linked with the user program) and SNXMAP, which is a separate program residing in each machine that Secure NetEx/IP users call on to perform common services for related to the Secure NetEx connections. Configuration of Secure NetEx/IP is done via the SNXMAP and its configuration file (snxmap.cfg). Each platform may have some unique configuration parameters. To understand them for your platform, refer to the appropriate installation section found in the Appendices and to Appendix G: Secure NetEx/IP Configuration File on page 135.

Chapter 2: Secure NetEx/IP and the ISO Model

In creating Secure NetEx/IP, Network Executive Software followed the guidelines set by the International Standards Organization (ISO) for Open Systems Interconnection (OSI). Open Systems Interconnection refers to the exchange of information among terminal devices, computers, people, and networks, that are open to communication with one another.

The ISO model is composed of seven layers. Each layer interacts only with adjacent layers in the model (see Table 1). By using this modular structure, the internal function of each layer is self-contained and does not affect the functioning of other layers.

Table 1. ISO Model as Implemented

Layer	**Major Functions**
Application	High level description of data to be privately transferred and the destination involved
Presentation	Select data formats and syntax
Session	Establish a private session connection, report exceptions, and select routing
Transport	Manage data transfer and provide Secure NetEx/IP-to- Secure NetEx/IP message delivery
Network	Point-to-point transfer, error detection, and error recovery
Data Link	Data link connection, error checking, and protocols
Physical	Mechanical and electrical protocols and interfaces

Although each layer physically interacts only with adjacent layers, each layer appears to communicate directly with the corresponding layer of the other model. Figure 2 illustrates this concept.

Application	←----------→	Application
Presentation	←----------→	Presentation
Session	←----------→	Session
Transport	←----------→	Transport
Network	←----------→	Network
Data Link	←----------→	Data Link
Physical	←----------→	Physical
Physical Media (IP)		

Figure 2. ISO Model Communication

Note: The corresponding layers appear to communicate directly as indicated by the lines with arrows, but actually they communicate only by progressing down through the layers of one model, through the physical media, and up through the layers of the other model.

Session Layer Services

At the highest layer within Secure NetEx/IP (referring to the ISO model) the session layer software provides the general interface to the user's application/utility program. The Secure NetEx/IP session layer services include: program-to-program connection using the best available network path, reading data, writing data, and disconnection. The user requests these services using a standard Secure NetEx/IP Request Block (NRB) (containing parameters), and the Secure NetEx/IP requests described in "Secure NetEx/IP Session Services". The session layer software implements user requests by requesting services from the underlying layer.

Chapter 3: Secure NetEx/IP Session Services

The user interface to Secure NetEx/IP is a library that provides the user with access to the session functions of Secure NetEx/IP.

To communicate using the session layer of Secure NetEx/IP, the calling programs (that is, the programs that are calling Secure NetEx/IP) must utilize the user interface library to first establish a session connection. Once the session is established, private data transfer may take place in a variety of ways, depending on the needs of the calling programs. Once Secure NetEx/IP accepts data, the user is assured of its delivery (with the possible exception of catastrophic failures – for example, a machine going down or a major problem with the communication line). Sessions may be terminated by either of the parties at any time, although this should be done by mutual agreement.

This chapter explains the concepts of session layer inter-task communication using Secure NetEx/IP. The following topics are discussed in this section:

Session layer requests

General concept of a session

Establishing a private session

Data transfer process

Terminating a session

Handling multiple connections

Error Recovery procedures

Code Conversion

Session LayerRequests

There are eight requests used by programs to call Secure NetEx/IP at the session level. These requests must be issued in a logical order according to rules described in the following paragraphs. These requests and a control structure called the NetEx Request Block (NRB) are the programmer's interface to Secure NetEx/IP. The NRB is updated by the calling program when the program issues requests (either directly or via Secure NetEx/IP), and it is updated by Secure NetEx/IP when requests are completed by Secure NetEx/IP. The NRB is described in the chapter "Chapter 4: NetEx Request Block".

The Secure NetEx/IP session requests, listed in the approximate order in which they are issued, are as follows.

SOFFER

This request makes a calling program using Secure NetEx/IP available to another program on either a remote or local host via a secure connection.

NSOFFER

This request makes a calling program using Secure NetEx/IP available to another program on either a remote or local host via a non-secure connection.

SCONNECT

This requests a secure/private session with a calling program that previously issued an SOFFER. The program may insert values defining characteristics of the upcoming session in the NRB when this re-

quest is issued. This request may also write data to the offering program provided this data does not exceed the maximum size specified on either the sending or receiving application.

NSCONNECT

This requests a non-secure session with a calling program that previously issued an NSOFFER. The program may insert values defining characteristics of the upcoming session in the NRB when this request is issued. This request may also write data to the offering program provided this data does not exceed the maximum size specified on either the sending or receiving application.

SCONFIRM

This request is the last step to establish the session. The host which initially issued the SOFFER replies to a SCONNECT with the SCONFIRM request if the characteristics of the session (for example, data block size, etcetera) specified in the SCONNECT are accepted. This request may also write data to the connecting program provided this data does not exceed the maximum size specified on either the sending or receiving application.

SWRITE

This request sends data to the other program. The SWRITE request may only be used after a session has been properly established. The SWRITE request is an asynchronous service (the user is free to continue when Secure NetEx/IP accepts the SWRITE). A datamode may be specified to invoke code conversion and data assembly or disassembly if supported by the platform.

SREAD

This request receives data that has been written by another program. The SREAD request is also used to accept indicators such as SCONFIRM, and DISCONNECT. The SREAD request is an asynchronous service, so the user may continue when Secure NetEx/IP accepts the SREAD Request.

SWAIT

This request is specified as part of a request or as a separate request, SWAIT suspends processing on the issuing program until Secure NetEx/IP completes the specified request(s). For SWRITE, SCLOSE, SCONNECT, SCONFIRM, or SDISCONNECT requests, Secure NetEx/IP accepts the request quickly and the issuing program can consider the request complete. For SREAD and SOFFER requests, the request does not complete until the other program issues a SWRITE, SCLOSE, SCONNECT, SCONFIRM, or SDISCONNECT request.

SCLOSE

This request is the last write operation for a connection. The SCLOSE request is issued to gracefully terminate a connection. It may contain data just like a WRITE request, but it also indicates that the sender is ready to terminate the connection. After the SCLOSE is issued, incoming data may continue to be read, but no other messages may be written. When the other party in the connection issues an SCLOSE, the session is terminated.

SDISCONNECT

This command immediately terminates a connected session. The SDISCONNECT request may be issued by either program at any time.

These requests are used during the session as described in the following paragraphs.

General Concept of a Session

Before explaining in detail how each part of a session is programmed, the next paragraph explains the general concept of a simple Secure NetEx/IP session.

Figure 1 shows a sample system configuration. Figure 3 is a simplified example of a session where one program reads data from another.

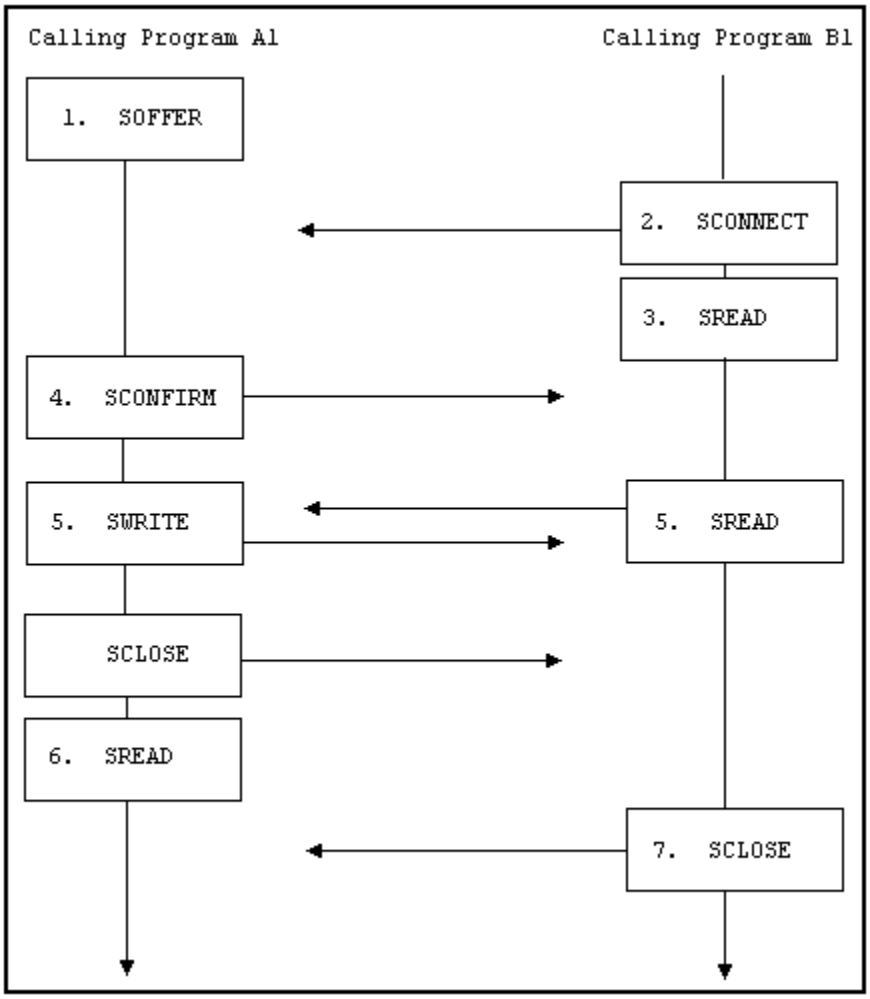


Figure 3. Simplified Session Example

The following text describes the session flow shown here in Figure 3:

1. Calling program A1 in host A issues an SOFFER (or NSOFFER) indicating that it is available to service other calling programs.
2. Calling program B1 requires a file controlled by program A1. To initiate a private data transfer session, program B1 issues an SCONNECT (or NSCONNECT) request. This request may contain data to be delivered to the offering program. (NSOFFER is only paired to NSCONNECT and SOFFER is only paired to SCONNECT.)
3. When the SCONNECT/NSCONNECT completes (is accepted by Secure NetEx/IP), program B1 issues an SREAD to prepare to receive program A1's response to the SCONNECT/NSCONNECT. (Since the SCONNECT/NSCONNECT can also transport data, program B1 could issue an SDISCONNECT and terminate the session immediately. In that case, B1 would not know the status of the data transmitted.)

4. When the SCONNECT/NSCONNECT is received by the Secure NetEx/IP in A1's host, the SOFFER/NSOFFER completes with a Connect Indication and with B1's SCONNECT data in the buffer associated with A1's SOFFER/NSOFFER. If program A1 finds the conditions associated with the SCONNECT/NSCONNECT acceptable, it responds by returning an SCONFIRM request.

   If the program A1 finds any conditions associated with the SCONNECT/NSCONNECT to be unacceptable, it would SDISCONNECT the session and may return data specifying the reason for the SDISCONNECT. For example, if one program determines the other program does not have the proper security code for data in that database, the session may be disconnected (SDISCONNECT).

   The SREAD that program B1 previously issued now indicated program A1's response. If program A1 issued an SCONFIRM, the SREAD will show a Confirm indication along with the data sent with the SCONFIRM. If program A1 issued an SDISCONNECT, the SREAD will complete with a Disconnect indication.

5. The programs may now begin the (private) data transfer. Program B1 issues an SREAD to prepare to receive data from program A1. Program A1 writes data to program B1. The SWRITE command is used until the last data is written. An SCLOSE is used to write the last data. Since we are only issuing one write request in this example, the SCLOSE is used.

6. After completion of the last data transfer, program A1 issues an SREAD to detect program B1's next request.

7. Program B1 issues an SCLOSE and the session is terminated. Both programs may perform disconnect functions (for example, closing files, etcetera). Program A1 could then SOFFER itself as ready for another session.

This is a simplified example of a session. The following sections describe how to program Secure NetEx/IP by describing (in detail) establishing a session, data transfer and terminating a session.

Data Transfer

Unlike rules for establishing a session, Secure NetEx/IP provides a great deal of flexibility in how session requests are used for the data transfer process. The following paragraphs describe two methods for programming data transfer. The first method is a basic data transfer technique; the second uses the more advanced technique of concurrent SWRITE and SREAD requests.

Write/Read Data Transfer

The following paragraphs describe the session requests that are used to transfer data in a straight-forward way. In general, calling programs use the SREAD and SWRITE requests to perform the data transfer. Figure 4 shows how the calling programs perform the data transfer. SWRITE requests are issued by one program which must be received by an SREAD issued by the other program.

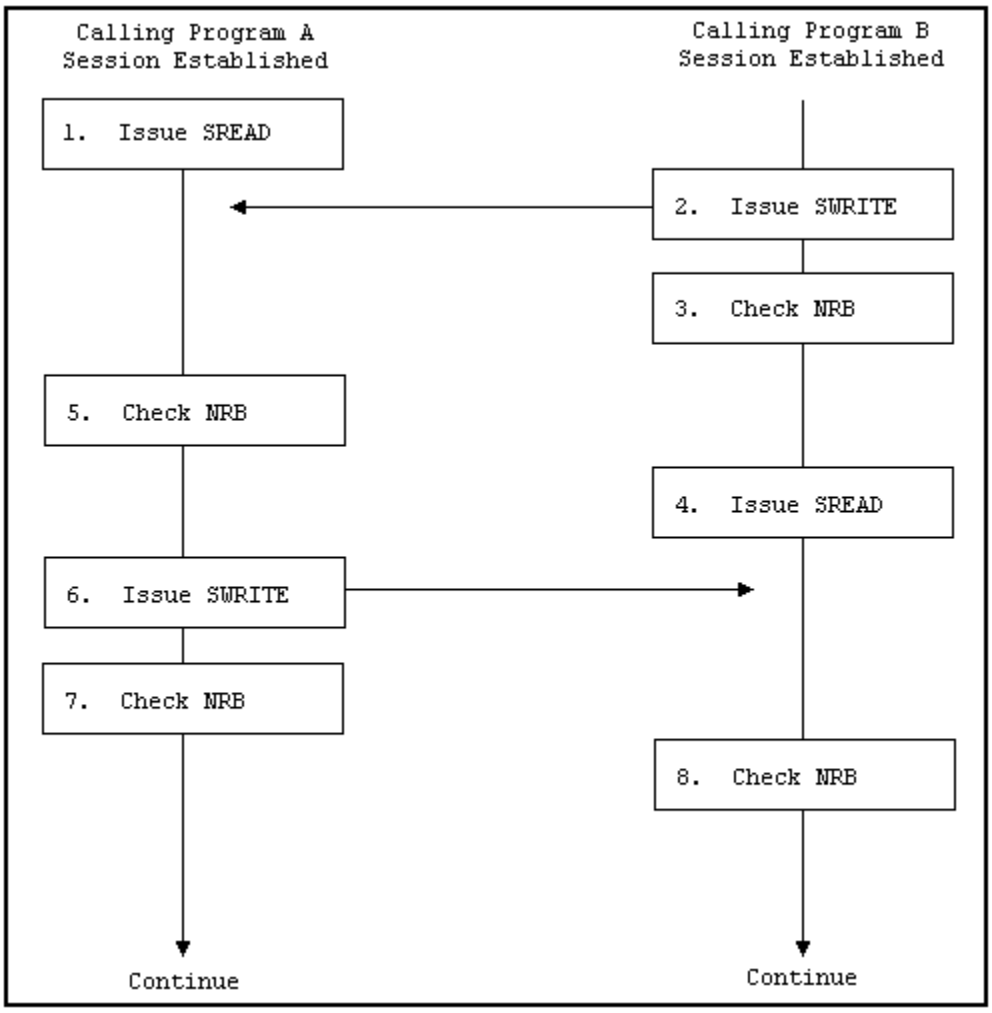


Figure 4. Write/Read Data Transfer

The following numbered items describe the steps in Figure 4. The steps are numbered to simplify the discussion and do not necessarily represent the exact order in which the events occur.

1. After a session has been established, program A issues an SREAD. The SREAD specifies the buffer for receiving data.
2. Program B issues an SWRITE. The SWRITE specifies where the data to be written is located and how long it is.
3. Program B checks the NRB to see if Secure NetEx/IP accepted the SWRITE or indicated an error.

   Once Secure NetEx/IP has accepted the data, it will deliver the data unless a catastrophic loss of connection occurs.
4. Program B issues an SREAD to detect program A's next request.
5. Program A received an updated NRB which indicates what program B has issued. In this case, program B has written data as program A expected. If the NRB word indicated an error, program A would act appropriately.

   If program B issued an SDISCONNECT, program A would check the reason for the SDISCONNECT and act appropriately.

6. Program A is programmed to SWRITE some data back to program B. Program A issues an SWRITE specifying the location of the data to be written.
7. Program A verifies that Secure NetEx/IP accepted the SWRITE by checking the NRB. If the NRB indicates the SWRITE was not accepted, program A would act appropriately.
8. Program B checks the NRB and determines what program A issued.

   Both programs continue with the data transfer until they have completed their functions.

As when establishing a session, the SWAIT request may be used with other requests. Abnormal terminations are discussed later in this chapter.

Concurrent Write and Read Data Transfer

A more advanced method of data transfer uses read and write requests that are issued without expecting the other calling program to respond immediately. This type of technique is useful for long distance communications, but is also well-suited for local data transfer.

Because Secure NetEx/IP will only accept one request using a specific NRB, two NRBs must be created by each program to perform concurrent read and writes. One NRB is used to establish the session, as previously described, and a second is created before data transfer begins. The second NRB must be created as a copy of the first to ensure that Secure NetEx/IP internal words are preserved.

Figure 5 shows how the programs perform the data transfer. As an example of what work the program is doing, consider the following. Program A first requests data from Program B. Program B then writes data until program A writes an acknowledgement or another message. Notice that program A does not respond to every SWRITE issued by program B. When program A has received what it needs, it terminates the session using the termination procedure discussed later in this chapter.

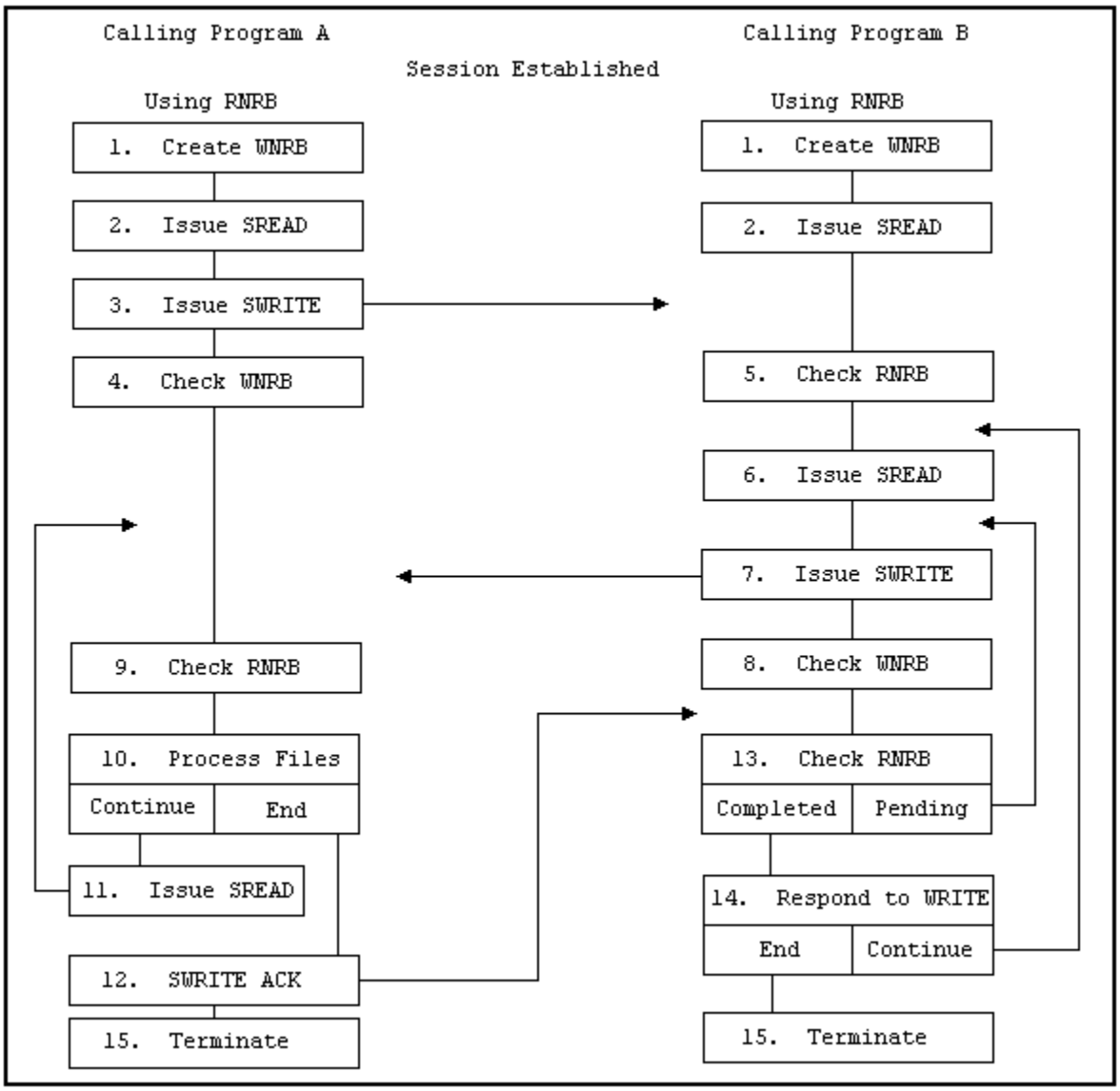


Figure 5. Concurrent SREAD and SWRITE Requests

The following numbered items describe the steps in Figure 5. The steps are numbered to simplify the discussion and do not necessarily represent the exact order which the event occur.

1. After a session has been established, both programs create duplicate NRBs for the SWRITE requests. The NRBs created before the sessions were established are assumed to have been called RNRB, and will be used with SREAD requests. New NRBs called WNRB are duplicates of the RNRB that will be used with the SWRITE requests.
2. Both programs issue an SREAD request to prepare to receive request from the other program. The RNRB is specified in the SREADs as the place for Secure NetEx/IP to respond to that request.
3. Program A issues an SWRITE to program B. The SWRITE contains a request for specific data from program B. The WNRB is specified in the SWRITE as the place for Secure NetEx/IP to respond to that request.

4. Program A checks the WNRB to see if Secure NetEx/IP accepted the SWRITE or indicated an error and acts accordingly.
5. Program B, which has been checking the RNRB or waiting for it to complete, received the SWRITE from program A. This SWRITE contains parameters which program B will use to determine what data to send to program A.
6. Program B issues another SREAD that will be "floating" while processing continues.
7. Program B begins to SWRITE the data requested by program A. The WNRB is specified to monitor the SWRITE requests.
8. Program B checks the WNRB to make sure Secure NetEx/IP accepted the SWRITE.
9. Program A checks its RNRB and discovers the SWRITE issued by program B.
10. Program A processes the files received. If program A has not yet received all the data it asked for in step 3 and wished to continue reading, it jumps to step 11. If program A wishes to respond to program B (to stop the transfer or to request other data), it jumps to step 12 and SWRITEs an appropriate message.
11. Program A issues an SREAD to continue receiving information from program B.
12. Program A SWRITEs an acknowledgement or a message to program B. Since program B has an SREAD floating, it will receive this SWRITE.
13. Program B checks the RNRB. If the SREAD has completed (meaning program A has written something), program B continues with step 14. If the SREAD is still pending (or floating), program B continues WRITING data to program A by jumping back to step 7.
14. Program B responds to program A's SWRITE. This response could include starting to transmit other data
15. The session is terminated normally when program A has received all the data it wanted, or by special request of one of the programs.

The previous example demonstrates the technique of using SREAD and SWRITE requests when using this technique.

One-Way Data Transfer

A typical use of Secure NetEx/IP is a one-way data transfer. Figure 6 shows how a one-way data transfer could take place. Programs A and B establish a session as described earlier in this section. Program A, which will receive data, creates a single NRB. Program B, which will send data, creates an RNRB (for monitoring SREAD requests) and a duplicate WNRB (for monitoring SWRITE requests).

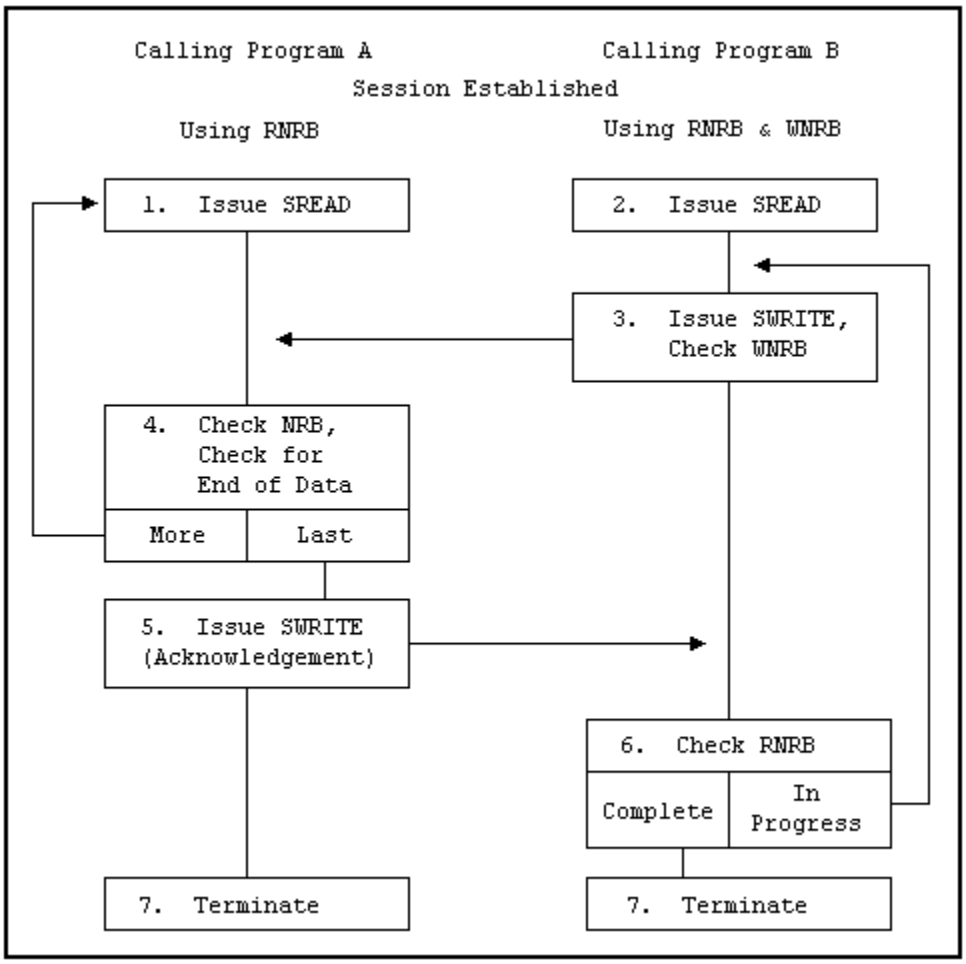


Figure 6. One-Way Data Transfer

The following numbered items describe the steps in Figure 6. The steps are numbered to simplify the discussion and do not necessarily represent the exact order in which the events occur.

1. Program A issues an SREAD request to prepare to receive data.
2. Program B issues an SREAD request to receive unexpected SWRITEs from program A. For example, if program A were unable to receive more data, it could notify program B using a previously determined set of indicators. Normally, this SREAD would not complete until all the information has been transferred.
3. Program B issues an SWRITE to transfer data to program A. An 'End of Data' indicator is placed in the data field with the last piece of information. Program B checks the WNRB to see if Secure NetEx/IP accepted the SWRITE or indicated an error and acts accordingly.
4. Program A checks the NRB to see if the SREAD completed. If it did not complete, program A continues to check it. When the SREAD completes, program A processes the incoming information, and checks for the 'End of Data' indicator. If there is more data coming (indicator not set), program A issues another SREAD (step 1). If this is the last piece of information, program A continues with step 5.
5. Program A issues an SWRITE acknowledging that all of the information has been received. (Secure NetEx/IP has verified the integrity of the data.)
6. Program B checks the RNRB. If it is still in progress (because program A has not written anything), program B jumps back to step 3 and SWRITEs more data. If all data has been written, program B will issue

an SWAIT to suspend execution until program A returns a response. When the RNRB completes, program B checks the data written by program A. Normally, this would be an acknowledgement of the last piece of data. In that case, program B would continue with step 7. If a problem is encountered, program B acts accordingly.

7. Program B terminates the session.

In the previous example, SWAIT requests may be used freely by program A, but should generally be used only after SWRITE requests by program B. An SWAIT could be issued by program B to wait on the RNRB after all data has been written.

Terminating a Session

Secure NetEx/IP sessions can terminate either normally or abnormally. A normal termination is planned by the programs involved. An abnormal termination is any unplanned termination of the session.

Normal Termination

Figure 7 shows the exchange of session calls associated with normal (planned) termination. The steps are numbered to simplify the discussion and do not necessarily represent the exact order in which the events occur.

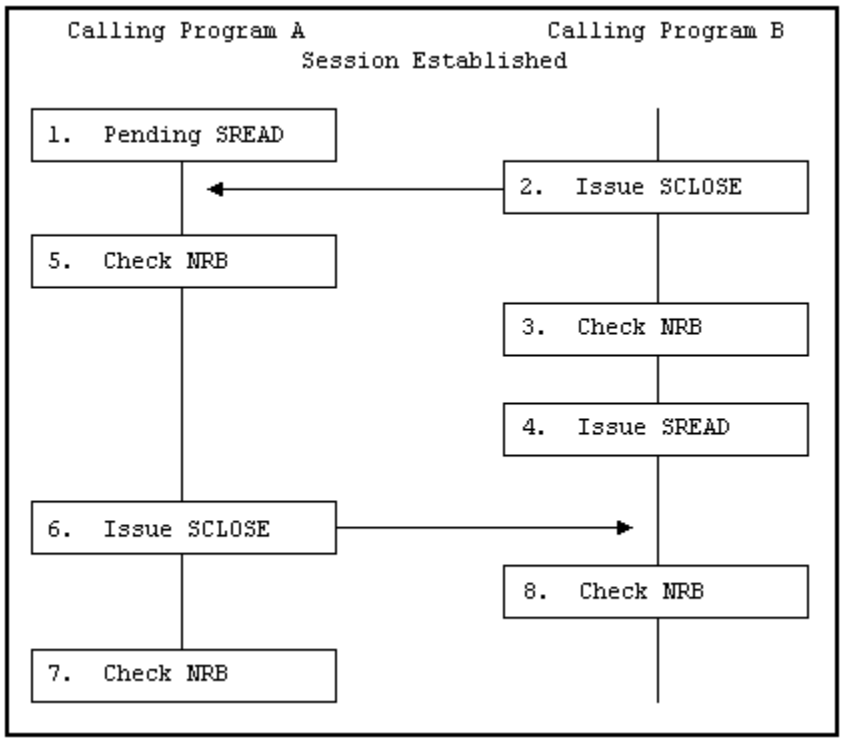


Figure 7. Normal Session Termination

The following numbered items refer to the steps in Figure 7.

1. Program A has a previously issued SREAD pending.
2. Program B issues an SCLOSE. The SCLOSE takes the same form as an SWRITE request.

3. Program B checks the NRB to verify that the SCLOSE was accepted by Secure NetEx/IP. If it was accepted, program B may close output files or perform other termination processing. No other Secure NetEx/IP write-type requests (except SDISCONNECT) may be issued by program B during this session. If the SCLOSE is not accepted, Program B must act appropriately (check if Secure NetEx/IP is down or if the other application is not there).
4. Program B issues an SREAD. Program A may still write data to program B, or program A may issue an SCLOSE or an SDISCONNECT to terminate the session.
5. Program A detects the SCLOSE.
6. Program A issues an SCLOSE to terminate the session. Data may be associated with this request. Program A may issue any number of SWRITE requests before the SCLOSE.
7. Program A checks the NRB to make sure that the SCLOSE completed successfully. Program A closes files and performs other termination processing.
8. The SREAD issued by program B completes and the NRB indicates that the session has been terminated. Program B closes files and performs other termination processing.

Abnormal Session Termination

The program must be able to react to abnormal terminations. Sessions may be abnormally terminated by the other program or by Secure NetEx/IP.

Sessions may be unexpectedly terminated by the other program for various reasons depending on how the program is written. Some typical reasons for immediate terminations are as follows:

A program fails to provide the proper password or authorization for a session

A program attempts to access data that it was not authorized to access.

The program detected an internal failure such as a program check.

A time limit was reached

A program encountered problems issuing a request to Secure NetEx/IP.

Some of these problems may be overcome by reconnecting with the calling program.

Secure NetEx/IP may terminate a session unexpectedly because of problems with the physical network or with a host computer. This type of error may not necessarily be solved by simply reconnecting with this host. Alternatives should be provided in the calling program.

Programming Notes

The following sections provide supplemental information intended to make programming Secure NetEx/IP simpler:

Handling Multiple Connections

Service Wait Options

Satellite Communication

Error Recovery Procedures

Code Conversion Options

Handling Multiple Connections

Secure NetEx/IP provides the capability for a program to be simultaneously connected to more than one other calling program. Requests coming from different connections are identified using the NRBNREF word of the NRB. NRBNREF contains a unique number assigned to a connection when it is established. The capability to handle multiple connections enables the programmer to establish database server and requester programs.

Database server programs allow a network of hosts to use each other's databases. A database server program simply OFFERs itself to other hosts. When another host (requester) establishes a connection, the server SREADs or SWRITEs files to or from its database as specified by the requester.

Database server programs may issue multiple offers (SOFFER) by specifying a different NRB with each SOFFER. The offers (SOFFER) are completed in the order that they are issued. Many users on one machine may issue multiple OFFERs if each is generated as if it were an individual host.

Program B in the concurrent write/read example (Figure 5) is an example of a simple database server. Program B SWRITEs the files program A requests, and then waits for more instructions. More sophisticated database servers could allow themselves to connect to several requesters at one time.

Program A in Figure 5 is a simple example of a database requester.

Service WAIT Options

On each session call the user has the option to wait or not to wait for the request to complete. If the waiting is desired then the "W' form of the call should be used. The User Request Manager will issue the wait on the users' behalf and return control to the user when the NRB is posted.

The SWAIT request may be used to wait on a single NRB, a list of NRBs or may be used to wait on 0 NRBs. The way to use the wait request depends on the situation.

If servicing a single connection where data moves logically in only one direction, use the wait option on the requests.

If servicing multiple simultaneous connections, or a single connection where data flows both ways, use SWAIT(n) to wait on a list of NRBs. If a listed NRB is not pending, the application may hang.

When servicing both Secure NetEx/IP and real-time applications, use SWAIT(0) to wait on 0 NRBs, then check the real-time device. When issuing an SWAIT on 0 NRBs, check the NRBSTAT fields of the NRBs for the requests that you are interested in.

Another way to wait for any NRB to complete, without specifying each NRB on the call, is to use SWAIT(-1). This form of wait will return to the user only when an NRB has completed. This differs from SWAIT(0), which may return without an NRB completing.

The following points apply to waiting:

A request cannot be marked complete until it is waited on.

The NRB and the data buffer cannot be reused until the request is complete.

Mitigating Propagation Delay

The standard Secure NetEx/IP requests may be used when large distances are a part of the network. Secure NetEx/IP was implemented with the capability to recover from errors and lost messages using protocols which make the solving of these problems transparent to the user.

IMPORTANT: Because of the long propagation delay inherent in the global networks (approximately 450ms round trip), the communications channel must be kept "full" of data. This is done by using concurrent

SREADs and SWRITEs which transmit data in large amounts before having the writing application stop to wait for an acknowledgement from the reading application. In this way, data is transmitted rapidly and the propagation delay has less effect on performance. This technique requires a large buffer area which require tuning TCP stacks for larger receive buffers, windows and window scaling.

For example, if the calling programs acknowledge every block written in one direction with a corresponding acknowledgement written in the other direction, the propagation delay would have major impact. But, if an entire file is transmitted before an acknowledgement is returned, the effect of the propagation delay is minimized.

Minimizing the effect of the delay in this manner must be balanced with the consideration that if there is a catastrophic failure of the network, Secure NetEx/IP, or the other host, there is no way to know how much unacknowledged data was successfully received.

Determining the frequency of the checkpoint acknowledgements is an important consideration. This decision must be made by considering the needs of individual implementations.

Secure NetEx/IP Error Recovery Procedures

Calling programs must be able to recover from errors identified by Secure NetEx/IP. These errors will be returned when a Secure NetEx/IP operation does not complete successfully. The following paragraphs describe the Secure NetEx/IP error codes and some common error recovery procedures.

Error Codes

Whenever a Secure NetEx/IP request is issued, the results of the request are returned in one or both of two NRB fields, NRBSTAT and NRBIND. These are located at the beginning of the NRB to make their subsequent examination by high level language programs a simple matter.

NRBSTAT indicates whether an operation is in progress and whether it completed successfully or not. NRBIND indicates the type of information that arrived as the result of a read-type command (SREAD or SOFFER).

When an operation is accepted by the Secure NetEx/IP user interface, the value of NRBSTAT is set to the local value of –1. Thus, the sign of this word is an “operation in progress” flag for all implementations.

If an operation completed successfully, NRBSTAT will be returned as all zeroes. If a read-type command was issued, then an “indication” will be set in NRBIND when the SREAD completes.

If the operation did not complete successfully, the NRBSTAT will contain a standard error code. NRBSTAT is represented as a decimal number that is potentially as large as 2^{15} –1(32,767) The 2^{16} bit is not used so that it may remain an “in progress” flag for the 16 bit machines. The error codes are listed and described in appendix A.

Common Error Recovery Procedures

The following items are some commonly encountered errors and an explanation of how to recover from them.

Other program not there – Operators or users must coordinate the running of the two Secure NetEx/IP programs so that one has not timed out before the other has had a chance to establish a session.

Other program busy – Retry Secure NetEx/IP after a suitable delay.

Secure NetEx/IP requests out of sequence – Sessions must be completely established before write or read requests can be issued. Sessions are established using the offer, connect, and confirm requests in that order.

Code Conversion

Secure NetEx/IP provides for common types of code conversion by using Secure NetEx/IP software facilities. The calling program uses the datamode (NRBDMODE) word of the NRB to specify code conversion. The caller simply specifies the source character set and the destination character set. Secure NetEx/IP then performs code conversion using software as necessary.

Chapter 4: NetEx Request Block

The NetEx Request Block (NRB) is a block of parameters used to pass information between calling programs and Secure NetEx/IP. The NRB is the means by which programs and Secure NetEx/IP communicate with one another. The NRB is created by a calling program and may be updated by the program to pass information to Secure NetEx/IP, or Secure NetEx/IP may update the NRB to pass information to a program.

Each time a program makes a request to Secure NetEx/IP, the program specifies an NRB to be associated with the request. Secure NetEx/IP passes status information about that request back to the program via the NRB. Therefore, only one Secure NetEx/IP request may use an NRB at one time. If concurrent read and write requests are used, or if a server program will be connected to more than one program at a time, several NRBs must be used.

Rules for NRB Usage

The following principles are designed to make high level language usage of Secure NetEx/IP somewhat transportable between machines.

Before initiating a connection, the user must clear the entire NRB structure. When the connection is initiated, the user places whichever non-default values are needed in the user part of the NRB, and invokes Secure NetEx/IP service. Once the connection is initiated, the user must not change the OS dependent part of the NRB between calls to Secure NetEx/IP.

If the calling program is using the connection in a full duplex manner, the user will need to make a copy of the NRB to produce a “read NRB” and a “write NRB.” This copying operation is the copying the entire length of the NRB to another area, at a time when the NRB being copied is not active. If a second request for the same connection is issued from the copied NRB, the user interface must detect the condition and handle the new part of the connection accordingly.

Many NRB values are specified in addressable units. An addressable unit is the amount of information contained in one memory location for that machine. For example, a Unisys Computer Systems 2200 is 36 bits, IBM systems are generally eight bits, etc.

For multi-threaded implementations, if a session is to be shared between threads, you must insure that each thread has its own copy(ies) of the NRB made while no Secure NetEx/IP calls for that NRB are active.

NRB Fields

Figure 8 shows the fields in the NRB. The NRB contains 40 fields. Refer to the netex.h header file for field sizes and order. All fields not defined in the table are reserved and should not be used.

Many of the NRB fields may be updated by either program or Secure NetEx/IP with every request. However, the fields NRBBLKI, NRBBLKO, NRBRSV, NRBOFFER, and NRBHOST are associated with the session negotiation process. Information in these fields is updated by Secure NetEx/IP as their values change. Some of these fields are initially specified by the user in the OFFER and CONNECT requests parameters. When the CONFIRM completes, the negotiated values are returned in the NRB associated with the read of the CONFIRM information. Subsequent attempts to change these fields will have no effect.

NRB fields may be referenced by the names in the netex.h header file. Figure 8 shows the names and has a short description of these fields.

nrbstat	Secure NetEx/IP request status returned to user
nrbind	Data type indication from OFFER/READ
nrblen	Length of data
nrbubit	Unused bit count
nrbreq	User request code
nrbnref	Secure NetEx/IP reference number identifying connection within the user application
nrbbufa	Starting address of user's buffer
nrbbufl	Length of user's buffer
nrbdmode	Datamode for Write request
nrbtime	Request timeout in seconds
nrbclass	Class of service (Not applicable)
nrbmaxrt	Maximum data rate permitted (Not applicable)
nrbblki	Maximum buffer size for input requests
nrbblko	Maximum buffer size for output requests
nrbprota	Address of Odata
nrbprotl	Length of Odata
nrboffer	(Session Level) Offer name
nrbhost	(Session Level) Remote hostname
nrbuser	User ID set by some Secure NetEx/IP APIs
nrbosd	Reserved (operating system dependent data)

Figure 8. NetEx Request Block (NRB)

The following paragraphs describe the fields in the NRB shown in Figure 8.

NRBSTAT

NRBSTAT contains a status summary of the request issued by the user. If the request is currently in progress, the entire field contains a -1 (all ones). If the request completed successfully, the NRBSTAT is 0. If the request was unsuccessful (Secure NetEx/IP or the service routine detected an error), NRBSTAT contains an error code. The meanings of the error codes are described in Appendix A and have definitions in the ntxerror.h file.

The implementation user interface must be constructed so that a program polling NRBSTAT (to determine if the request as successful) proceeds to examine the error code if a positive value is found in NRBSTAT.

A request is marked complete only after one of the following conditions is met:

A WAIT option was integrated into the service call.

An SWAIT request has been issued where one of the NRBs on the SWAIT list is the NRB specified.

Any Secure NetEx/IP service call is issued, and the Secure NetEx/IP service finds that the request has completed recently.

NRBIND

NRBIND indicates the type of data received in response to a read, offer, or status request. If any of those read-type requests are issued, NRBIND will always receive a nonzero value.

The values returned in NRBIND are defined as follows:

INDCONNECT (1) – Connect indication

INDCONFIRM (2) – Confirm indication

INDDATA (3) – Normal data indication

INDEXPDATA (4) – Expedited data

INDCLOSE (5) – Close indication

INDDISCONNECT (6) – Disconnect indication

INDSTATUS (7) – Status Indication

If a write-type request (write, connect, confirm, close, or disconnect) is issued, the returned value of NRBIND is usually zero. If an error is returned to the write-type request, that means the connection is broken or was never established.

If an operation did not complete successfully, then NRBSTAT will be set to a positive, nonzero value. If NRBSTAT is nonzero, then NRBIND will have one of the following values:

If the error results in the loss of the connection or the connection not being established in the first place, then a Disconnect Indication (6) will be in NRBIND.

If the error means that the request could not be processed but the connection remains in effect, then NRBIND will be set to zero.

NRBLEN and NRBUBIT

NRBLEN and NRBUBIT together define the amount of useful data for input and output. NRBLEN specifies the number of addressable units that are needed to contain the data. NRBUBIT specifies the number of bits in the last addressable unit that are not significant information. This allows information to be sent on the network on a logical bit basis without damaging the data.

For example, suppose a Unisys computer wished to send exactly 75 of its 36-bit words to an IBM processor and wished it returned at a later date. The Unisys user would specify NRBLEN=75 and NRBUBIT=0. Datamode would be bit stream. Secure NetEx/IP would record 75*36=2700 bits of information was sent over the network. The IBM user would receive the information with NRBLEN=338 (bytes) (8*338=2704 bits) and NRBUBIT=4 (bits). The IBM user could later specify the same length parameters on output and return precisely 75 words back to the Unisys system.

Note: Those programs that do not need the NRBUBIT can simply ignore its existence, knowing that simply handling the information specified by NRBLEN will ensure that all information sent by the other machine will be stored or processed.

Transmitting or receiving zero-length information is possible. Zero-length data is treated as a separate transmission and is received at the other end in chronological order (as is any other data). On both the transmit and receive sides, NRBLEN will be set to zero.

If NRBUBIT is none-zero, the unused bits are not set to zero or any other value by Secure NetEx/IP. The calling program must handle any “garbage” that may be placed in the last word of the transfer.

NRBREQ

NRBREQ is generally filled in by the API when the Secure NetEx/IP call is made and contains the type of request (example: SREAD) that Secure NetEx/IP is to perform.

NRBREQ has the following format:

Function indicates the specific type of request to be issued. The values are assigned (in hexadecimal) as follows and are 12 bits:

x001 - Connect

x002 - Confirm

x003 - Write

x004 - Reserved

x005 - Close

xx06 - Disconnect

xx07 to xx80 - Reserved

xx81 - Offer

xx82 - Read

xx83 - Status

xx84 to xxFF - Reserved

NRBNREF

NRBNREF is an internal Secure NetEx/IP identifier that distinguishes this connection from all others maintained for this user application. This value is assigned by Secure NetEx/IP when a connection is established.

NRBBUFA

NRBBUFA contains the start of the data buffer to be used for either input or output requests. The user must supply a valid buffer address before each input or output request. For a write request, the contents of this buffer must be left unchanged until the associated Secure NetEx/IP write type request has completed. If a read request is issued, then the contents of the buffer should be examined when the read request completes successfully.

NRBBUFL

On input, NRBBUFL specifies the maximum size of the Pdata (ordinary data) that Secure NetEx/IP can store in the buffer. This field is effectively ignored on output (NRBLEN and NRBUBIT are used to determine the actual length of output data). This usage difference allows a Secure NetEx/IP user to associate an NRB with a

single buffer and never change this field even if many READs and WRITEs are issued. NRBBUFL is specified in addressable units.

NRBDMODE

NRBDMODE is specified by the transmitting Secure NetEx/IP application on any write-type request (connect, confirm, write, close, or disconnect). It is always specified as a 16-bit quantity. Datamode is forwarded through all layers of Secure NetEx/IP. When the receiving entity received the data, the datamode specified by the transmitter (with possible modifications as described below) is inserted into the NRB associated with the receiving SREAD or SOFFER request.

Datamode is designed for all common Secure NetEx/IP transfers. When datamode is selected, the user identifies the source and destination character sets, and Secure NetEx/IP selects the appropriate assembly/disassembly and code conversion. Secure NetEx/IP will perform code conversion only when the selected conversion is meaningful to the receiving machine

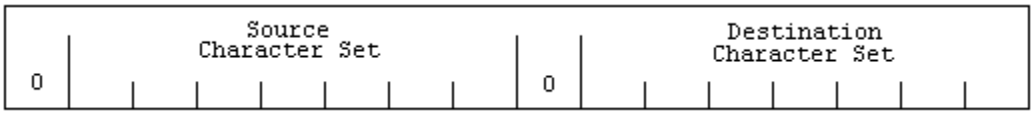
Datamode supports three conversion options.

Bit Stream where the bit pattern sent is precisely reproduced in the destination machine.

Octet where eight-bit binary quantities are sent from one machine to another, using an eight-bit byte representation appropriate to each machine.

Character where character information is sent from one machine to another with a full range of character assembly and code conversion options.

The conversion options are selected in the NRBDMODE field. The datamode has the following format in the NRBDMODE field:



'0' must be in the high order bit

Source Character Set indicates the conversion option of the data used in the write buffer of the transmitter.

'0' must be in the high bit of the low order byte

Destination Character Set indicates the conversion option of the data going to the destination program. For example, a conversion from EBCDIC (3) to ASCII (2) would be entered as a hexadecimal value of 0302.

Value	Conversion Option
0	Bit stream mode
1	Octet Mode
2	ASCII (8 bit)
3	EBCDIC
4	Reserved
5	BCD
6	Field-data (Unisys)

If the incoming Secure NetEx/IP determines that the destination character set is not “meaningful” on its own host, then it treats the incoming character set as “octet mode” data and provides the user with the data along with an error code in NRBSTAT indicating that the data was “damaged”; no conversion is done.

NRBTIME

NRBTIME specifies the length of elapsed time that the associated read-type command is to remain in effect. If a time interval equal to the number of seconds in NRBTIME has elapsed, and no data or connection information has arrived to satisfy the READ or OFFER, then the request will end with an error.

If the value in NRBTIME is “0”, then the request will wait indefinitely.

NRBCLASS – N/A

This field is no longer applicable. The value will be whatever was set by the user.

NRBMAXRT – N/A

This field is no longer applicable. The value will be whatever was set by the user.

NRBBLKI and NRBBLKO

NRBBLKI and NRBBLKO are connection negotiation parameters that specify the maximum amount of data that the calling program expects to read or write at one time during the coming connection. This parameter should be provided with the connect or offer request. During the session negotiation process, the NRBBLKI of one program will be compared with the NRBBLKO (output maximum buffer size) specified at the other end, and the lesser of the two values will be returned in the two respective fields.

For the connecting program, the negotiated results will be returned in the NRB along with the confirm data read following the connect. The offering program will receive the negotiated values on completion of the offer and hence may decide if the negotiated values are acceptable for the work at hand.

The Secure NetEx/IP installation systems programmer must supply two values controlling these buffer sizes in the snxmap.cfg file. First, the default input and output block sizes to be used if these are not specified (left zero) by the caller. Secondly, the maximum input and output block sizes permitted by the installation.

As an example of the block negotiation process: Program A issues a connect with NRBBLKI=256 and NRBBLKO=4096. The offering program B to which A will connect specifies 64K for both, allowing the connector to set any reasonable value in these fields. When the offer completes, B sees NRBBLKO=256 and NRBBLKI=4096, the minimum of the two sets of values. When A’s read following the connect completes, it will see NRBBLKI=256 and NRBBLKO=4096, which are the same values as B with the directions reversed.

Two default options are available with these fields. If a zero is specified (in connect or offer) in either one of these fields, then the value used for negotiation will be an installation supplied default that is provided at Secure NetEx/IP installation time (snxmap.cfg). If the value in this field is the machine representation of –1, then the size used for negotiation will be the maximum size available for that installation, which is also a parameter specified at initialization time (snxmap.cfg). Note that the values implied using zero or –1 will be used for negotiation of the connection block sizes. The actual size negotiated will be supplied in these fields on completion of the connect or offer.

NRBPROTA and NRBPROTL

The NRBPROTA and NRBPROTL are parameters that permit the application to provide Odata to the remote application. NRBPROTA specifies the address of the buffer containing the Odata, and NRBPROTL specifies the number of octets of Odata in that buffer.

When a write-type command is issued, the Odata provided (if any) will be added to the message, and eventually delivered as Odata to the receiving application's read type command. As a result, this is a second buffer that is handled in a similar way to the Pdata that is specified by NRBBUFA and NRBLEN/NRBUBIT. There are some distinct differences that are as follows:

Odata is always sent and received in "octet mode," which means it will be represented in the best way that the particular host can handle strings of eight-bit binary quantities, (for example, 1/byte, 4/36-bit word, etc.).

Users should be warned that sending excessive amounts of Odata with normal transmissions may result in a multiple network messages, which will increase network traffic and decrease network performance, often by a factor of two.

On a write-type operation, no Odata will be sent if NRBPROTL is zero. If a non-zero length is specified, then the Odata will be transmitted along with the Pdata, if present. When the read takes place, the Odata will be placed in the address specified by NRBPROTA and its incoming length will be set in NRBPROTL.

Always, NRBPROTL contains the length of the Odata in octets, not "addressable units".

NRBOFFER

NRBOFFER is a required parameter for connect and offer, which specifies the offered name used to match when the offer and connect requests meet). Names of all processes are compared as uppercase alphanumeric data that are up to eight characters in length. Names less than eight characters long will be padded with blanks. Process names will be converted to the ASCII character set for transmission between hosts, so only those characters that are meaningful in ASCII should be used during the name matching process.

NRBHOST

NRBHOST is a required parameter for connects which specifies the Secure NetEx/IP hostname or group name of the host computer that will be addressed to match an offer request. Names of all hosts need to be resolvable. All host names are up to eight characters in length. Names less than eight characters long will be padded with blanks. In order to provide a mechanism to isolate NetEx hosts in a network, Secure NetEx/IP will first try to resolve the NRBHOST with a prefix of NTX (i.e. NTXzostest1). If that does not resolve, it will then try to resolve NRBHOST (i.e. zostest1). This provides a way for a site to define NetEx host IP addresses differently than IP host addresses.

NRBUSER

This field may be used differently on each platform. The content of this field is maintained by Secure NetEx/IP during a session.

NRBOSD

NRBOSD is reserved for internal use. Secure NetEx/IP software uses this field to service and monitor the progress of NRB requests. The contents of these fields are maintained by Secure NetEx/IP during a session.

If the NRBOSD field is altered by the calling program, the results are unpredictable.

Creating an NRB

A single NRB should be created before a calling program OFFERs or CONNECTs to another program. The NRB should initially be zero-filled. Programs may create several NRBs initially.

If several NRBs are required to service a single connection, they should be duplicated from the initial NRB, as described in the following paragraphs.

Duplicating an NRB

Duplicating NRBs is necessary when using multiple NRBs within a session. By duplicating the NRB, the connection-negotiation parameters, the connection reference number and the internal NRBOSD information is preserved, allowing the duplicate NRB to be valid.

To duplicate an NRB, wait until the initial CONNECT or OFFER has completed successfully, then copy the entire "working" NRB (up to and including the NRBOSD field) to a blank NRB at a different location. The second NRB can now be used for Secure NetEx/IP requests. The NRB should only be duplicated when it is not in use by Secure NetEx/IP; that is, when NRBSTAT is 0.

Chapter 5: C High Level Interface

Secure NetEx/IP includes a library of subroutines that are designed to be called by the C high level object module. Also included is a library of copy files that may be included (#INCLUDE) into a C program and inserted at compile time. When the user makes a call to the user interface, the appropriate information is supplied in parameter format to pass to Secure NetEx/IP.

There are two components that are used to establish working communications through Secure NetEx/IP: one or more NETEX Request Blocks (NRBs) that must be supplied by the C caller, and the Secure NetEx/IP-provided subroutines that are used to invoke Secure NetEx/IP services. The NRB is described first, followed by the calls to the subroutines. The calls are presented in the approximate order in which they are used:

soffr	offer services (secure transmissions)
nsoffr	offer services (non-secure transmissions)
sconn	connect to an offered program in secure mode
nsconn	connect to an offered program in a non-secure mode
sconf	confirm acceptance of connect
sread	read incoming request or data
swrit	write data
sclos	write last data
swait	wait for previous request(s) to complete
sdisc	immediate disconnect

C NETEX Request Blocks

The C user builds an NRB by declaring it to be a structure of type nrb. Various fields of this structure will hold the information to be transferred to Secure NetEx/IP, and others will contain the information that is returned by Secure NetEx/IP. If more than one NRB will be required to service an application, one NRB type should be defined and several NRB variables should be declared to be of that type. Before these NRB structures are used for any Secure NetEx/IP request, Network Executive Software advises to zero all the members of the record. This will allow defaults for fields not explicitly used by the caller to take effect.

The Secure NetEx/IP C functions have the philosophy that arguments commonly passed to Secure NetEx/IP (such as a data buffer address) will be passed as parameters to the function. “Exotic” elements to be passed to Secure NetEx/IP, such as maximum input block size; will be supplied by storing the desired value in the proper member of the NRB structure. When the request completes, the user C program directly accesses the desired members of the NRB structure to determine if the operation completed properly.

SOFFR C Function

The SOFFR (secure offer) function provides a means for a program desiring to accept a secure/private connection from a caller on the network to post the availability of the service. The SOFFR is actually a specialized form of read request. The SOFFR reads any data associated with the SCONN.

Before issuing an SOFFR function, the user must provide an NRB (described in Chapter 4: NetEx Request Block) to be used by the user interface. Also, SNXMAP must be active in the system.

SOFFR Function Format

The SOFFR function has the following format:

Function (Select One)	Required Parameters
soffr soffrw	(nrb, buffer, length, timeout, pname)

SOFFR Parameters

The following parameters are used in the SOFFR function. The parameters must be specified in the order presented. All parameters are required.

soffr

soffrw

This is the verb for this function. SOFFRW causes the calling program to be blocked until the request completes (before processing will be resumed).

nrb

This required parameter is a pointer to the NRB data area to be passed to the Secure NetEx/IP API.

buffer

This required parameter is a pointer to the buffer data area to receive data sent by the corresponding SCONN request.

length

This required parameter is the length of the buffer (in addressable units) that will hold the data sent by the corresponding SCONN. When called, *length* should contain the maximum size of the buffer. On return, the NRBLEN (NRB.length) field will contain the number of bytes of information actually sent to the offering application. The data type of length should be INT.

timeout

This required parameter is the number of seconds that the OFFER request should remain outstanding. If no application connects during this interval, then the OFFER will end abnormally. If *timeout* is specified as zero, the OFFER will remain outstanding indefinitely.

pname

This required parameter is the alphabetic name of the process to be offered to the corresponding calling program. The name offered is arbitrary, but must be known to the connecting program. This name may be provided as a packed array(.1..8.) of character or as a string in the CALL statement, if padded with blanks to eight characters in length.

SOFFER Entry Parameters

If required, the following NRB fields must be set by the user prior to the SOFFR call.

NRBPROTA	Address for incoming Odata
NRBPROTL	Length of buffer to hold Odata
NRBBLKO	Maximum transmission size acceptable
NRBBLKI	Maximum reception size acceptable

SOFFR Results

The following NRB fields are updated when SOFFR completes.

NRBSTAT	Success/failure code
NRBIND	Contains Connect Indication
NRBLEN	Length of incoming Pdata
NRBUBIT	Unused bit count of Pdata
NRBREQ	Request Code
NRBNREF	S-ref assigned this connection
NRBBUFA	Address for incoming Pdata
NRBBUFL	Length of buffer to hold Pdata
NRBDMODE	Datamode of the Pdata
NRBTIME	Number of seconds offer can be outstanding
NRBBLKI	Maximum reception Pdata size
NRBBLKO	Maximum transmission Pdata Size**NRBPROTL** Length of Odata received
NRBOFFER	Application name to offer
NRBHOST	Name of host where SCONN originated

NSOFFR C Function

The NSOFFR (non-secure offer) function provides a means for a program desiring to accept a non-secure connection from a caller on the network to post the availability of the service. The NSOFFR is actually a specialized form of read request. The NSOFFR reads any data associated with the NSCONN.

Before issuing an NSOFFR function, the user must provide an NRB (described in Chapter 4: NetEx Request Block) to be used by the user interface. Also, SNXMAP must be active in the system.

NSOFFR Function Format

The SOFFR function has the following format:

Function (Select One)	Required Parameters
nsoffr nsoffrw	(nrb, buffer, length, timeout, pname)

NSOFFR Parameters

The following parameters are used in the NSOFFR function. The parameters must be specified in the order presented. All parameters are required.

nsoffr

nsoffrw

This is the verb for this function. NSOFFRW causes the calling program to be blocked until the request completes (before processing will be resumed).

nrb

This required parameter is a pointer to the NRB data area to be passed to the Secure NetEx/IP API.

buffer

This required parameter is a pointer to the buffer data area to receive data sent by the corresponding NSCONN request.

length

This required parameter is the length of the buffer (in addressable units) that will hold the data sent by the corresponding NSCONN. When called, *length* should contain the maximum size of the buffer. On return, the NRBLEN (NRB.length) field will contain the number of bytes of information actually sent to the offering application. The data type of length should be INT.

timeout

This required parameter is the number of seconds that the OFFER request should remain outstanding. If no application connects during this interval, then the OFFER will end abnormally. If *timeout* is specified as zero, the OFFER will remain outstanding indefinitely.

pname

This required parameter is the alphabetic name of the process to be offered to the corresponding calling program. The name offered is arbitrary, but must be known to the connecting program. This name may be provided as a packed array(.1..8.) of character or as a string in the CALL statement, if padded with blanks to eight characters in length.

NSOFFR Entry Parameters

If required, the following NRB fields must be set by the user prior to the NSOFFR call.

NRBPROTA Address for incoming Odata

NRBPROTL Length of buffer to hold Odata

NRBBLKO Maximum transmission size acceptable

NRBBLKI Maximum reception size acceptable

NSOFFR Results

The following NRB fields are updated when NSOFFR completes.

NRBSTAT Success/failure code

NRBIND Contains Connect Indication

NRBLEN Length of incoming Pdata

NRBUBIT Unused bit count of Pdata

NRBREQ Request Code

NRBNREF S-ref assigned this connection

NRBBUFA Address for incoming Pdata

NRBBUFL Length of buffer to hold Pdata

NRBDMODE Datamode of the Pdata

NRBTIME Number of seconds offer can be outstanding

NRBBLKI Maximum reception Pdata size

NRBBLKO Maximum transmission Pdata Size

NRBPROTL Length of Odata received

NRBOFFER Application name to offer

NRBHOST Name of host where NSCONN originated

SCONN C Function

The SCONN (connect) function provides a means for a program to request a secure/private session with a program that has issued an SOFFR. The SCONN is a specialized form of a write request. The SCONN initiates the secure session and may optionally write data to the offerer at the same time, provided this data does not exceed the maximum size specified on either the sending or receiving application.

Before issuing the SCONN, an NRB must be provided for use by the user interface.

SCONN Function Format

The SCONN function has the following format:

Function (Select One)	Required Parameters
sconn sconnw	(nrb, buffer, length, datamd, pname, hname)

SCONN Parameters

The following parameters are used in the SCONN function. The parameters must be specified in the order presented. All parameters are required.

sconn
sconnw

This is the verb for this function. SCONNW causes the calling program to be blocked until the request completes (i.e. before processing will be resumed).

nrb

This required parameter is a pointer to the NRB data area to be passed to Secure NetEx/IP API.

buffer

This required parameter is a pointer to the buffer data area that holds the user data to be sent to the corresponding application

length

This required parameter is the length of the data (in addressable units) to be sent to the corresponding application, to be presented with the completion of the corresponding application's OFFER request. If no data needs to be sent to the other application, the length may be set to zero. The data type of *length* should be INT.

datamd

This required parameter is the datamode to be used to send the connect data to the corresponding application. The data type of *datamd* should be INTEGER. Refer to NRBDMODE in Chapter 4: NetEx Request Block for a discussion of the datamode parameter.

pname

This required parameter is the alphabetic name of the process offered (SOFFR) by the corresponding calling program. The name offered is determined by the other calling program. This name may be provided as a packed array(.1..8.) of character or as a string in the call invocation, if padded with blanks to eight characters in length.

hname

This required parameter is the alphabetic name of the host computer to be accessed to determine if the correct SOFFR is available. The "names" of the host computers in the network are determined by the Secure NetEx/IP installation systems programmer. This may be provided as a packed array(.1..8.) of character or provided as a string in the call invocation, if padded with blanks to eight characters in length.

SCONN Entry Parameters

If required, the following NRB fields must be set by the user prior to the SCONN call.

NRBPROTA	Address of outgoing Odata
NRBPROTL	Length of outgoing Odata
NRBBLKO	Maximum transmission size acceptable
NRBBLKI	Maximum reception size acceptable

SCONN Results

The following NRB fields are updated when SCONN completes.

NRBSTAT	Success/failure code
NRBIND	Set to zero
NRBLEN	Length of outgoing Pdata
NRBREQ	Request Code
NRBNREF	S-ref (Session ID) assigned
NRBBUFA	Address of outgoing Pdata
NRBUBIT	Pdata unused bit count
NRBDMODE	datamode of Pdata
NRBBLKO	Maximum transmission Pdata size
NRBBLKI	Maximum reception Pdata size
NRBOFFER	Alphanumeric "application" name
NRBHOST	8 character "host" name

NSCONN C Function

The NSCONN (non-secure connect) function provides a means for a program to request a non-secure session with a program that has issued an NSOFFR. The NSCONN is a specialized form of a write request. The NSCONN initiates the non-secure session and may optionally write data to the offerer at the same time, provided this data does not exceed the maximum size specified on either the sending or receiving application.

Before issuing the NSCONN, an NRB must be provided for use by the user interface.

NSCONN Function Format

The NSCONN function has the following format:

Function (Select One)	Required Parameters
nsconn nsconnw	(nrb, buffer, length, datamd, pname, hname)

NSCONN Parameters

The following parameters are used in the NSCONN function. The parameters must be specified in the order presented. All parameters are required.

nsconn
nsconnw

This is the verb for this function. NSCONNW causes the calling program to be blocked until the request completes (i.e. before processing will be resumed).

nrb

This required parameter is a pointer to the NRB data area to be passed to Secure NetEx/IP API.

buffer

This required parameter is a pointer to the buffer data area that holds the user data to be sent to the corresponding application

length

This required parameter is the length of the data (in addressable units) to be sent to the corresponding application, to be presented with the completion of the corresponding application's OFFER request. If no data needs to be sent to the other application, the length may be set to zero. The data type of *length* should be INT.

datamd

This required parameter is the datamode to be used to send the connect data to the corresponding application. The data type of *datamd* should be INTEGER. Refer to NRBDMODE in Chapter 4: NetEx Request Block for a discussion of the datamode parameter.

pname

This required parameter is the alphabetic name of the process offered (NSOFFR) by the corresponding calling program. The name offered is determined by the other calling program. This name may be provided as a packed array(.1..8.) of character or as a string in the call invocation, if padded with blanks to eight characters in length.

hname

This required parameter is the 8 character name of the host computer to be accessed to determine if the correct SOFFR is available. The "names" of the host computers in the network are determined by the Secure NetEx/IP installation systems programmer. This may be provided as a packed array(.1..8.) of character or provided as a string in the call invocation, if padded with blanks to eight characters in length.

NSCONN Entry Parameters

If required, the following NRB fields must be set by the user prior to the NSCONN call.

NRBPROTA Address of outgoing Odata

NRBPROTL	Length of outgoing Odata
NRBBLKO	Maximum transmission size acceptable
NRBBLKI	Maximum reception size acceptable

SCONN Results

The following NRB fields are updated when NSCONN completes.

NRBSTAT	Success/failure code
NRBIND	Set to zero
NRBLEN	Length of outgoing Pdata
NRBREQ	Request Code
NRBNREF	S-ref (Session ID) assigned
NRBBUFA	Address of outgoing Pdata
NRBUBIT	Pdata unused bit count
NRBDMODE	datamode of Pdata
NRBBLKO	Maximum transmission Pdata size
NRBBLKI	Maximum reception Pdata size
NRBOFFER	Alphanumeric “application” name
NRBHOST	8 character “host” name

SCONF C Function

The SCONF (confirm) function provides a means for an offering program to confirm (to the connector) that the connection has been successfully completed. A negative response to an SCONN would be an SDISC.

The SCONF is a specialized form of write request. Data may optionally be written during the confirm process with this command, provided this data does not exceed the maximum size specified on either the sending or receiving Secure NetEx/IP.

Before issuing the SCONF call, an SOFFR must have completed successfully by receiving an SCONN response. The calling program must provide a NRB with an NRBNREF relating to this session.

SCONF Function Format

The SCONF function has the following format:

Function (Select One)	Required Parameters
sconf sconfw	(nrb, buffer, length, datamd)

SCONF Parameters

The following parameters are used in the SCONF function. The parameters must be specified in the order presented. All parameters are required.

sconf
sconfw

This is the verb for this function. SCONFW causes the calling program to be blocked until the request completes (i.e. before processing will be resumed).

nrb

This required parameter is a pointer to the NRB data area to be passed to Secure NetEx/IP API.

buffer

This required parameter is a pointer to the buffer data area that holds the user data to be sent to the corresponding application.

length

This required parameter is the length of the data (in addressable units) to be sent to the corresponding application, to be presented with the completion of the corresponding application's SREAD request. If no data needs to be sent to the other application, the length may be set to zero. The data type of *length* should be INT.

datamd

This required parameter is the datamode to be used to send the connect data to the corresponding application. The data type of *datamd* should be INTEGER. Refer to NRBDMODE in Chapter 4: NetEx Request Block for a discussion of the datamode parameter.

SCONF Entry Parameters

If required, the following NRB fields must be set by the user prior to the SCONF call.

NRBUBIT	Pdata unused bit count
NRBPROTA	Address of outgoing Odata
NRBPROTL	Length of outgoing Odata

SCONF Results

The following NRB fields are updated when SCONF completes

NRBSTAT	Success/failure code
NRBIND	Set to zero
NRBREQ	Request Code
NRBBUFA	Address of outgoing Pdata (move mode)
NRBLEN	Length of outgoing Pdata
NRBDMODE	datamode of Pdata

SREAD C Function

The SREAD function provides a means for a program to receive data from another host or an indication from Secure NetEx/IP API of an abnormal condition with the connection.

Before an SREAD can be issued, a connection must be established. The NRB specified must have been used for a previous Secure NetEx/IP request for the particular connection, or a copy of another NRB that has been used to service the desired connection.

IMPORTANT: Keep an SREAD request outstanding to receive incoming data. Secure NetEx/IP will automatically terminate a connection if a request is waiting to be read for too long. This read-timeout value is set at installation time.

Defaults for unspecified parameters are assumed to be the parameters existing in the NRB. Thus, after *buffer* and *length* are agreed on during the connection process, these parameters can be omitted.

SREAD Function Format

The SREAD function has the following format:

Function (Select One)	**Required Parameters**
sread sreadw	(nrb, buffer, length, timeout)

SREAD Parameters

The following parameters are used in the SREAD function. The parameters must be specified in the order presented. All parameters are required.

sread
sreadw

This is the verb for this function. SREADW causes the calling program to be blocked until the request completes (i.e. before processing will be resumed).

nrb

This required parameter is a pointer to the NRB data area to be passed to Secure NetEx/IP API.

buffer

This required parameter is a pointer to the buffer data area to receive the data sent by the corresponding application's SWRITE or SCONF request.

length

This required parameter is the length of the buffer (in addressable units) to hold the data sent by the corresponding SWRITE. When called, *length* should contain the maximum size of the buffer. On return, the actual length will be in NRBLEN. Programs that wish to work with the Unused Bit Count on input should examine the NRBUBIT field. The data type of *length* should be INT.

timeout

This required parameter is the number of seconds that the READ request should remain outstanding. If the corresponding application does not send data during this interval, then the read will end abnormally. If *timeout* is specified as zero, the READ will remain outstanding indefinitely.

SREAD Entry Parameters

If required, the following NRB fields must be set by the user prior to the SREAD call.

NRBPROTA Address for incoming Odata

NRBPROTL Length of buffer to hold Odata

SREAD Results

The following NRB fields are updated when SREAD completes.

NRBSTAT Success/failure code

NRBIND Contains Connect Indication

NRBLEN Length of incoming Pdata

NRBUBIT Unused bit count of Pdata

NRBREQ Request code

NRBBUFA Address for incoming Pdata (move mode)

NRBBUFL Length of buffer to hold Pdata

NRBTIME Number of seconds offer is to be outstanding

NRBBLKO Maximum transmission Pdata size (On Read of Confirm only)

NRBBLKI Maximum reception Pdata size (On Read of Confirm only)

NRBPROTL Length of Odata received

SWRIT C Function

The SWRIT (write) function provides a means for a program to transmit data to another calling program.

Before an SWRIT can be issued, a connection must be established.

SWRIT Function Format

The SWRIT function has the following format:

Function (Select One)	Required Parameters
swrit swritw	(nrb, buffer, length, datamd)

SWRIT Parameters

The following parameters are used in the SWRIT function. The parameters must be specified in the order presented. All parameters are required.

swrite
swritw

This is the verb for this function. SWRITW causes the calling program to be blocked until the request completes (i.e. before processing will be resumed).

nrb

This required parameter is a pointer to the NRB data area to be passed to Secure NetEx/IP API.

buffer

This required parameter is a pointer to the buffer data area that holds the user data to be sent to the corresponding application.

length

This required parameters is the length of the data (in addressable units) to be sent to the corresponding application. The length may be set to zero. The data type of *length* should be INT.

datamd

This required parameter is the datamode to be used to send the connect data to the corresponding application. The data type of *datamd* should be INT. Refer to NRBDMODE in Chapter 4: NetEx Request Block for a discussion of the datamode parameter.

SWRIT Entry Parameters

If required, the following NRB fields must be set by the user prior to the SWRIT call.

NRBPROTA Address of outgoing Odata

NRBPROTL Length of outgoing Odata

SWRIT Results

The following NRB fields are updated when SWRIT completes.

NRBSTAT	Success/failure code
NRBIND	Set to zero
NRBLEN	Length of outgoing Pdata
NRBUBIT	Pdata unused bit count
NRBREQ	Request Code
NRBDMODE	Datamode of Pdata
NRBBUFA	Address of outgoing Pdata

SCLOS C Function

The SCLOS (close) function provides a way for a program to transmit data to another corresponding calling program and indicate that this is the last data to be sent. The data must be received by a read request in the other program.

After a program has issued an SCLOS, no other data may be written by that program. If the other program had previously issued an SCLOS, the data is written and then the connection is disconnected. If the other program has not issued an SCLOS, it is still free to write data to the program that did issue the SCLOS.

Before issuing the SCLOS, a connection must be fully established.

SCLOS Function Format

The SCLOS function has the following format:

Function (Select One)	Required Parameters
sclos sclosw	(nrb, buffer, length, datamd)

SCLOS Parameters

The following parameters are used in the SCLOS function. The parameters must be specified in the order presented. All parameters are required.

sclos
sclosw

This is the verb for this function. SCLOSW causes the calling program to be blocked until the request completes (i.e. before processing will be resumed).

nrb

This required parameter is a pointer to the NRB data area to be passed to Secure NetEx/IP API.

buffer

This required parameter is a pointer to the buffer data area that holds the user data to be sent to the corresponding application.

length

This required parameter is the length of the data (in addressable units) to be sent to the corresponding application. The length may be set to zero. The data type of *length* should be INT.

datamd

This required parameter is the datamode to be used to send the connect data to the corresponding application. The data type of *datamd* should be INT. Refer to NRBDMODE in Chapter 4: NetEx Request Block for a discussion of the datamode parameter.

SCLOS Entry Parameters

If required, the following NRB fields must be set by the user prior to the SCLOS call.

NRBPROTA Address of outgoing Odata

NRBPROTL Length of outgoing Odata

SCLOS Results

The following NRB fields are updated when SCLOS completes

NRBSTAT Success/failure code

NRBIND Set to zero

NRBLEN Length of outgoing Pdata

NRBUBIT Pdata unused bit count

NRBREQ Request Code

NRBBUFA Contains zero (if ptr mode selected)

NRBDMODE Datamode of Pdata

SWAIT C Function

The SWAIT function provides the means to wait for the completion of requests that have not been previously waited for. Control will be returned to the SWAIT caller when any one of the NRBs specified no longer has the "in progress" flag set. Return from the subroutine will be immediate if any one of the NRBs has completed before the SWAIT call. By waiting on 0 NRBs, the Secure NetEx/IP API will take control and update all NRBs.

SWAIT(0) will return after checking all of the NRBs that are pending. It may return before any NRB has completed. Therefore, Network Executive Software advises to check the NRBSTAT after an SWAIT(0) and send another SWAIT(0) if the status is –1. This behavior is true whether SWAIT(0) is used in a single or multi-threaded application.

When SWAIT(-1) is called, Secure NetEx/IP API takes control and checks all NRBs, just as SWAIT(0) does. SWAIT(-1) will only return to the user when at least one NRB has completed.

Note: In a multi-threaded application, the NRB which completed may not belong to the thread making the swait(-1) call. If your thread has no NRBs active, you may end up waiting for another thread's NRB to complete! Think globally for this form of the call and use with caution!

In a multi-threaded environment, swait(nrbnum, nrb, nrb …) works as you would expect. This form of the call is recommended over the swait(-1) form.

After control is returned, the calling program must determine which of the NRBs in the list has completed. This can be done by examining the NRBSTAT field of each of the NRBs.

See "ntxteat.c" and "ntxtgen.c" in the "tests" subdirectory of your Secure NetEx/IP installed distribution for a sample server/client multi-threaded application. It also contains samples of all 3 forms of the 'swait' call. The sample of the swait(-1) call is not foolproof!

C SWAIT Examples

Figure 9 shows an example of an SWAIT (0).

```
(program)
.
.
swrit (nrba,buffa,len,dmode);
swrit (nrbb,buffb,len,dmode);
i=0;
while (i++<5&&
            nrba.nrbstat<0&&
            nrbb.nrbstat<0) {

            delay(1);
            swait(0);
}
if (nrba.nrbstat>=0) {
  /*process nrba */
}
else {
  /*process nrbb */
}
```

Figure 9. SWAIT(0) Example

Figure 10 shows an example of an SWAIT(-1).

```
(program)
.
.
swrit (nrba,buffa,len,dmode);
swrit (nrbb,buffb,len,dmode);
swait(-1);
if (nrba.nrbstat>=0) {
   /*process nrba */
}
else {
   /*process nrbb */
}
```

Figure 10. SWAIT(-1) Example

SWAIT Function Format

The SWAIT function has the following format:

Function	Required Parameters
swait	(nrbnum, nrb, nrb, ...)

SWAIT Parameters

The following parameters are used in the SWAIT function. The parameters must be specified in the order presented. All parameters are required.

swait

This is the verb for this function.

nrbnum

This required parameter is the number of NRBs to wait for. Control will be returned after the completion of any calls/NRBs specified. If 0 is specified, the NetEx subroutine library will take control and update NRBs or perform other functions.

nrb

This required parameter is a pointer to one or more NRBs (the number of NRBs specified in nrbnum) associated with the request to be waited for. An *nrb* is required for each NRB specified in *nrbnum*.

SDISC C Function

The SDISC (disconnect) function provides the means for any connected program to terminate a session. The request is immediate and any data currently in transport may not be delivered. If data delivery is required, the program must wait for confirmation of previous SREAD or SWRIT functions before issuing the SDISC.

When an SDISC is issued, an outstanding SREAD in the other program will terminate with an error in NRBSTAT.

NETEX does not ensure that data written with an SDISC macro will actually be received by the other program.

SDISC Function Format

The SDISC function has the following format

Function (Select One)	**Required Parameters**
sdisc sdiscw	(nrb, buffer, length, datamd)

SDISC Parameters

The following parameters are used in the SDISC function. The parameters must be specified in the order presented. All parameters are required.

sdisc
sdiscw

This is the verb for this function. SDISCW causes the calling program to be blocked until the request completes (i.e. before processing will be resumed).

nrb

This required parameter is a pointer to the NRB data area to be passed to NETEX.

buffer

This required parameter is a pointer to the buffer data area that holds the user data to be sent to the corresponding application.

Note: For SDISC, delivery of DISCONNECT data is **not** reliable, although the actual disconnection will always occur.

length

This required parameter is the length of the data (in addressable units) to be sent to the corresponding application, to be presented with the completion of the corresponding application's SREAD request. If no data needs to be sent to the other application, the length may be set to zero. The data type of *length* should be INT.

datamd

This required parameter is the datamode to be used to send the disconnect data to the corresponding application. The data type of *datamd* should be INT. Refer to NRBDMODE in Chapter 4: NetEx Request Block for a discussion of the datamode parameter.

On completion of the SDISC, the connection will no longer exist; new commands against that connection will be rejected. An SOFFR or SCONN must be issued to establish a new connection.

SDISC Entry Parameters

If required, the following NRB fields must be set by the user prior to the SDISC call.

NRBUBIT Pdata unused bit count

NRBPROTA Address of outgoing Odata

NRBPROTL Length of outgoing Odata

SDISC Results

The following NRB fields are updated when SDISC completes.

NRBSTAT Success/failure code

NRBIND Set to zero

NRBLEN Length of outgoing Pdata

NRBREQ Request Code

NRBBUFA Address of outgoing Pdata

NRBDMODE Datamode of Pdata

C Program Examples

The following figures are examples of C language offer and connect programs designed for the transfer of computer generated data.

To compile these programs use:

```
cc –o nsef001 nsef001.c –lntx
cc –o nsef002 nsef002.c –lntx
```

To run the programs:

```
nsef001  [datamode]
nsef002  hostname [datamode]
```

datamode

The default is 0, if the datamode is not specified.

To run intra-host:

```
nsef001 &
nsef002  localhostname
```

Figure 11 shows the offering side (nsef001) and Figure 12 shows the connecting side (nsef002) of a NetEx example.

```
/*
 * This is the offering side of the test, the basic sequence is
 *              offer with data verification
 *              confirm
 *              read and write to remote with data verification
 *              read disconnect
 * the session is terminated if there are any errors.
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <netex/netex.h>
#include <netex/error.h>

#define ELEMENTS 500

nrb n;
short buf[ELEMENTS],verf[ELEMENTS];
int step,i,mode, bytes = sizeof(buf[0]);
char *job[] = { "offer","connect","confirm","read","write","disc","wait"};

/*
 * terminate the session on error condition
 * verify > 0 indicates a data verification error.
 */
void
terminate(int verify)
{
```

```
        if(verify--)
                printf(" data miscompared at n=%d, buf=%d, verf=%d\n",
                        verify,buf[verify],verf[verify]);
        printf(" **%s** nrbstat=%ld, nrbind=%ld, nrblen=%ld, mode=%ld\n",
                  job[step],n.nrbstat,n.nrbind,n.nrblen,n.nrbdmode);
        sdiscw(&n,(char *)buf,1,mode);
        if(n.nrbstat != SUCCESS || n.nrbind != INDDISCONNECT)
                printf("**disc stat = %ld**, nrbind = %ld, nrblen = %ld\n",
                        n.nrbstat,n.nrbind,n.nrblen);
        exit(-1);
}

int
main(int argc,char *argv[]) /* nsef001 [datamode] */
{
        memset(&n,'\0',sizeof(n));
        memset(buf,'\0',sizeof(buf));
        step = 0;
        printf(" this is the start of nsef001\n");
        if(argc < 2)
                mode = 0;
        else
                sscanf(argv[1],"%x",&mode);
        verf[0] = 1234;

        /* offer, check for success, verify data */
        soffrw(&n,(char *)buf,4,300,"NSEF0011");
        if(n.nrbstat != SUCCESS || n.nrbind != INDCONNECT)
                terminate(0);
        if(buf[0] != verf[0])
                terminate(1);
        buf[0] = 1234;
        buf[1] = 5678;

        /* confirm the connection */
        step = 2;
        sconfw(&n,(char *)buf,8,mode);
        if(n.nrbstat != SUCCESS)
                terminate(0);

        /* read and verify data */
        step = 3;
        sreadw(&n,(char *)buf,400*bytes,120);
        for(i=0; i<400; i++) {
                verf[i] = i+1;
                if(buf[i] != verf[i])
                        terminate(i+1);
        }
        if(n.nrbstat != SUCCESS || n.nrbind != INDDATA || n.nrblen != 400*bytes)
                terminate(0);
        printf(" first read in nsef001 has completed and verified data\n");
        for(i=0; i<420; i++)
                buf[i] = i+1;

        /* write to remote */
        step = 4;
```

```
    swritw(&n,(char *)buf,420*bytes,mode);
    if(n.nrbstat != SUCCESS)
        terminate(0);

    /* read and verify data */
    step = 3;
    sreadw(&n,(char *)buf,40*bytes,120);
    for(i=0; i<40; i++) {
        if(buf[i] != verf[i])
            terminate(i+1);
    }
    if(n.nrbstat != SUCCESS || n.nrbind != INDDATA  || n.nrblen != 40*bytes)
        terminate(0);
    printf(" second read in nsef001 has completed and verified data\n");
    for(i=0; i<420; i++)
        buf[i] = i+1;

    /* write to remote */
    step = 4;
    swritw(&n,(char *)buf,420*bytes,mode);
    if(n.nrbstat != SUCCESS)
        terminate(0);

    /* read the disconnect */
    step = 3;
    sreadw(&n,(char *)buf,1,30);
    /* verify disconnect */
    if(n.nrbstat != SUCCESS ||n.nrbind != INDDISCONNECT)
        terminate(0);
    printf(" reading disconnect in nsef001 completed\n");
    printf(" this test completed successfully nsef001\n");
    exit(0);
}
```

Figure 11. Example: Offering Side of a NetEx Session (nsef001.c)

```
/*
 * this is the connecting side of the test the basic sequence is
 *            connect to remote
 *            read confirm with data verification
 *            write to and read from remote host with data verification
 *            disconnect
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <netex/netex.h>
#include <netex/error.h>

#define ELEMENTS 500

nrb n;
short buf[ELEMENTS],verf[ELEMENTS];
int step,i,mode,bytes = sizeof(buf[0]);
char *job[] = { "offer","connect","confirm","read","write","disc","wait"};
char host[sizeof(n.nrbhost) + 1];

/*
 * terminate the session because of an error
 * verify > 0 indicates data verification error
 */
void
terminate(int verify)
{
      if(verify--)
            printf(" data miscompared at n=%d, buf=%d, verf=%d\n",
                  verify,buf[verify],verf[verify]);
      printf(" **%s** nrbstat=%ld, nrbind=%ld, nrblen=%ld, mode=%ld\n",
           job[step],n.nrbstat,n.nrbind,n.nrblen,n.nrbdmode);
      sdiscw(&n,(char *)buf,1,mode);
      if(n.nrbstat != SUCCESS || n.nrbind != INDDISCONNECT)
            printf("**disc stat = %ld**, nrbind = %ld, nrblen = %ld\n",
                  n.nrbstat,n.nrbind,n.nrblen);
      exit(-1);
}

int
main(int argc,char *argv[]) /* nsef002 host [datamode] */
{
      memset(&n,'\0',sizeof(n));      /* zero out nrb */
      memset(buf,'\0',sizeof(buf));
      printf(" this is the start of nsef002\n");
      if(argc == 2)
            mode = 0;
      else if(argc == 3)
            sscanf(argv[2],"%x",&mode);
      else {
            printf(" nsef002: wrong arg count\n");
            exit(1);
      }
```

```
strncpy(host,argv[1], sizeof(host));
verf[0] = 1234;
verf[1] = 5678;
buf[0] = 1234;

/* connect to remote */
step = 1;
sconnw(&n,(char *)buf,4,mode,"NSEF0011",host);
if(n.nrbstat != SUCCESS)
      terminate(0);
step = 3;

/* read confirm, verify data */
sreadw(&n,(char *)buf,8,120);
for(i=0; i<2; i++)
      if(buf[i] != verf[i])
            terminate(i+1);
if(n.nrbstat != SUCCESS || n.nrbind != INDCONFIRM)
      terminate(0);
printf(" reading the sconfw completed successfully");
for(i=0; i<400; i++)
      buf[i] = i+1;

/* write to remote */
step = 4;
printf(" this is the mode before write %d, nrbmode = %ld\n",
     mode, n.nrbdmode);
swritw(&n,(char *)buf,400*bytes,mode);
printf(" this is the mode after swrite %d, nrbmode = %ld\n",
     mode, n.nrbdmode);
if(n.nrbstat != SUCCESS)
      terminate(0);

/* read from remote, verify data */
step = 3;
sreadw(&n,(char *)buf,420*bytes,30);
for(i=0; i<420; i++)
      if(buf[i] != (verf[i] = i+1))
            terminate(i+1);
if(n.nrbstat != SUCCESS || n.nrbind != INDDATA || n.nrblen != 420*bytes)
      terminate(0);
printf(" second read in nsef002 completed with data verified\n");

/* write to remote */
step = 4;
for(i=0; i<40; i++)
      buf[i] = i+1;
swritw(&n,(char *)buf,40*bytes,mode);
if(n.nrbstat != SUCCESS)
      terminate(0);

/* read from remote, verify data */
step = 3;
sreadw(&n,(char *)buf,420*bytes,120);
for(i=0; i<420; i++)
      if(buf[i] != verf[i])
```

```
                terminate(i+1);
        if(n.nrbstat != SUCCESS || n.nrbind != INDDATA || n.nrblen != 420*bytes)
                terminate(0);
        printf(" third read in nsef002 completed with data verified\n");

        /* disconnect */
        step = 5;
        sdiscw(&n,(char *)buf,1,0);
        if(n.nrbstat != SUCCESS || n.nrbind != 0) {
                printf(" ***disconnect*** nrbstat=%ld, nrbind=%ld\n",
                        n.nrbstat,n.nrbind);
                exit(-1);
        }
        printf(" this test completed successfully nsef002\n");
        exit(0);
}
```

Figure 12. Example: Connecting Side of a NetEx Session (nsef002.c)

Chapter 6: Operator Interface

There is a locally accessible operator interface for the SNXMAP component (snxmapop) which provides a small set of commands that allow you to display outstanding Secure NetEx offers and display current configuration parameters.

Command Descriptions Conventions

The following notational conventions are used in the Operator Command Help functions.

Format	Description
[ ]	One of the selections within the brackets are optional
…	One or more items may be specified

Command Line Mode

When you execute operator commands in command line mode, you initiate the SNXMAPOP program for each command and provide the command parameters as arguments to the program. To execute an operator command in command line mode, use the following general format:

```
snxmapop [option]… [command]
```

For the OS2200 system, modify the sample ECL in the release library to meet your dataset naming requirements (snxmapop/samp).

Operator Option Descriptions

This section details all the SNXMAPOP operator interface options. The following table briefly summarizes each option.

Operator Option	**Description**
-d	Enable debug output
-h, --help	Provide usage and a list of options and commands

Figure 13. SNXMAPOP Operator Options

Operator Command Descriptions

This section details all the SNXMAPOP operator interface commands. The following table briefly summarizes each command. ALL is the default if a command is not entered.

Operator Command	Description
a[ll]	Show all registered offers (default)
p[arms]	Show configuration parameters
k[ey]	Show license key and status
l[oad]	Load license key from key file
dbgon	Enable debug messages in the user application
dbgoff	Disable debug messages in the user application

Figure 14. SNXMAPOP Operator Commands

ALL

Description

Displays a list of the pending SOFFERs in this SNXMAP waiting for SCONNECTs.

Format

snxmapop all

Examples

The following figure shows a sample ALL on an IBM AIX platform:

```
aix3# snxmapop all
Thu Jul 27 12:53:26 2017 snxmapop: connecting to snxmap addr=127.0.0.1:3919
Thu Jul 27 12:53:26 2017 snxmapop: snxmap status=0
Thu Jul 27 12:53:26 2017 snxmapop: snxmap returned 1 offers:
BFXJS    : 32000 : secure
```

Figure 15. ALL SNXMAPOP Command

PARMS

Description

Displays the current configuration parameters.

Format

snxmapop parms

Examples

The following figure shows a sample PARMS command output on an IBM AIX platform (output varies by platform):

```
aix3# snxmapop parms
Fri Oct  6 12:53:31 2017 snxmapop: connecting to snxmap addr=127.0.0.1:3919
Fri Oct  6 12:53:31 2017 snxmapop: snxmap status=0
    CNVERIFY=OFF
    DEBUG=OFF
    DEFBI=32768
    DEFBO=32768
    DNSRR=OFF
    IDLETO=6
    MAXBI=65535
    MAXBO=65535
    SMWAIT=15
    LOG=0
    SYSLOG=1
    SYSLOGFAC=local6
    LCLHOST=AIX3
    CERTFILE=/usr/share/nesi/bfx/cert.pem
    KEYFILE=/usr/share/nesi/bfx/key.pem
    CAFILE=/usr/share/nesi/bfx/certs/trusted.pem
    CAPATH=/usr/share/nesi/bfx/certs
    PORTNUM=1000
    PORTSTART=32000
```

Figure 16. PARMS SNXMAPOP Command

See Appendix G: Secure NetEx/IP Configuration File for defaults and meanings of the SNXMAP parameters.

KEY

Description

Displays the license key and status.

Format

snxmapop key

Examples

The following figure shows a sample KEY command output on an IBM AIX platform (output varies by platform):

```
aix3# snxmapop key
Fri Oct  6 13:02:47 2017 snxmapop: connecting to snxmap addr=127.0.0.1:3919
Fri Oct  6 13:02:47 2017 snxmapop: snxmap status=0
License Key     : DGXI-YAA6-AAAW-U5UD-5YKX-SFLG
Expiration      : Fri Dec 01 23:59:59 CST 2017
Not Operational : Sun Dec 31 23:59:59 CST 2017
```

Figure 17. KEY SNXMAPOP Command

LOAD

Description

Load license key from key file and display the license key and status.

Format

snxmapop load

Examples

The following figure shows a sample LOAD command output on an IBM AIX platform (output varies by platform):

```
aix3# snxmapop load
Fri Oct  6 13:02:23 2017 snxmapop: connecting to snxmap addr=127.0.0.1:3919
Fri Oct  6 13:02:24 2017 snxmapop: snxmap status=0
License Key     : DGXI-YAA6-AAAW-U5UD-5YKX-SFLG
Expiration      : Fri Dec 01 23:59:59 CST 2017
Not Operational : Sun Dec 31 23:59:59 CST 2017
```

Figure 18. LOAD SNXMAPOP Command

DBGON

Description

Enable debug messages in the user application.

Format

snxmapop dbgon

Examples

The following figure shows a sample DBGON command output on an IBM AIX platform (output varies by platform):

```
aix3 ~ 0]$ snxmapop dbgon
Thu Oct 26 14:04:23 2017 snxmapop: connecting to snxmap addr=127.0.0.1:3919
Thu Oct 26 14:04:23 2017 snxmapop: snxmap status=0
```

Figure 19. DBGON SNXMAPOP Command

DBGOFF

Description

Disable debug messages in the user application.

Format

snxmapop dbgoff

Examples

The following figure shows a sample DBGOFF command output on an IBM AIX platform (output varies by platform):

```
aix3 ~ 0]$ snxmapop dbgoff
Thu Oct 26 14:04:31 2017 snxmapop: connecting to snxmap addr=127.0.0.1:3919
Thu Oct 26 14:04:31 2017 snxmapop: snxmap status=0
```

Figure 20. DBGOFF SNXMAPOP Command

Appendix A: NRB Error Codes

When a NETEX request is issued, the results of the request are returned in one or both of two NRB fields, NRBSTAT and NRBIND. These fields are located at the beginning of the NRB to make their subsequent examination by high level language programs a simpler matter.

NRBSTAT is designed to indicate if an operation is in progress and whether it completed successfully. NRBIND is designed to indicate the type of information that arrived as the result of a read type command (OFFER or READ).

When the operation is accepted by the NETEX user interface, the value of NRBSTAT is set to a –1. Thus, the sign of this word is an "operation in progress" flag for all implementations.

If an operation completed successfully, NRBSTAT is returned as all zeroes. If a read–type command was issued, then an "indication" is set in NRBIND. The termination of a session is always indicated by a disconnect indication in NRBIND regardless of the request type.

If the operation did not complete successfully, then NRBSTAT contains a standard error code. NRBSTAT is represented as four decimal digits. The thousands digit denotes the origin of the error; the low order three digits identify the error type. The codes for error origin are as follows.

Code	Description
0xxx	NETEX general. Errors detected by the user interface that prohibit proper process of the command.
1xxx	Reserved for Legacy NetEx Driver level errors (no longer issued)
2xxx	Reserved for Legacy NetEx Transport level errors (no longer issued)
3xxx	Session level errors.
4xxx	Reserved for Legacy NetEx Network level errors (no longer issued)
5xxx-8xxx	Reserved for future NETEX functions
90xx	Reserved for errors returned by user exits on the local host
91xx	Reserved for errors returned by user exits on the remote host during the connection process.
9200-32767	Reserved for future NETEX functions.

Table 2. Origin of NRB Error Codes

Note the following when using these codes:

Some errors cause the loss of the connection or result in a connection not being established. Any status code that implies that the connection is no longer useful has a 6 (Disconnect Indication) returned in NRBSTAT. Any attempts to issue further requests to that connection have a x100 (no Nref) error returned to it.

All errors that result in the loss of the connection and a Disconnect Indication in NRBIND are indicated by an asterisk (*) following the error code number.

Note: A 0000 in field NRBSTAT means successful completion of the NETEX request. A -1 means that the request is still in progress.

The following sections describe the errors in numerical order starting with general NETEX error, followed by driver, transport, and session level errors.

General Errors

The following errors are general NETEX errors.

0000	Successful completion
0001	A read type operation completed normally within NETEX, but the Pdata buffer provided by the user was not large enough to hold the data. NRBLEN and NRBUBIT reflect the amount of data the other party intended to send. However, the amount of data moved to the user's program was only the amount of addressable units specified in NRBBUFL. NRBIND specifies the type of data sent to the user. Requests affected: OFFER, READ. The status of the connection is not affected.
0004	The request code (NRBREQ) is not valid. The operation is ignored, and the status of the connection specified by NRBNREF is not affected
0005	The buffer size specified (in NRBBUFL for read and NRBLEN for write) exceeds an implementation defined NETEX maximum. The operation is suppressed. The status of the connection is not affected.
0006	Offer name was not specified or contains non-alphanumeric characters
0007	Host name was not specified or contains non-alphanumeric characters
0011	A read-type operation completed normally within NETEX, but the Odata buffer provided by the user was not large enough to hold the data. NRBPROTL reflects the amount of data the other party intended to send; however, the amount of data moved to the user's program was only the amount of addressable units originally specified in NRBPROTL. NRBIND specifies the type of data sent to the user. Requests affected: OFFER, READ. The status of the connection is not affected
0021	A read-type operation completed normally within NETEX, but BOTH the Odata and Pdata buffers were too small to hold the incoming data. NRBLEN/NRBUBIT and NRBPROTL reflect the amount of data the other party intended to send; however, the amount of data moved to the user's program was only the amount of addressable units originally specified in NRBLEN and NRBPROTL. NRBIND specifies the type of data sent to the user. Request affected: OFFER, READ. The status of the connection is not affected.
0310	The user has attempted to re-use an NRB before a previous request issued with that NRB has completed. The request will be rejected. When the original request issued with that NRB completes, then the NRB will be once more updated with the status of that request.

Table 3. General NRB Error Codes

License Specific Errors

0610	Can't read license
0611	License is invalid
0612	License has expired
0613	License does not support TNP

Table 4. License Specific NRB Error Codes

Session Service Errors

2300	An SREAD request timed-out before a response was received on the network. The status of the connection is not affected.
3005	During a WRITE operation, the length of the buffer as specified by NRBLEN exceeds the maximum buffer size found in NRBBLKO. The WRITE operation is rejected. The connection remains outstanding.
3006	The length of PDATA sent on a CONNECT, CONFIRM, or DISCONNECT is greater than the maximum allowed. The request is rejected.
3100*	The Sref specified by NRBNREF is not in use or is not owned by this applications program. The request is rejected. The status of other connections owned by this application remains unchanged.
3101	On an SWRITE request, a DATAMODE was specified that is not supported on this host.
3201	A general TCP socket error occurred; review errors logged prior to this error for platform specific socket errors.
3300	An SOFFER request timed-out before a response was received on the network. When the SOFFER timed out, then the connection will not have taken place.
3390	NETEX received a TCP close with a 0 data length indicator
3402*	The remote application has failed to issue an SREAD request for a period of elapsed time (READTO) specified by the installation systems programmer on the remote host. The connection is terminated. A Disconnect Indication will be found in NRBIND.
3422	Session was disconnected by application. SOFFER terminates with a Disconnect Indication in NRBIND.
3501*	The PNAME specified (with the specified security mode) is not OFFERed on the HOST specified during the SCONNECT. The SCONNECT terminates with a Disconnect Indication in NRBIND.
3503*	The number of user session connections permitted by NETEX has been exceeded. Session service cannot be offered at this time. The SCONNECT or SOFFER is rejected.
3506*	The HOST specified in an SCONNECT request does not exist anywhere on the network generated by the installation systems programmer. The SDISCONNECT terminates with a Disconnect Indication in NRBIND.
3509*	The specified value of NRBBLKO exceeds an installation or implementation defined maximum. The connection request is rejected.
3510*	The specified value of NRBBLKI exceeds an installation or implementation defined maximum. The connection request is rejected.
3969	A memory request for a request block failed. Internal NRB could not be allocated. Session is unchanged.
3970	A memory request for a work area failed. Failed to initialize Secure NetEx/IP API global work area. No session can be established. NRBIND set to Disconnect indication.
3971	A memory request for a thread mutex failed. Failed to initialize Secure NetEx/IP API global work area. No session can be established. NRBIND set to Disconnect indication.

3972	An initialization request or a thread mutex failed. Failed to initialize Secure NetEx/IP API global work area. No session can be established. NRBIND set to Disconnect indication.
3973	A memory request for pthread condition failed. Failed to initialize Secure NetEx/IP API global work area. No session can be established. NRBIND set to Disconnect indication.
3974	An initialization request for a thread condition failed. Failed to initialize Secure NetEx/IP API global work area. No session can be established. NRBIND set to Disconnect indication.
3975	On an SOFFER or SCONNECT, the session control block could not be allocated. No session can be established. NRBIND set to Disconnect indication.
3980	A request to SNXMAP failed to complete. The session is terminated with a NRBIND set to Disconnect indication.
3981	A request to SNXMAP failed with a return code. The session is terminated with a NRBIND set to Disconnect indication.
3982	A request to create a new thread failed with the return code. The session is terminated with a NRBIND set to Disconnect indication.
3984	A request to create a new thread failed with the return code. The session is terminated with a NRBIND set to Disconnect indication.
3987	SNXMAP failed a memory request. The session is terminated with a NRBIND set to Disconnect indication.
3988	SNXMAP received an invalid request. The session is terminated with a NRBIND set to Disconnect indication.
3989	The length in the protocol header was incorrect. Secure NetEx/IP protocol failed. For SOFFER, the session is terminated with a NRBIND set to Disconnect indication. Otherwise the session is unchanged.
3990	The version in the protocol header was incorrect. Secure NetEx/IP protocol failed. For SOFFER, the session is terminated with a NRBIND set to Disconnect indication. Otherwise the session is unchanged.
3991	The header type in the protocol header was incorrect. Secure NetEx/IP protocol failed. For SOFFER, the session is terminated with a NRBIND set to Disconnect indication. Otherwise the session is unchanged.
3992	An error occurred with creating the encrypted connection. SSL failure. Secure NetEx/IP protocol failed. For SOFFER, the session is terminated with a NRBIND set to Disconnect indication. Otherwise the session is unchanged.
3995	A memory request failed for the GSK parameters. SSL failure. Secure NetEx/IP protocol failed. For SOFFER, the session is terminated with a NRBIND set to Disconnect indication. Otherwise the session is unchanged.

Table 5. Session Service NRB Error Codes

Appendix B: Secure NetEx Messages

This section contains a description of the messages issued by Secure NetEx/IP. These messages are displayed in the stdout/stderr file of the user's application if logging is set to the appropriate level. Message numbers are post fixed with a D, T, I or E to reflect the category as well. Refer to the specific platform installation/configuration Appendix for information on setting up messages.

Message numbers are categorized by the following:

SNX1000D – Debug (not documented in this manual; for NetEx Support only)

SNX2000T – Trace (not documented in this manual; for NetEx Support only)

SNX3xxxI – Informational

SNX4xxxE – Error

SNXxxxxC – Continuation of previous message

Messages may be prefixed with a date and timestamp of the following format if utilizing a logging facility:

Sun Jan 27 01:03:52 2017

Information Messages

SNX3100I %s"

Description: %s contains the version and tab level of the Secure NetEx/IP code

User Response:

None

SNX3101I csokcntx(%d): Setting nsswitch failed

Description: On an IBM zOS system, an attempt to define the name resolver search did not complete successfully. The system search order will be used instead.

%d uniquely identifies the internal session ID in use

User Response:

Insure the host names being used are resolved as intended

SNX3102I csokcntx(%d): RFCONNECT doportmap GET to %s timed out

Description: While processing a CONNECT request, a response was not received from the remote SNXMAP facility. The next IP address returned from the name resolver will be tried.

%d uniquely identifies the internal session ID in use

%s identifies the IP address that did not respond

User Response:

None

SNX3103I csokcntx(%d): RFCONNECT doportmap GET to %s error, errno=%d1

Description: On the connecting side, this message informs the user there was a error trying to connect to the remote SNXMAP facility to determine the OFFERing application port number. Previous error messages have more details.

%d uniquely identifies the internal session ID in use

%s IP address the remote NetEx hostname resolved to

%d1 System error when (trying to or) communicating with the remote host SNXMAP facility

User Response:

Review previous error messages.

SNX3104I csokcntx(%d): RFCONNECT doportmap GET to %s error, status=%d1

Description: On a connect request initiating communication to the SNXMAP facility on a remote host returned an error status.

%d uniquely identifies the internal session ID in use

%s IP address the remote Netex hostname resolved to

%d1 the nrb status returned

User Response:

Refer to NRB error codes for details regarding this issue.

SNX3105I csokcntx(%d): RFCONNECT securexfr=%d1, port=%d2

Description: Preparing to create the user connection.

%d uniquely identifies the internal session ID in use

%d1 transfer mode (0 – unsecure, 1 – secure)

%d2 remote offer port

User Response:

None

SNX3106I csokcntx(%d): COMAPI=%i, addr=%x

Description: On a Unisys OS2200 system, displays the TCP configuration information for this session.

%d uniquely identifies the internal session ID in use

%i identifies the COMAPI being used

%x identifies the local IP address (i.e. 0x7F000001) where local SNXMAP facility is configured for

User Response:

None

SNXI thread(%d): RFCONNECT connecting sock=%d1, rmtaddr=%s:%d2

Description: Connect request in progress; preparing for connect.

%d uniquely identifies the internal session ID in use

%d1 identifies the socket

%s the remote IP address (the OFFER application)

%d2 the remote port number (the OFFER application)

User Response:

None

SNX3202I thread(%d): setting secure opts

Description: On a Unisys OS2200 system, setup of secure options prior to establishing the connection to the remote host.

%d uniquely identifies the internal session ID in use

User Response:

None

SNX3203I thread(%d): Non secure transfer

Description: On a Unisys OS2200 system, indicates using a non-secure connection (connect/client).

%d uniquely identifies the internal session ID in use

User Response:

None

SNX3204I thread(%d): RFCONNECT connected sock=%d1, lcladdr=%s:%d2

Description: Connect request in progress; socket is connected.

%d	uniquely identifies the internal session ID in use
%d1	identifies the socket
%s	the remote IP address (the OFFER application)
%d2	the remote port number (the OFFER application)

User Response:

None

SNX3205I thread(%d): SSL connection using %s

Description: On a RFCONNECT, on a Unix system, the secure socket is connected using the cipher description (connect/client).

%d	uniquely identifies the internal session ID in use
%s	the cipher description

User Response:

None

SNX3206I thread(%d): RFOFFER request (%d1 secs)

Description: RFOFFER request processing.

%d	uniquely identifies the internal session ID in use
%d1	Offer timeout set by the application

User Response:

None

SNX3207I thread(%d): RFOFFER listening: sock=%d1, addr=%s:%d2

Description: Offer request; prepare to do the listen.

%d	uniquely identifies the internal session ID in use
%d1	identifies the socket
%s	the IP address (the OFFER application)
%d2	the port number (the OFFER application)

User Response:

None

SNX3208I thread(%d): setting secure opts

Description: On a Unisys OS2200 system, setup of secure options prior to the listen (Offer/server).

%d uniquely identifies the internal session ID in use

User Response:

None

SNX3209I thread(%d): Non secure transfer

Description: On a Unisys OS2200 system, not a secure connection (Offer/server).

%d uniquely identifies the internal session ID in use

User Response:

None

SNX3210I thread(%d): RFOFFER accepted: sock=%d1, newsock=%d2, rmtaddr=%s:%d3

Description: Processing Offer when Connect is received.

%d uniquely identifies the internal session ID in use

%d1 identifies the listen socket

%d2 identifies the new accept socket

%s the remote IP address (the CONNECT application)

%d3 the remote port number (the CONNECT application)

User Response:

None

SNX3211I thread(%d): SSL connection using %s

Description: On a RFOFFER, on a Unix system, the secure socket is connected using the cipher description (offer/server).

%d uniquely identifies the internal session ID in use

%s the cipher description

User Response:

None

SNX3301I Certificate is self-signed

Description: Self-signed certificates must be added to trust store (local verify).

User Response:

None

SNX3302I Certificate subject name: %s

Description: Local certificate subject name field (local cert verify).

%s local certificate subject name

User Response:

None

SNX3303I Peer certificate:

Description: Peer certificate data will follow.

User Response:

None

SNX3304I Issuer: %s

Description: Peer certificate issuer field

%s peer certificate issuer

User Response:

None

SNX3305I Subject: %s

Description: Peer certificate subject field

%s Peer certificate subject

User Response:

None

SNX3306I Not Before: %s

Description: Peer certificate ‘Not Before’ field

%s peer certificate ‘not before’

User Response:

None

SNX3307I Not After : %s

Description: Peer certificate ‘Not After’ field

%s peer certificate ‘not after’

User Response:

None

SNX3308I %s

Description: Peer certificate V3 extension fields (ex.SAN)

%s peer certificate V3 extension

User Response:

None

SNX3309I No peer certificate

Description: No peer certificate received during SSL handshake

User Response:

None

SNX3401I show_cert(%d): id=%d1(%s)

Description: On IBM zOS system this is the certificate to be used for this connection.

%d uniquely identifies the internal session ID in use

%d1 identifies the attribute in the certificate

%s contents of the attribute in the certificate

User Response:

None

SNX3402I show_cert(%d): elem_count=%d1

Description: On IBM zOS system this displays the number of the elements in the certificate to be used for this connection.

%d uniquely identifies the internal session ID in use

%d1 number of elements in the certificate

User Response:

None

SNX3403I show_cert(%d): i=%d1, cert_data_id=%d2, cert_data_l=%d3

Description: On IBM zOS system this displays the element id and data in the certificate to be used for this connection.

%d	uniquely identifies the internal session ID in use
%d1	index of certificate element
%d2	the element identifier in the certificate
%d3	identifies the length of the element in the certificate

User Response:

None

SNX3404I show_cert(%d): i=%d1, cert_data_id=%d2, cert_data_p='%s', cert_data_l=%d3

Description: On IBM zOS system this displays the attribute and data in the certificate to be used for this connection

%d	uniquely identifies the internal session ID in use
%d1	index of certificate element
%d2	the element identifier in the certificate
%s	the contents of the element in the certificate
%d3	identifies the length of the element in the certificate

User Response:

None

SNX3405I show_cert(%d): no cert data

Description: On an IBM zOS system, there was no certificate for this connection.

%d	uniquely identifies the internal session ID in use

User Response:

You may need to verify with your site security that the NetEx Hostname (IP address) has security credentials.

SNX3406I show_cert(%d): gsk_attribute_get_cert_info failure, rc=%d1

Description: On an IBM zOS system, an error was returned when attempting to get the certificate info.

%d	uniquely identifies the internal session ID in use
%d1	error returned on the call

User Response:

Contact Support@netex.com for details on the error. You may need to verify with your site security that the NetEx Hostname (IP address) has security credentials.

SNX3407I find_cert_data(%d): slen(%d1) > dlen(%d2), using dlen

Description: On an IBM zOS system, when evaluating the certificate the length of the element is longer than the size of the certificate.

%d	uniquely identifies the internal session ID in use
%d1	length of the certificate element data
%d2	length of the supplied area for the element data

User Response:

None

SNX3408I find_cert_data(%d): elem %d not found

Description: On an IBM zOS system, when evaluating the certificate the element to verify is not found in the certificate.

%d	uniquely identifies the internal session ID in use
%d1	the element to verify

User Response:

You may need to verify with your site security that the NetEx Hostname (IP address) has security credentials.

SNX3409I find find_cert_data(%d): no cert data

Description: On an IBM zOS system, no certificate data was returned for this NetEx Hostname (IP Address).

%d	uniquely identifies the internal session ID in use

User Response:

You may need to verify with your site security that the NetEx Hostname (IP address) has security credentials.

SNX3501I doportmap(%d): read response select timed out

Description: SNXMAP select read timeout on the socket (no response from SNXMAP for 10 seconds)

%d	uniquely identifies the internal session ID in use

User Response:

None

SNX3502I doportmap(%d): environment variable 'SNETEX_SERVICE' set port %d1

Description: On a Unix system this shows that an SNXMAP port number was specified in the environment.

%d	uniquely identifies the internal session ID in use
%d1	port that will be used to communicate with SNXMAP

User Response:

None

SNX3503I doportmap(%d): getservbyname '%s' returned port %d1

Description: Shows successful SNXMAP service name resolution.

%d	uniquely identifies the internal session ID in use
%s	service name
%d1	port that will be used to communicate with SNXMAP

User Response:

None

SNX3504I doportmap(%d): getservbyname '%s' returned NULL, using default port %d1

Description: Shows unsuccessful SNXMAP service name resolution. Default port will be used.

%d	uniquely identifies the internal session ID in use
%s	service name
%d1	port that will be used to communicate with SNXMAP

User Response:

None

SNX3900I no TCP config file '%s' : using defaults

Description: On a Unisys OS2200 system, no configuration file was found, defaults will be used instead.

%s	name of configuration file

User Response:

None

Error Messages

SNX4101E csokcntx(%d): RFCONNECT doportmap LICCHK timed out

Description: While processing a CONNECT request, a response was not received from the local SNXMAP facility within 5s.

%d uniquely identifies the internal session ID in use

User Response:

Verify the local SNXMAP is listening on the ‘snetex’ service port.

SNX4102E csokcntx(%d): RFCONNECT doportmap LICCHK socket error, errno=%d1: %s

Description: While processing a CONNECT request, a socket error was not received connecting to the local SNXMAP facility.

%d uniquely identifies the internal session ID in use

%d1 system error

%s system error description

User Response:

Refer to the system error description for an indication of the resolution.

SNX4103E csokcntx(%d): RFCONNECT only AF_INET address types are supported

Description: While processing a CONNECT request, the returned IP address for the remote NetEx hostname is not an IPv4 address.

%d uniquely identifies the internal session ID in use

User Response:

Insure the remote NetEx host names being used are resolved correctly.

SNX4104E csokcntx(%d): RFCONNECT do_req_snd_connection failed, rc=%d1

Description: On a Unix or Unisys system, while processing a CONNECT request there was a error returned when attempting to create the socket.

%d uniquely identifies the internal session ID in use

%d1 system error returned

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4105E csokcntx(%d): RFCONNECT scbreqsndsock accept error, errno=%d1: %s

Description: On a Unix or Unisys system, while processing a CONNECT request, the accept on the internal socket connection failed.

%d uniquely identifies the internal session ID in use

%d1 system error

%s system error description

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4106E csokcntx(%d): RFOFFER do_req_snd_connection failed, rc=%d1

Description: Description: On a Unix or Unisys system, while processing an OFFER request there was an error returned when attempting to create the socket.

%d uniquely identifies the internal session ID in use

%d1 system error returned

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4107E csokcntx(%d): RFOFFER do_req_snd_connection accept error, errno=%d1: %s

Description: On a Unix or Unisys system, while processing a OFFER request, the accept on the internal socket connection failed.

%d uniquely identifies the internal session ID in use

%d1 system error

%s system error description

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4108E csokcntx(%d): clean1() send failed errno=%d1: %s

Description: On a Unix or Unisys system, the send on the internal socket connection failed.

%d uniquely identifies the internal session ID in use

%d1 system error

%s system error description

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

167 SNX4109E csokcntx(%d): clean1() recv failed errno=%d1: %s

Description: On a Unix or Unisys system, the receive on the internal socket connection failed.

%d	uniquely identifies the internal session ID in use
%d1	system error
%s	system error description

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4110E csokcntx(%d): clean1(): !!Error allocating cleanup nrb!!

Description: Resources could not be allocated to clean up the connection.

%d	uniquely identifies the internal session ID in use

User Response:

The amount of memory available to the program is not sufficient.

SNX4111E csokcntx(%d): do_req_snd_connection() socket error, errno=%d1: %s

Description: On a Unix or Unisys system, the socket allocation failed.

%d	uniquely identifies the internal session ID in use
%d1	system error
%s	system error description

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4112E csokcntx(%d): do_req_snd_connection() bind error, errno=%d1: %s

Description: On a Unix or Unisys system, the bind for the internal socket connection failed.

%d	uniquely identifies the internal session ID in use
%d1	system error
%s	system error description

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages

.

SNX4113E csokcntx(%d): do_req_snd_connection() getsockname error, errno=%d1: %s

Description: On a Unix or Unisys system, could not get local address.

%d	uniquely identifies the internal session ID in use
%d1	system error
%s	system error description

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4114E csokcntx(%d): do_req_snd_connection() listen error, errno=%d1: %s

Description: On a Unix or Unisys system, the listen on the internal socket connection failed.

%d	uniquely identifies the internal session ID in use
%d1	system error
%s	system error description

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4115E csokcntx(%d): RFOFFER doportmap LICCHK timed out

Description: Connect to local SNXMAP timeout (no response within 5s).

%d	uniquely identifies the internal session ID in use

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4116E csokcntx(%d): RFOFFER doportmap LICCHK socket error, errno=%d1: %s

Description: Connect to local SNXMAP timeout (no response within 5s).

%d	uniquely identifies the internal session ID in use
%d1	system error
%s	system error description

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4201E thread(%d): do_req_rcv_connection failed, rc=%d1

Description: On a Unix or Unisys system, the connect to the internal listen failed.

%d	uniquely identifies the internal session ID in use
%d1	system error returned

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4202E thread(%d): scb->scbreq_q is NULL

Description: The thread signaled a request was ready, but no request was sent.

%d	uniquely identifies the internal session ID in use

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4203E thread(%d): RFCONNECT pipi_init error rc=%d

Description: On and IBM system, while processing a connect, the PIPI environment initialization failed.

%d	uniquely identifies the internal session ID in use
%d1	system error returned

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4204E thread(%d): RFCONNECT socket error, errno=%d1, stat=%d2: %s

Description: While processing a Connect request, the connect to the remote offer socket received an error.

%d	uniquely identifies the internal session ID in use
%d1	system error returned
%d2	status
%s	system error description

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4205E thread(%d): SSL setsockopt error, errno=%d1, stat=%d2: %s

Description: On a Unisys only system, the set secure socket options failed.

%d	uniquely identifies the internal session ID in use
%d1	system error returned
%d2	status
%s	system error description

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4206E thread(%d): RFCONNECT keepalive setsockopt error

Description: While processing a Connect, the keepalive set socket option failed (connect/client).

%d	uniquely identifies the internal session ID in use

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4207E thread(%d): RFCONNECT connect error, errno=%d1, stat=%d2: %s

Description: While processing a Connect, the connect to offer on remote host failed.

%d	uniquely identifies the internal session ID in use
%d1	system error returned
%d2	status
%s	system error description

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4208E thread(%d): RFCONNECT getsockname error, errno=%d1, stat=%d2: %s

Description: While processing a Connect, could not get the local address.

%d	uniquely identifies the internal session ID in use
%d1	system error returned
%d2	status
%s	system error description

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4209E thread(%d): RFCONNECT setup_mvsgsk_env error, errno=%d1

Description: On IBM systems, while processing a Connect, received a system error when initializing the environment.

%d uniquely identifies the internal session ID in use

%d1 system error returned

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4210E thread(%d): RFCONNECT setup_mvsgsk_env error, stat=%d1

Description: On IBM systems, while processing a Connect, environment initialization failed.

%d uniquely identifies the internal session ID in use

%d1 status

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4211E thread(%d): RFCONNECT setup_mvsgsk_soc error, errno=%d1

Description: On IBM systems, while processing a Connect, socket initialization failed with a system error.

%d uniquely identifies the internal session ID in use

%d1 system error returned

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4212E thread(%d): RFCONNECT setup_mvsgsk_soc error stat=%d1

Description: On IBM systems, while processing a Connect, socket initialization failed.

%d uniquely identifies the internal session ID in use

%d1 status

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4213E thread(%d): RFCONNECT find_cert_data rc=%d1

Description: On IBM systems, while processing a Connect, unable to find peer certificate common name.

%d uniquely identifies the internal session ID in use

%d1 status

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4214E thread(%d): RFCONNECT ssl_init error, rc=%d1

Description: On Unix systems, while processing a Connect, a SSL initialization error was detected (connect/client).

%d uniquely identifies the internal session ID in use

%d1 status

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4215E thread(%d): RFCONNECT SSL_new failed

Description: On Unix systems, while processing a Connect, a SSL session allocation error was received (connect/client).

%d uniquely identifies the internal session ID in use

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4216E thread(%d): RFCONNECT SSL_set_fd failed

Description: On Unix systems, while processing a Connect, SSL session socket I/O assignment error was detected (connect/client).

%d uniquely identifies the internal session ID in use

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4217E thread(%d): RFCONNECT SSL_connect error, rc=%d1

Description: On Unix systems, while processing a Connect, an SSL session error was detected (connect/client).

%d uniquely identifies the internal session ID in use

%d1 status

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4218E thread(%d): RFCONNECT SSL_host check failed

Description: While processing a Connect, an SSL peer certificate Common Name error was detected (connect/client).

%d uniquely identifies the internal session ID in use

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4219E thread(%d): RFOFFER pipi_init error rc=%d1

Description: On IBM systems, while processing an Offer, the PIPI environment initialization failed.

%d	uniquely identifies the internal session ID in use
%d1	status

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4220E thread(%d): BIND error, errno=%d1, stat=%d2: %s

Description: While processing an Offer, the bind for the offer port failed.

%d	uniquely identifies the internal session ID in use
%d1	system error returned
%d2	status
%s	system error description

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4221E thread(%d): SSL setsockopt error, errno=%d1, stat=%d2: %s

Description: On Unisys systems, while processing an Offer, the set secure connection failed (offer/server).

%d	uniquely identifies the internal session ID in use
%d1	system error returned
%d2	status
%s	system error description

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4222E thread(%d): listen failed, errno=%d1, stat=%d2: %s

Description: The listen on the offer port failed.

%d	uniquely identifies the internal session ID in use
%d1	system error returned
%d2	status
%s	system error description

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4223E thread(%d): RFOFFER select on portmapsock error, errno=%d1, stat=%d2: %s

Description: While processing an Offer the select on SNXMAP connection failed.

%d	uniquely identifies the internal session ID in use
%d1	system error returned
%d2	status
%s	system error description

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4224E thread(%d): RFOFFER select error, errno=%d1, stat=%d2: %s

Description: While processing an Offer the wait for the remote host connection failed.

%d	uniquely identifies the internal session ID in use
%d1	system error returned
%d2	status
%s	system error description

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4225E thread(%d): RFOFFER accept error, errno=%d1, stat=%d2: %s

Description: While processing an Offer the accept for the remote host connection failed.

%d	uniquely identifies the internal session ID in use
%d1	system error returned
%d2	status
%s	system error description

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4226E thread(%d): RFOFFER keepalive setsockopt error

Description: While processing an Offer the set keepalive socket option failed (offer/server).

%d	uniquely identifies the internal session ID in use

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4227E thread(%d): RFOFFER setup_mvsgsk_env error errno=%d1

Description: On IBM systems, while processing an Offer the GSK environment initialization failed (offer/server).

%d uniquely identifies the internal session ID in use

%d1 system error returned

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4228E thread(%d): RFOFFER setup_mvsgsk_env error stat=%d1

Description: On IBM systems, while processing an Offer the GSK environment initialization failed.

%d uniquely identifies the internal session ID in use

%d1 status

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4229E thread(%d): RFOFFER setup_mvsgsk_soc error, errno=%d1

Description: On IBM systems, while processing an Offer the socket initialization failed.

%d uniquely identifies the internal session ID in use

%d1 system error returned

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4230E thread(%d): RFOFFER setup_mvsgsk_soc error, stat=%d1

Description: On IBM systems, while processing an Offer the GSK socket initialization failed.

%d uniquely identifies the internal session ID in use

%d1 status

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4232E thread(%d): RFOFFER ssl_init error, rc=%d1

Description: On Unix systems, while processing an Offer an SSL initialization error was detected (offer/server).

%d uniquely identifies the internal session ID in use

%d1 status

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4233E thread(%d): RFOFFER SSL_new failed

Description: On Unix systems, while processing an Offer an SSL session allocation error was detected (offer/server).

%d uniquely identifies the internal session ID in use

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4234E thread(%d): RFOFFER SSL_set_fd failed

Description: On Unix systems, while processing an Offer an SSL session socket I/O assignment error was detected (offer/server).

%d uniquely identifies the internal session ID in use

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4235E thread(%d): RFOFFER SSL_accept error

Description: On Unix systems, while processing an Offer an SSL session accept error was detected (offer/server).

%d uniquely identifies the internal session ID in use

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4236E thread(%d): setsockopt TCP_KEEPIDLE failed, errno=%d1: %s

Description: On a Unix system, the set keepalive idle time socket option failed.

%d uniquely identifies the internal session ID in use

%d1 system error returned

%s system error description

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4237E thread(%d): setsockopt TCP_KEEPCNT failed, errno=%d1: %s

Description: On a Unix system, the set keepalive probe count socket option failed.

%d	uniquely identifies the internal session ID in use
%d1	system error returned
%s	system error description

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4238E thread(%d): setsockopt TCP_KEEPINTVL failed, errno=%d1: %s

Description: On a Unix system, the set keepalive probe interval time socket option failed.

%d	uniquely identifies the internal session ID in use
%d1	system error returned
%s	system error description

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4239E thread(%d): setsockopt SO_KEEPALIVE failed, errno=%d1: %s

Description: The set keepalive socket option failed.

%d	uniquely identifies the internal session ID in use
%d1	system error returned
%s	system error description

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4240E thread(%d): RFOFFER socket error, errno=%d1, stat=%d2: %s

Description: Offer socket allocation error.

%d	uniquely identifies the internal session ID in use
%d1	system error returned
%d2	status
%s	system error description

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4241E thread(%d): RFOFFER bind to addr=%s:%d1 errno 111 %d2 times, max is 5

Description: On an IBM system, bind for offer port failed with errno 111 more than 5 times.

%d	uniquely identifies the internal session ID in use

%s bind address

%d1 bind port

%d2 number of bind attempts

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4242E thread(%d): RFOFFER no port available in range (%d1-%d2)

Description: Configured port range contains no open ports for offers.

%d uniquely identifies the internal session ID in use

%d1 first port number

%d2 last port number

User Response:

Increase PORTNUM or use ephemeral ports (PORTSTART=0).

SNX4301E %s

Description: On Unix systems, an OpenSSL error was detected.

%s The error strings will have the following format:

[pid]:error:[error code]:[library name]:[function name]:[reason string]:[file name]:[line]:[optional text message]

pid and error code are 8 digit hexadecimal number

library name, function name and reason string are ASCII text, as is optional text message f one was set for the respective error code. If there is no text string registered for the given error code, the error string will contain the numeric code.

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4302E Certificate error %d at depth %d1: %s

Description: A local certificate verify error was detected.

%d error id

%d1 certificate depth

%s error description

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4303E Certificate error %d at depth %d1: %s

Description: During an SSL handshake a peer certificate verify error was detected.

%d error id

%d1 certificate depth

%s error description

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4304E No certificate file

Local certificate file not specified (see SNXMAP configuration).

Description: Local certificate file not specified.

User Response:

Check the SNXMAP configuration file for proper specification.

SNX4305E BIO_new failed

Description: During the local certificate verify, a memory allocation error was detected.

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4306E BIO_read_filename failed

Description: During the local certificate verify, an error was detected reading the certificate file.

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4307E PEM_read_bio failed

Description: During the local certificate verify, a PEM certificate data read error was detected.

User Response:

Check certificate file for valid PEM format.

SNX4308E SSL_CTX_get_cert_store failed

Description: During the local certificate verify, an SSL context certificate store error was detected.

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4309E X509_STORE_CTX_new failed

Description: During the local certificate verify, a memory allocation error was detected.

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4310E X509_STORE_CTX_init failed

Description: During the local certificate verify, a store initialization error was detected.

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4311E ssl_init: init_locks() failed

Description: Failed to initialize the SSL thread-safe lock controls.

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4312E ssl_init: SSL_CTX_new failed

Description: Failed to allocate the SSL context.

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4313E ssl_init: SSL_CTX_set_cipher_list failed

Description: Failed to set the SSL supported cipher list.

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4314E ssl_init: Failed to load trusted CA certificates

SNX4314C ssl_init: Check CAFILE and/or CAPATH are correct

Description: Failed to set the locations for the trusted CA certificates.

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4315E ssl_init: Certificate verification failed (%s)

Description: Verify of local certificate failed.

%s certificate file

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4316E ssl_init: SSL_CTX_use_certificate_file (%s) error

Description: Failed to load certificate file.

%s certificate file

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4317E ssl_init: SSL_CTX_use_PrivateKey_file (%s) error

Description: Failed to load private key file.

%s key file

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4318E ssl_init: Private key does not match the certificate

Description: Certificate and private key are inconsistent.

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4401E gsksetup(%d): req %d1 gsk_environment_open error, rc=%d2

Description: On IBM systems, an error was detected when opening the GSK environment.

%d uniquely identifies the internal session ID in use

%d1 1: connect, 129:offer

%d2 error returned

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4402E gsksetup(%d): req %d1 gsk_attribute_set_enum PROTOCOL_SSLV2 ON error, rc=%d2

Description: On IBM systems, an error was detected when attempting to set the protocol to SSL V2.

%d uniquely identifies the internal session ID in use

%d1 1:connect, 129:offer

%d2 error returned

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4403E gsksetup(%d): req %d1 gsk_attribute_set_enum PROTOCOL_SSLV3 ON error, rc=%d2

Description: On IBM systems, an error was detected when attempting to set the protocol to SSL V3.

%d uniquely identifies the internal session ID in use

%d1 1:connect, 129:offer

%d2 error returned

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4404E gsksetup(%d): req %d1 gsk_attribute_set_enum PROTOCOL_TLSV1 ON error, rc=%d2

Description: On IBM systems, an error was detected when attempting to set the protocol to TLS V1.

%d uniquely identifies the internal session ID in use

%d1 1:connect, 129:offer

%d2 error returned

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4405E gsksetup(%d): req %d1 gsk_attribute_set_buffer KEYRING_FILE '%s' error, rc=%d2

Description: On IBM systems, an error was detected when attempting to set the keyring filename.

%d uniquely identifies the internal session ID in use

%d1 1:connect, 129:offer

%d2 error returned

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4406E gsksetup(%d): req %d1 gsk_attribute_set_buffer KEYRING_PW '%s' error, rc=%d2

Description: On IBM systems, an error was detected when attempting to set the keyring password.

%d uniquely identifies the internal session ID in use

%d1 1:connect, 129:offer

%s the keyring password that is being used

%d2 error returned

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4407E gsksetup(%d): req %d1 gsk_environment_init error, rc=%d2

Description: On IBM systems, an error was detected when attempting initialize the GSK environment.

%d uniquely identifies the internal session ID in use

%d1 1:connect, 129:offer

%d2 error returned

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4408E gsksetup(%d): req %d1 gsk_secure_socket_open error, rc=%d2

Description: On IBM systems, an error was detected when attempting to open the socket.

%d uniquely identifies the internal session ID in use

%d1 1:connect, 129:offer

%d2 error returned

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4409E gsksetup(%d): req %d1 gsk_attribute_set_numeric_value GSK_FD (sock %d2) error, rc=%d3

Description: On IBM systems, an error was detected when attempting to associate the socket with GSK.

%d uniquely identifies the internal session ID in use

%d1 1:connect, 129:offer

%d2 the socket number to be associated with the connection

%d3 error returned

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4410E gsksetup(%d): req %d1 gsk_attribute_set_buffer KEYRING_LABEL '%s' error, rc=%d2

Description: On IBM systems, an error was detected when attempting to set the keyring label.

%d uniquely identifies the internal session ID in use

%d1 1:connect, 129:offer

%s the keyring label for the certificate to be used

%d2 error returned

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4411E gsksetup(%d): req %d1 gsk_attribute_set_enum SESSION_TYPE SERVER_SESSION error, rc=%d2

Description: On IBM systems, an error was detected when attempting to set the session type to a server session.

%d uniquely identifies the internal session ID in use

%d1 1:connect, 129:offer

%d2 error returned

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4412E gsksetup(%d): req %d1 gsk_attribute_set_enum SESSION_TYPE CLIENT_SESSION error, rc=%d2

Description: On IBM systems, an error was detected when attempting to set the session to a client session.

%d uniquely identifies the internal session ID in use

%d1 1:connect, 129:offer

%d2 error returned

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4413E gsksetup(%d): req %d1 gsk_attribute_set_buffer V2_CIPHER_SPECS NULL error, rc=%d2

Description: On IBM systems, an error was detected when attempting to set the connection ciphers.

%d uniquely identifies the internal session ID in use

%d1 1:connect, 129:offer

%d2 error returned

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4414E gsksetup(%d): req %d1 gsk_attribute_set_buffer V3_CIPHER_SPECS NULL error, rc=%d2

Description: On IBM systems, an error was detected when attempting to set the connection ciphers.

%d uniquely identifies the internal session ID in use

%d1 1:connect, 129:offer

%d2 error returned

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4415E gsksetup(%d): req %d1 gsk_attribute_set_buffer V3_CIPHER_SPECS_EXPANDED '0035002F000A' error, rc=%d2

Description: On IBM systems, an error was detected when attempting to set the connection ciphers.

%d uniquely identifies the internal session ID in use

%d1 1:connect, 129:offer

%d2 error returned

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4416E gsksetup(%d): req %d1 gsk_attribute_set_enum V3_CIPHERS V3_CIPHERS_CHAR4 error, rc=%d2

Description: On IBM systems, an error was detected when attempting to set the connection ciphers

%d uniquely identifies the internal session ID in use

%d1 1:connect, 129:offer

%d2 error returned

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4417E gsksetup(%d): req %d1 gsk_secure_socket_init error, rc=%d2

Description: On IBM systems, an error was detected when attempting to initialize the secure socket.

%d uniquely identifies the internal session ID in use

%d1 1:connect, 129:offer

%d2 error returned

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4418E gsksetup(%d): req %d1 gsk_attribute_set_enum CLIENT_AUTH_TYPE_CLIENT_AUTH_FULL_TYPE error, rc=%d2

Description: On IBM systems, an error was detected when attempting to initialize attribute of a secure socket.

%d uniquely identifies the internal session ID in use

%d1 1:connect, 129:offer

%d2 error returned

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4419E gsksetup(%d): req %d1 gsk_attribute_set_enum CLIENT_AUTH_ALERT_CLIENT_AUTH_NOCERT_ALERT_ON error, rc=%d2

Description: On IBM systems, an error was detected when attempting to initialize attribute of a secure socket.

%d uniquely identifies the internal session ID in use

%d1 1:connect, 129:offer

%d2 error returned

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4420E pipi_sub_call(%d): CEEPIPI_call_sub for func %d1 (%s) failed - rc is %d2

Description: On IBM systems, an error was detected when attempting to make the CEEPIPI call.

%d uniquely identifies the internal session ID in use

%d1 the attempted socket function number

%s the attempted socket function name

%d2 error returned

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4421E pipi_sub_call(%d): CEEPIPI_call_sub for func %d1 (%s) subretc < 0, errno=%d2

Description: On IBM systems, an error was detected when attempting to make the CEEPIPI call.

%d uniquely identifies the internal session ID in use

%d1 the attempted socket function number

%s the attempted socet function name

%d2 error returned

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4422E pipi_init(%d): calloc of gskcall_parms failed

Description: On IBM systems, an error was detected when attempting to allocate the CEEPIPI call control block.

%d uniquely identifies the internal session ID in use

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4423E pipi_init(%d): Out of memory allocating pre-init table

Description: On IBM systems, out of memory was detected when attempting to allocate the pre-init table.

%d uniquely identifies the internal session ID in use

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4424E pipi_init(%d): __CEEPIPI_init_sub() failed - rc is %d

Description: On IBM systems, out of memory was detected when attempting to initialize the CEEPIPI interface.

%d uniquely identifies the internal session ID in use

%d1 error returned

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4425E pipi_term(%d): __CEEPIPI_term() failed - rc is %d1

Description: On IBM systems, out of memory was detected when attempting to terminate the CEEPIPI interface.

%d uniquely identifies the internal session ID in use

%d1 error returned

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4426E pipi_sub_call(%d): __CEEPIPI_call_sub for func %d1 (%s) subretc=%d2 (%s1)

Description: On IBM systems, out of memory was detected when attempting to terminate the CEEPIPI interface.

%d uniquely identifies the internal session ID in use

%d1 function number

%s function name

%d2 error returned

%s1 error description

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4501E doportmap(%d): socket error, errno=%d1: %s

Description: Socket allocation failed.

%d	uniquely identifies the internal session ID in use
%d1	system error
%s	system error description

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4502E doportmap(%d): connect error, errno=%d1: %s

Description: SNXMAP connect failure.

%d	uniquely identifies the internal session ID in use
%d1	system error
%s	system error description

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4503E doportmap(%d): send request error, errno=%d1: %s

Description: SNXMAP send failure.

%d	uniquely identifies the internal session ID in use
%d1	system error
%s	system error description

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4504E doportmap(%d): read response select error, errno=%d1: %s

Description: SNXMAP select failure.

%d	uniquely identifies the internal session ID in use
%d1	system error
%s	system error description

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4505E doportmap(%d): recv response error, errno=%d1: %s

Description: SNXMAP receive failure.

%d	uniquely identifies the internal session ID in use
%d1	system error
%s	system error description

User Response:

Contact Support@netex.com for details on the error. Review the previous error messages.

SNX4506E doportmap(%d): could not allocate %d1 bytes

Description: On behalf of the CONNECT or OFFER, an out of memory condition is detected by the local SNXMAP facility.

%d	uniquely identifies the internal session ID in use
%d1	amount of data required on the allocation

User Response:

Retry the program; system resources may have freed up.

SNX4900E error allocating tcp config environment

Description: On a Unisys OS220 system, unable to allocate memory for the TCP configuration environment.

User Response:

Retry the program; resources may have freed up.

SNX4901E error reading config file '%s'

Description: While processing the configuration file, parsing problems were encounterd. The errors were logged before this message.

User Response:

Correct the parameters in error

Appendix C: H214 for z Series/zOS Installation

Prerequisites

The following are hardware and software prerequisites for installing the H214 product.

- A z Series compatible system running a supported zOS release. Review the website for supported OS distributions.
- At least one other processor on the network running Secure NetEx/IP software. This processor should be connected with another Secure NetEx/IP (not required for intra-host test/evaluation).
- Customers must obtain a software KEY from Support@netex.com prior to running the H214 software. Customers must contact Support@netex.com with the customer site name, hostname, and the Secure NetEx/IP product designator (e.g., H214). Support@netex.com will supply the necessary key once this information has been received. The customer needs to place this key into the NESikeys file as discussed later in this section.
- (Required only to use secure job and/or data transfers) Use the gskkyman utility in order to create a key database which must contain the required client and server certificates and any necessary remote server certificates. The name of the key database, its password, and the client and server certificate labels must be specified in the SNXMAP configuration file.

All requirements for the equipment listed above must be met before proceeding with the installation.

Hardware Installation

Install and verify proper operation of the appropriate operating system.

Accessing the H214 software distribution

The H214 Secure NetEx/IP software is available as an binary XMIT file which may be downloaded from NetEx. Contact Support@netex.com to request the download link.

Obtaining the Software Key

As part of the installation process, a software key must be obtained from Network Executive Software, Inc. This software key is based on the serial number of the CPU on which Secure NetEx/IP will be used, and on the particular features supported by H214, and will authorize Secure NetEx/IP to be used on a particular LPAR.

The software key can be obtained in advance by using the following procedure:

Issue the following command on the z/OS system on which Secure NetEx/IP will be installed:

```
D M=CPU
```

When this command is issued, it will display the

CPU serial number

LPAR NAME (LP NAME)

LPAR ID (LP ID)

This command should be issued on each LPAR in which H214 will be executed.

An example of the D M=CPU command is shown in Figure 21.

```
D M=CPU
IEE174I 14.27.09 DISPLAY M 981
PROCESSOR STATUS
ID  CPU                  SERIAL
00  +                     01BC7F2096
01  +                     01BC7F2096
02  N

CPC ND = 002096.R07.IBM.02.00000003BC7F
CPC SI = 2096.C02.IBM.02.000000000003BC7F
CPC ID = 00
CPC NAME = P003BC7F
LP NAME = ZOS1       LP ID =  1
CSS ID  = 0
MIF ID  = 1

+ ONLINE    - OFFLINE    . DOES NOT EXIST    W WLM-MANAGED
N NOT AVAILABLE

CPC ND  CENTRAL PROCESSING COMPLEX NODE DESCRIPTOR
CPC SI  SYSTEM INFORMATION FROM STSI INSTRUCTION
CPC ID  CENTRAL PROCESSING COMPLEX IDENTIFIER
CPC NAME CENTRAL PROCESSING COMPLEX NAME
LP NAME  LOGICAL PARTITION NAME
LP ID    LOGICAL PARTITION IDENTIFIER
CSS ID   CHANNEL SUBSYSTEM IDENTIFIER
MIF ID   MULTIPLE IMAGE FACILITY IMAGE IDENTIFIER
```

Figure 21. Output display of 'D M=CPU' command

Note: if keys are needed for a machine on which the 'D M=CPU' command cannot be issued (e.g. it is a new machine that is not yet installed, or it is an offsite third-party DR system), you must still provide the same information (machine serial number, model, and LPAR names) in order for the key to get generated.

Contact Network Executive Software, Inc. by using either of the following methods:

telephone at (800) 854-0359

email: support@netex.com

Please provide the following information:

Customer name

CPU serial number (entire 10 digits)

LPAR name(s)

NetEx/IP product being installed (H214)

Network Executive Software, Inc. will generate the key(s) and return them by e-mail. This key is required during the installation process.

Installation Process

This section describes the installation procedure for the H214 NetEx/IP Release 1.0 distribution.

The following steps outline the installation process. Before proceeding with the installation, please read the Memo to Users accompanying the distribution for any additions or changes to the installation instructions.

Step 2 Obtain the H214 distribution file.

Step 3 Upload the distribution file to z/OS.

Step 4 TSO RECEIVE the distribution file.

Step 5 Modify and Submit the SNXINST job on z/OS.

Step 6 Check for required updates.

Step 7 Obtain the H214 software key.

Step 8 Review the H214 initialization parameters.

Step 9 Authorize the SNXALOAD load library.

Step 10 Define SNETEX Service

Step 11 (Optional) Update Policy Agent

Step 1. Step 12 (Optional) System Performance Consideration

Consider updating the SCHED member in your parmlib to run the snxmap program as noswappable.

PPT PGMNAME(SNXMAP) NOSWAP

Consider your Workload Manager configuration to insure the SNXMAP task will run at a dispatch level that is appropriate for your system. This is a server application and should run at a server level priority. If you experience NRBSTATs of 3980 while communicating with the SNXMAP program, an internal TCP request did not complete in 10 seconds. This is usually a configuration issue or the system is very busy and could not service the request.

Review Installed JCL

Step 14 Start SNXMAP

Step 15 (Optional) Execute the SNXMVEAT Program

Step 16 (Optional) Execute the SNXMVGEN Program

Step 2. Obtain the H214 distribution file.

The distribution file 'h214.bin' can be downloaded from Network Executive Software. Contact support@netex.com to obtain the download instructions.

Step 3. Upload the distribution file to z/OS.

FTP (binary mode) the h214.bin file to the z/OS system as follows:

Connect via FTP to your z/OS system.

Change the directory to your desired high level qualifier:

cd ‘high-level-qualifier’

If necessary, change the location of your local directory to the location of the distribution file:

lcd ‘directory-name’

Set the required attributes for the file:

quote site lrecl=80 blksize=3120 recfm=fb prim=5000 sec=200 blocks

Transfer the distribution file in bin mode:

bin

put h214.bin distpkg.xmit

Quit your FTP client

Using the above names results in the distribution file residing on z/OS as the following file:

‘high-level-qualifier.DISTPKG.XMIT’

Step 4. TSO RECEIVE the distribution file.

Issue the TSO PROFILE PROMPT command to be sure prompting is allowed. Then issue the TSO RECEIVE command against the distribution file uploaded in “Upload the distribution file to z/OS.” as follows:

RECEIVE INDSNAME(‘dsn’)

where ‘dsn’ specifies the name of the distribution file that was FTP’d to z/OS.

(for example: RECEIVE INDSNAME(‘high-level-qualifier.DISTPKG.XMIT’)

The RECEIVE command will issue the following prompt:

Enter restore parameters or ‘DELETE’ or ‘END’

Reply with:

DSN(‘dsn1’)

where ‘dsn1’ specifies the name of a PDS distribution library that will be created from ‘dsn’.

(for example: DSN(‘high-level-qualifier.DFILE’)

The resulting ‘high-level-qualifier.DFILE’ dataset is a PDS distribution library that contains three members:

COMPDD

SNXINST

SNX010

SNXINST is a job that uses SNX010 to create the H214 libraries and performs the installation.

Step 5. Modify and Submit the SNXINST job on z/OS.

The SNXINST installation job consists of three phases:

LOAD Allocates the H214 base installation datasets and loads the datasets from the downloaded distribution.

EDIT Automatically tailors the H214 SNXMAP startup proc and startup JCL.

COPY Copies the tailored SNXMAP startup proc to the specified PROCLIB library.

Tailor the installation job in SNXINST.

Warning: DO NOT ISSUE "CHANGE xxx ALL" commands against SNXINST. Change the keyword values on an individual basis only.

Change the following to your site requirements:

The JOB card

The unit name "UNIT=(SYSALLDA,,DEFER)" on the WORK DD card. Change SYSALLDA to a valid unit name matching your site requirements.

Review and tailor the installation parameters contained in SNXINST. At a minimum, the following parameters should be specified:

START(BEGIN)

STOP(END)

HLQ(hlqname)

DFILE(dfilename)

Submit the SNXINST job. This job will allocate and load the following H214 distribution libraries:

hlq.SNXCTL	Base control library
hlq.SNXMAC	Base macro library (limited distribution)
hlq.SNXOBJ	Base object library (limited distribution)
hlq.SNXLOAD	Base load library
hlq.SNXALOAD	Base authorized load library (for SNXMAP)

SNXINST installation parameters:

HELP Default value: NO

Allowed values: YES | NO

The HELP parameter is used to produce a description of the H214 z/OS NetEx/IP installation parameters and their usage. The value "YES" will only produce the HELP output, and no other installation job phases will be executed.

START Default value: BEGIN

Allowed values: BEGIN | LOAD | EDIT | COPY

The START parameter is used to determine which phase the H214 installation job will be started at. It can be used in conjunction with the STOP parameter to cause only a portion of the installation job to be executed.

STOP Default value: END

Allowed values: LOAD | EDIT | COPY | END

The STOP parameter is used to determine which phase the H214 installation job will be stopped at. It can be used in conjunction with the START parameter to cause only a portion of the installation job to be executed. For example, to only execute the COPY phase, code the START and STOP parameters as:

START(COPY)

STOP(COPY)

HLQ Default value: SNX.H2140100

Allowed values: Any valid data set name qualifier

The HLQ parameter is used to provide the high level qualifier of the data set names used by the installation job. By default, an HLQ of SNX.H2140100 is used, which results in all of the datasets identified in step 0 on page 106 being created using names in the following format:

SNX.H2140100.SNXCTL Base control library

Note: The data sets defined by the HLQ parameter will be deleted and recreated by the LOAD phase when performing an installation.

Example: HLQ(SNX.H2140100)

COMPRESS Default value: NO

Allowed values: YES | NO

The COMPRESS parameter is used to determine whether or not the IEBCOPY utility will be invoked to compress the H214 installation data sets before the data set is updated by the installation job. The COMPRESS parameter will only cause a compress of the H214 data sets. Other data sets used by the installation job will not be compressed. System data sets used by the COPY phase will not be compressed prior to the copies.

Example: COMPRESS(NO)

SYSOUT Default value: *

Allowed values: A-Z, 0-9, *

The SYSOUT parameter is used to provide the JES SYSOUT class for utility output (IEBCOPY, etc.). The value "*" will cause the installation job MSGCLASS SYSOUT class to be used for utility output.

Example: SYSOUT(A)

DFILE Default value: SNX.SNX.DFILE

Allowed values: Any valid fully qualified data set name

The DFILE parameter is used to provide the name of the dataset that was created when the FTP downloaded distribution file was TSO RECEIVE'd in “TSO RECEIVE the distribution file.”

Example: DFILE(SYSP.PROD.DFILE)

VOLUME Default value: ‘’ (none)

Allowed values: Any valid direct access (DASD) volume name

The VOLUME parameter is used to provide the z/OS volume name which will be used to allocate the H214 installation data sets.

Note: If an SMS policy is in effect for the HLQ name specified, the UNIT and VOLUME parameters can be omitted.

Example: VOLUME(PROD01)

UNIT Default value: ‘’ (none)

Allowed values: Any valid direct access (DASD) unit name

The UNIT parameter is used to provide the z/OS unit name which will be used to allocate the H214 installation data sets.

Note: If an SMS policy is in effect for the HLQ name specified, the UNIT and VOLUME parameters can be omitted.

Example: UNIT(SYSDA)

SNXPROC Default value: SNXMAP

Allowed values: Any valid JCL procedure name

The SNXPROC parameter is used to provide the name for the z/OS started task used to run SNXMAP. This parameter will also be used as the SNXPROC member name in the dataset defined by the PROCLIB parameter.

Example: SNXPROC(SNXMAP)

SNXMAPCF Default value: hlq.SNXCTL(SNXMAPCF)

Allowed values: Any valid fully qualified z/OS dataset name

The SNXMAPCF parameter is used to provide the fully qualified name of the SNXMAP initialization file. This name will be added to the NETEX procedure that is built when performing

theinstallation. Example:
SNXMAPCF(SNX.H2140100.DISTCTL(SNXMAPCF)

PRODCONF Default value: hlq.SNXCTL(PRODCONF)

Allowed values: Any valid fully qualified z/OS dataset name

The PRODCONF parameter is used to provide the fully qualified name of an existing NetEx/IP PRODCONF license key file. This name will be added to the NETEX procedure that is built when performing a BASE installation. This parameter is normally used when an existing customer is upgrading from a prior H214 release, and would like to incorporate their current PRODCONF file into the new NETEX procedure.

Example: PRODCONF(SYSP.PROD.NETEX.LICENSE(PRODCONF))

LICCODES Default value: hlq.SNXCTL(LICCODES)

Allowed values: Any valid fully qualified z/OS dataset name

The LICCODES parameter is used to provide the fully qualified name of an existing Secure NetEx/IP LICCODES license key file. This name will be added to the NETEX procedure that is built when performing a BASE installation. This parameter is normally used when an existing customer is upgrading from a prior H214 release, and would like to incorporate their current LICCODES file into the new NETEX procedure.

Example: LICCODES(SYSP.PROD.NETEX.LICENSE(LICCODES))

SNXSYSCL Default value: A

Allowed values: A-Z, 0-9

The SNXSYSCL parameter is used to provide the JES output class for SYSOUT datasets in the SNETEX procedure.

Example: SNXSYSCL(H)

SYSTCPD Default value: ‘’ (none)

Allowed values: Any valid fully qualified z/OS dataset name

The SYSTCPD parameter is used to provide the fully qualified name of an alternate TCP/IP stack for Secure NetEx/IP usage (OSA only). If this parameter is used, it is specified in the form of an alternate TCPDATA dataset (e.g. TCPIP.TCPIP.DATA). If this parameter is not specified, the default TCP/IP stack will be used by Secure NetEx/IP.

Example: SYSTCPD(TCPIPZ.TCPIP.DATA)

PROCLIB Default value: (None)

Allowed values: " or any valid fully qualified data set name

The PROCLIB parameter is used to provide the fully qualified name of the JCL procedure library data set that will contain the Secure NetEx/IP startup procedure, as specified by the

SNXPROC parameter. The EDIT phase creates the started task JCL procedure and the COPY phase copies it into your specified JCL procedure library for later activation. To bypass the copy of the JCL procedure, either do not execute the COPY phase of the installation job, or specify this parameter as:

PROCLIB('')

Example: PROCLIB(SYSP.NETEX.PROCLIB)

REPLACE Default value: NO

Allowed values: YES | NO

The REPLACE parameter is used to determine whether or not existing members in the dataset specified by PROCLIB will be replaced during the COPY phase.

Note: If the Secure NetEx/IP proc name specified by SNXPROC already exists in the PROCLIB dataset specified by PROCLIB, be sure to specify REPLACE(YES).

Example: REPLACE(YES)

DISP Default value: OLD

Allowed values: OLD | SHR

The DISP parameter is used to determine whether the dataset specified by PROCLIB will be allocated for exclusive use during the COPY phase. To ensure being able to successfully copy into the dataset specified by PROCLIB, specify this parameter as:

DISP(SHR)

Step 6. Check for required updates.

Refer to the H214 Memo-to-Users for instructions on checking for product updates at www.netex.com.

Step 7. Obtain the H214 software key.

Obtain the appropriate H214 software key from Secure NetEx/IP support (www.netex.com), by providing the output of the 'D M=CPU' command (see "Obtaining the Software Key" on page 101). When the key is received, update the LICCODES file with the new key.

The PRODCONF file consists of a record that contains a LICPATH keyword that identifies the location of the LICCODES file. The LICCODES file contains a record that identifies the actual software key provided by Network Executive Software, Inc. Absence of the correct software key will prevent H214 from executing properly.

A sample PRODCONF file is illustrated in Figure 22. A sample LICCODES file is illustrated in Figure 23.

The LICCODES file may contain multiple keys. If H214 is installed on multiple LPARs on the same system, and if the PRODCONF and LICCODES files are shared across these multiple LPARs, then keys for all of these instances of H214 can be placed in the same LICCODES file. Each instance of H214 will use the first key found that contains its fingerprint.

If there are multiple keys for the same LPAR, H214 will use the first key found for the LPAR on which H214 is being started, as it sequentially reads the keys from the file. This makes it important to add new H214 keys for an existing LPAR to the front of the file. For example, if a new key is installed that provides a license date extension, or adds a new feature to H214, adding this new key to the file before the old key ensures the new key will be used rather than the old key.

The LICCODES file may also contain keys in the old format (i.e. pre-7.1 keys), which means the same LICCODES file can be shared for both Release 7.3 as well as with prior releases. If NetEx eFT213 product is installed, it can also share the same PRODCONF and LICCODES files with H214.

The LICCODES files may also contain comments, as shown in Figure 23.

A PRODCONF DD statement must be included in the H214 startup JCL. If the DDN format is used for the LICPATH specification, then a LICCODES DD statement must also be included in the H214 startup JCL, and it must specify the name of the LICCODES file. If the DSN format is used for the LICPATH specification, then the dsname (or dsname/membername) specified by LICPATH identifies the LICCODES file, and a LICCODES DD statement is NOT required in the H214 startup JCL. However, in both cases, the LICCODES file must exist, and be able to be allocated and checked during the execution of H214.

Sample JCL to load the PRODCONF and LICCODES files is contained in the "hlq.SNXCTL" file, as members LOADLICP (to load PRODCONF) or LOADLIC (to load LICCODES).

```
LICPATH //DDN:LICCODES
-- or --
LICPATH  //DSN:dsname
-- or –
LICPATH //DSN:dsname(membername)
```

Figure 22. Sample PRODCONF records

```
*
* H214 Software license key for CPU/LPAR
*
*   Comments (preceded by either an '*' or a '#')
*   may be placed in the LICCODES file.
*
* add software license key obtained from NESi after this statement
*
*   NTX214IP
B6YL-6AEA-BAMF-UAH7-AH7Q-CAPU-GAHS-NYX5-5DZE-ZB26
```

Figure 23. Sample LICCODES record

Step 8. Review the H214 initialization parameters.

A sample configuration file is located in hlq.SNXCTL(SNXMAPCF). Refer to Appendix H for a description of the available statements.

After the installation, member SNXMAPCF should be in the data set described by the SNXMAPCF DD statement in the NetEx start-up JCL. Edit member SNXMAPCF to your site specifications.

Step 9. Authorize the SNXALOAD load library.

Use your site procedures to ensure the hlq.SNXALOAD library is authorized. For example, this can be accomplished by adding it to either the 'IEAAPFxx' or 'PROGxx' PARMLIB members, depending on the customer method in use for authorizing libraries. If you are using the 'PROGxx' method, and you have the dynamic APF list format specified, then the SNXALOAD library can be authorized dynamically without requiring an IPL. If you are using the 'IEAAPFxx' method, or the static APF list format is specified in the 'PROGxx' member, then an IPL is required to authorize the SNXALOAD library.

Refer to the IBM publication "*z/OS Initialization and Tuning Guide"* for complete descriptions of the methods used to authorize libraries.

Step 10. Define SNETEX Service

SNETEX must be defined as a TCP service in TCPIP.ETC.SERVICES

```
SNETEX          6951/tcp        snetex  # secure netex/ip
```

Step 11. (Optional) Update Policy Agent

If z/OS Policy Agent is being used, determine if you require any site required policy changes in order to authorize NetEx/IP access over the IP network.

Step 12. (Optional) System Performance Consideration

Consider updating the SCHED member in your parmlib to run the snxmap program as noswappable.

PPT PGMNAME(SNXMAP) NOSWAP

Consider your Workload Manager configuration to insure the SNXMAP task will run at a dispatch level that is appropriate for your system. This is a server application and should run at a server level priority. If you experience NRBSTATs of 3980 while communicating with the SNXMAP program, an internal TCP request did not complete in 10 seconds. This is usually a configuration issue or the system is very busy and could not service the request.

Step 13. Review Installed JCL

Review the SNXMAP started task JCL, along with other JCL in '&hlq.SNXCTL', and make any necessary changes to satisfy your site requirements.

Step 14. Start SNXMAP

Start SNXMAP by issuing the 'S SNXMAP' command.

Since SNXMAP functions as a long running task, Network Executive Software advises running SNXMAP as a started task as opposed to a batch job. If this is not possible, then you should give the job enough time on the JOB card so that a time-out condition does not occur.

Start SNXMAP by entering the following command:

```
S SNXMAP
```

Note: The test programs in the “hlq.SNXCTL” data set, may not be compatible with other distributions. Only run these as directed by Support@netex.com.

Step 15. (Optional) Execute the SNXMVEAT Program

The SNXMVEAT program “offers” its services. This program works in tandem with SMVGEN.

Execute the SNXMVEAT program. Edit member SNXMVEAT, shown in Figure 24 on page 113, in the “hlq.SNXCTL” data set, and change the parameters to your specifications. The “offer name” for both the SNXMVEAT and SNXMVGEN jobs must be the same. Save the changes and submit the job to z/OS for execution. This program remains running and must be cancelled when you complete processing.

```
//SNXMVEAT JOB ,NTXMVSET,CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//*
//*        THE FOLLOWING JCL WILL EXECUTE THE SECURE NETEX/IP SNXMVEAT PGM
//*
//*        STDIN : NUMSESS SECURE VALIDATE OFFERNAME HOSTNAME
//*
//SNXMVEAT EXEC PGM=SNXMVEAT,REGION=4096K
//*
//STEPLIB  DD  DISP=SHR,DSN=BETATST.H2140100.SNXLOAD
//*
//*MEATDBG  DD  DUMMY
//*SNXALL  DD  DUMMY
//*SNXGTRC DD  DUMMY
//STDOUT   DD  SYSOUT=*
//STDERR   DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//SYSTERM  DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//*
//*  (OPTIONAL) SYSTCPD POINTS TO ALTERNATE TCP/IP STACK
//*
//*SYSTCPD DD  DISP=SHR,
//*SYSTCPD+1   DSN=TCPIP.TCPIP.DATA
//STDIN    DD  *
001 Y Y SNXMVEAT ZOSA
//*
//
```

Figure 24. Sample SNXMVEAT Job, Member ‘SNXMVEAT in hlq.SNXCTL

Step 16. (Optional) Execute the SNXMVGEN Program

The SNXMVGEN “connects” to SNXMVEAT and sends a variable number (and size) of data blocks to the SNXMVEAT program.

Execute the SNXMVEAT program. Edit member SNXMVGEN, shown in Figure 25, in the "hlq.SNXCTL" data set, and change the parameters to your specifications. Specify the value of the host name on which the SNXMVEAT program is executing in the PARM field. The "offer name" for both the SNXMVEAT and SNXMVGEN jobs must be the same. The values specified in the PARM field of the SNXMVGEN are shown in Figure 25.. Save the changes and submit the job to z/OS for execution. This program ends when the request is complete.

```
//SNXMVGEN JOB ,SNXMVGEN,CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//*
//*        THE FOLLOWING JCL WILL EXECUTE THE SECURE NETEX/IP SNXMVGEN PGM.
//*
//* STDIN : NUMSESS NUMBLKS SIZE ODATA LOOPS DMODE SECURE VALIDATE HOST OFFERNM
//*
//SNXMVGEN EXEC PGM=SNXMVGEN,REGION=4096K
//STEPLIB  DD  DISP=SHR,DSN=BETATST.H2140100.SNXLOAD
//*
//*        STDIN SUPPLIES THE NEEDED INPUT
//*        PARAMETERS TO THE SNXMVSGN UTILITY PROGRAM AND IS
//*        DESCRIBED AS - AAA BBBBB C DDDD S V EEEEEEEE {FFFFFFFF}
//*
//*        AAA      - IS THE NUMBER OF BLOCKS TO TRANSMIT.
//*        BBBBB    - IS THE SIZE (IN BYTES) OF EACH BLOCK,
//*                   THE MAXIMUM IS 65535.
//*        C        - IS THE NUMBER OF LOOPS.
//*        DDDD     - IS THE AUTO DATAMODE CHARACTER SET.
//*        S        - SECURE (Y or N).
//*        V        - VALIDATE (Y or N).
//*        EEEEEEEE - IS THE HOST NAME.
//*        FFFFFFFF - IS THE OFFER NAME.
//*
//*MGENDBG  DD  DUMMY
//*SNXALL  DD  DUMMY
//*SNXGTRC DD  DUMMY
//STDOUT   DD  SYSOUT=*
//STDERR   DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//SYSTERM  DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//*
//*  (OPTIONAL) SYSTCPD POINTS TO ALTERNATE TCP/IP STACK
//*
//*SYSTCPD DD  DISP=SHR,
//*SYSTCPD+1   DSN=TCPIP.TCPIP.DATA
//STDIN    DD  *
001 01000 16000 000 001 000 Y N ZOSA SNXMVEAT
//*
//                                                                      //
```

Figure 25. Sample SNXMVGEN Job, Member 'SNXMVGEN in hlq.SNXCTL

Debugging User Applications

Users may wish to see Secure NetEx/IP information when attempting to debug application problems. The following can be used in the application job as DD DUMMY cards.

SNXERR	display errors seen by Secure NetEx/IP
SNXINF	display informational messages seen by Secure NetEx/IP
SNXDBG	debug messages seen by Secure NetEx/IP
SNXTRC	display the NRBS on input and completion
SNXALL	display all messages except SNXGTRC and SNXLETRC
SNXGTRC	trace the gsk traffic (file gsk.<process>.ff.trace will be put in the z/OS UNIX /tmp directory)
SNXLETRC	trace the LEC calls

Appendix D: H304 for Unisys Dorado/OS2200 Installation

Prerequisites

The following are hardware and software prerequisites for installing the H304 product.

- A Unisys Dorado compatible system running a supported OS2200 release. Review the website for supported OS distributions.
- At least one other processor on the network running Secure NetEx/IP software. This processor should be connected with another Secure NetEx/IP (not required for intra-host test/evaluation).
- Customers must obtain a software KEY from Support@netex.com prior to running the H304 software. Customers must contact Support@netex.com with the customer site name, system-info (see Post Installations Step 1), and the Secure NetEx/IP product designator (e.g., H304). Support@netex.com will supply the necessary key once this information has been received. The customer needs to place this key into the NESikeys file as discussed later in this section.
- H304 can coexist in the same OS2200 partition as H300e and/or H300IPC. They will each need a separate license key, as they are different products.
- Certificates and Key: If your certificates will NOT be signed by a third party, the self-signed certificates will need to be installed on the systems using secure transfer. For a system using CPCOMM, the certificate will need to be copied into the CPCOMM configuration file pointed to by the "TRUSTED_CERTIFICATE_FILE" parameter of the SSLTLS-security statement. For CPCOMM/OS, all certificates installed are trusted.

  For CPCOMM the sample SSL/TLS statement could look like:

```
SSL/TLS-SECURITY,SECCPFTP ;
  RSA-PRIVATE-KEY-FILE,DON*KEY.TESTPR ;
  RSA-CERTIFICATE-FILE,DON*CERT.CERTSIGNED ;
  TRUSTED-CERTIFICATES-FILE,DON*CERT.CERTSIGNED ;
  CIPHER-SUITE-MINIMUM,RSA_WITH_RC4_128_MD5
```

  For CPCOMM/OS the sample SSL/TLS statement could look like:

```
SSL/TLS-SECURITY,SECCPFTP ;
      RSA-PRIVATE-KEY-FILE,UNID4150key.pem ;
      RSA-CERTIFICATE-FILE,UNID4150cert.pem ;
.     These above two files reside in the SAIL partition
      CIPHER-SUITE-MINIMUM,RSA_WITH_NULL_MD5
```

  Care should be used when specifying the "CIPHER-SUITE-MINIMUM". Setting it to high could prevent system from negotiating a connection. Setting it to low, allows for simpler algorithms to be used, allowing for easier cracking of the security. The OS2200 operating system will negotiate to the highest possible level of security with partner system.

  Configure the COMAPI interface into CPCOMM;

  The COMAPI used by the secure Netex should specify a local IP V4 address to use to communicate with other components on the same host. By default, this should be 127.0.0.1. This address must match the

IPv4addr in the TCPCFG element in the load libray. The COMAPI parameter in this element must match the COMAPI mode you wish to use.

The PROCESS statement for COMAPI **must include** the following three statements:

SSL/TLS-CLIENT-AUTH,WEAK;
SSL/TLS-SECURITY,SECCPFTP ; Points to the correct SSL/TLS statement
INPUT-QUEUE-THRESHOLDS,50,1000,1000000

IT MUST NOT INCLUDE:

SSL/TLS-SECURE-PORTS,nnnn

Note: The COMAPI interface can support application imposed security or system imposed security. It cannot support both features in the same application. Multiple copies of COMAPI can be configured if both features are required. Secure Transfer allows a connection to the COMAPI interface running in a mode other than "A". (See the COMAPI = configuration parameter).

All requirements for the equipment listed above must be met before proceeding with the installation.

Hardware Installation

Install and verify proper operation of the appropriate operating system.

Accessing the H304 Software Distribution

The H304 Secure NetEx/IP software is available as a CFMT file. This file should be FTPed to your OS2200 system using the BIN, and QUOTE SITE CFMT options in ftp. Contact Support@netex.com to request the download link.

Upgrading H304

This is a new product. You must follow the "Software Installation" instructions.

Removing H304

You will then need to delete any files used by H304.

Software Installation

1. No software installation is required for H304. The release file is distributed as an executable program library. The only programs that can be executed from this library are the SNXMAP, SNXMAPOP, and the SYSINFO programs. You may run these programs from the release library, or copy the entire file to a new program library. This file is used to link with user applications like BFX that utilize the secure NETEX interface.

Post Installation Considerations

Configuring H304

Once the software package installation has been successfully completed, Secure NetEx/IP must be configured prior to execution. Sample elements can all be found in your EXEC file.

SNXMAP/UNIECL	contains sample ECL to start the NetEx portmapper application.
SNXMAPOP/UNIECL	contains sample ECL to start the console interface
SNXMAP/CFG	contains the NetEx initialization parameters.

The SNXMAP/UNIECL, and PTMPOP/SAMP should be copied to your SYSLIB*RUN$. These should be renamed to reflect the run names you wish to use. The RELEASE file should not be updated. Update the copy in your SYSLIB*RUN$ file.

SNXMAP/UNIECL:

Correct the run card to reflect your run name, accounting codes and userid requirements
Change the LOG file ECL to reflect the destination of your portmapper output.
Change the Dump data set to reflect the file name for the pads diagnostic file.
Change the PROD$CONF to point to the keys file (Step 1).
Change the SNXMAPCFG statement to point to your Secure NetEx/IP configuration data (Step2).

SNXMAPOP/UNIECL :

Correct the run card to reflect your run name, accounting codes and userid requirements
Change the PRINT file ECL to reflect the destination of your console output.

1. Create the 'NESikeys' file if necessary

A single NetEx License Key file must reside on each system where one or more NetEx products containing license support will be installed. The following guidelines apply:

- The SNXMAP/UNIECL use statement (@use PROD$CONF) should point to the product configuration file to use. This file should contain two members. The PRODCONF member should contain a single LICPATH statement naming this library. The second element is KEY. This element contains the product key issued by Support@netex.com. Multiple keys can exist in this file; the product will find the valid key. Sample elements are in the release file and maybe copied in to use a template in creating this file.
- In the KEY element, a leading '#', '*', '"', '/', or '!' character denotes a comment line.
- The systems programmer installing this software must edit this file (in quarter-word format (@ed,uq)) to add a new encrypted Software Key each time such a key is obtained from Support@netex.com for H304 and/or other license-enabled NetEx products. This should be done prior to installing the product, and must be done prior to any attempt to run the product successfully.
- To obtain a key, contact Support@netex.com, supplying the fingerprint of the machine the software is to be installed on. This may be obtained by using the following ECL:

  @USE RELEASE.,NETEX*RELEASEFILE. (This was FTP'ed to your system.)

  @XQT RELEASE.SYSINFO

(*This is the same information used in the H300IPC product*.)

- The KEY element may contain multiple keys per product due to new product releases or a change to the platform's fingerprint. If there are multiple keys the product will use the first key found that matches the product name and system fingerprint starting at the top of the file. This makes it important to add new keys for an existing system to the top of the file. For example, if a new key is installed that provides a license date extension, or adds a new feature, adding this new key to the file before the old key ensures the new key will be used rather than the old key. To make the file easier to maintain over time, it is recommended that you precede each Key entry with a comment line that documents the product designator (e.g., H304) and release level of the product that the key is associated with. It will then be easier to delete older Keys that are no longer valid for the product.

- Example of creating the product configuration file.
  - @cat,p netex*prodconf.,///50
  - @copy,s release.prodconf to the file cataloged.
  - @copy,s release.key to the file cataloged
  - The prodconf member, contains the name of the file to use (netex*prodconf). It some other name is used, update the element prodconf to point to the correct file.
  - To update the KEY to your prodconf file

    @ed,uq netex*prodconf.KEY (This file must be in quarter word format)

    Insert a line and paste in the key supplied to you. It must start in column 1.

    Comments may be added.
  - The following shows an example of what a *NESikeys* file might look like after adding a key to the file:

```
# Network Executive Software, Inc. Software License Key file
# Key for H304 R1.0:
CGGZ-4AAA-AAAE-IAO5-O5OJ-SBHX-AUZ5-PL4D
```

2. Create the SNXMAP configuration file

See Appendix G: Secure NetEx/IP Configuration on page 135.

This file is pointed to by the snxmap/uniecl use statement SNXMAPCFG use statement. This contains the configuration parameters for the SNXMAP program. This file contains the Secure NetEx/IP hostname for this host. Secure NetEx/IP hostnames may duplicate the Netex hostname used by the non-secure versions of the product. A sample member (SNXMAP/CFG) is in the release file.

EXAMPLE:

@cat,p netex*snxmap-cfg.,///1000

@copy,i release. SNXMAP/CFG,netex*snxmap-cfg.

@ed,uq netex*snxmap-cfg.

Insert a new line containing the Secure NetEx/IP host name for this system

LCLHOST NETXNME /* NETXNME is the Secure NetEx/IP Host name */

3. Edit the TCPCFG file

Users will need to configure the application interface to allow programs like BFX to communicate with TCP. This is located in your load library, or the release file.

- COMAPI
  - This is the mode value for the COMAPI you wish to use.
- INTV4ADDR
  - This is an internal IP V4 ip address assigned to the configured COMAPI Secure NetEx/IP will be using.
  - This is normally 127.0.0.1. Other 127 addresses could be used for different COMAPIs.
- PTMPORT
  - This is the port number Secure NetEx/IP will communicate with the SNXMAP application. 3919 is the default. This must be a site wide agreed upon port.

4. Create Secure NetEx/IP addressing information

There are two methods of creating/updating the IP information on your system or network to allow for Secure NetEx/IP to operate properly.

1. **Update DNS nameserver information.**

   This method requires that you update the relevant DNS lookup tables with the IP addresses and Secure NetEx IP hostnames. If you want to isolate the Secure NetEx hosts to specific IP Addresses on that host, the IP hostnames **must** be in the following format (case is important):

   NTX***IPHostname***

   Where ***IPHostname is the name of the host to be used***.

2. **Update local host file (normally only for the local host)**

   See CPCOM CONFIGURATION AND OPERATIONS GUIDE

5. Start SNXMAP

Start the SNXMAP run proc. This is a server application. Set the performance level to insure the correct dispatching priority. If you experience NRBSTATs of 3980 while communicating with the SNXMAP program, an internal TCP request did not complete in 10 seconds. This is usually a configuration issue or the system is very busy and could not service the request.

6. Start SNXMAPOP

snxmapop can run from a telnet terminal window on your OS2200 system. In this case all input and output is directed to the terminal window only.

```
@xqt H304*release000.snxmapop
```

This can be used as a simple configuration validity test. Pressing enter when prompt for input, the SNXMAPOP program will communicate with the SNXMAP program and display and Offers that are currently outstanding.

Linking User Applications

Users may link other Secure NetEx/IP applications with the Secure NetEx/IP interface. In the release file is a member named linkappl. The use statement for USERFILE should point to your EXTENDED MODE object file. The example is set up to link two applications SNXMEAT and SNXMGEN with the Secure NetEx/IP interface. These applications are already linked with this version of Secure NetEx/IP. SI points at the release file. The member TCPCFG should reside in your load library. This symbolic member specifies which comapi mode to use and the internal IPv4 address and port number to use when Secure NetEx/IP communicates with the local SNXMAP program.

Debugging User Applications

Users may wish to see Secure NetEx/IP information when attempting to debug application problems. In your run stream you may add any of the following statements:

@USE SNXERR,tpf$. display errors seen by Secure NetEx/IP

@USE SNXINF,tpf$. display informational messages seen by Secure NetEx/IP

@USE SNXDBG.,tpf$. debug messages seen by Secure NetEx/IP

@USE SNXTRC.,tpf$ .display the NRBS on input and completion

@USE SNXALL.,tpf$ .display all messages

@USE COMAPIDBG.,tpf$ .display comapi logging information

Appendix E: H804 Linux Installation

Prerequisites

The following are hardware and software prerequisites for installing the H804 product.

- An Intel compatible system running a supported Linux OS. Review the website for supported OS distributions.
- At least one other processor on the network running Secure NetEx/IP software. This processor should be connected with another Secure NetEx/IP (not required for intra-host test/evaluation).
- Customers must obtain a software KEY from Support@netex.com prior to running the H804 software. Customers must contact Support@netex.com with the customer site name, hostname, and the Secure NetEx/IP product designator (e.g., H804). Support@netex.com will supply the necessary key once this information has been received. The customer needs to place this key into the NESikeys file as discussed later in this section.
- Certificate and Key: Contact your Security Administrator to obtain PEM formatted keys and certificates. By default, these files are expected in the following files and their location can be specified in the snxmap.cfg file:

  /usr/share/nesi/snetex/key.pem

  usr/share/nesi/snetex/cert.pem

  If you would like to generate a self-signed certificate for testing, you can utilize OpenSSL tools to do this. Any self-signed certificate will need to be added to the trusted CA store for Secure NetEx/IP in order for the secure connection to work.

  One example of this is the following command:

```
openssl req -x509 -nodes -days 365 -newkey rsa:1024 -keyout
/usr/share/nesi/snetex/key.pem -out /usr/share/nesi/snetex/cert.pem
```

All requirements for the equipment listed above must be met before proceeding with the installation.

Hardware Installation

Install and verify proper operation of the appropriate operating system.

Accessing the H804 software distribution

The H804 Secure NetEx/IP software is available as an RPM which may be downloaded from NetEx. Contact Support@netex.com to request the download link.

Getting the NetEx Public Key

The RPM software distribution package is signed to ensure integrity and authenticity. It is recommended to install the NetEx public key and verify the signature of any software packages before installation.

You can download the NetEx public key from the H804 document page on the netex.com/support website.

Importing the NetEx Public Key

Install the public key as super user with the command:

```
# rpm –-import RPM-GPG-KEY-netex.txt
```

Verifying Signatures

You can verify the RPM signature to ensure that a package has not been modified since it has been signed. Verification will also check that a package is signed by the vendors or packagers key.

To verify the signature, use the -K or –checksig option to the rpm command:

```
# rpm –K H804-1.0.i386.rpm
```

Software Installation

If this is an initial installation, install the software as super user with the command:

```
# rpm –i H804-1.0.i386.rpm
```

If the NetEx public key has not been installed use the command:

```
# rpm –i –-nosignature H804-1.0.i386.rpm
```

Upgrading H804

If you are upgrading an existing installation of H804, it is strongly recommended that any running Secure NetEx/IP process be stopped. *If you are upgrading from a release which is older than the most recent previous release you should save the configuration files, remove or uninstall, and perform the instructions for an initial install.* Using the "rpm -U" command preserves any customized files in this package and the replacement files are installed with extensions of ".rpmnew". Any files that are not in the package but in package directories will also be preserved. Upgrade the software as super user with the command:

```
# rpm –U H804-1.0.i386.rpm
```

If the NESi public key has not been installed use the command:

```
# rpm –U –-nosignature H804-1.0.i386.rpm
```

Removing H804

If H805 is installed it should be removed prior to the removal of H804.

During RPM removal, any customized files and log files (e.g., **snxmap.cfg**, **prodconf, NESikeys,** etc.**)** will not be deleted. Remove the software as super user with the command:

```
# rpm –e H804
```

Removing the NESi Public Key

To remove the NESi public key, as super user issue the command:

```
# rpm –e gpg-pubkey-3d6b35d3-51bb5907
```

Starting, Stopping & Verifying Install of Secure NetEx/IP

The service command should be used to stop & start Secure NetEx/IP:

```
# service snetex stop
# service snetex start
# service snetex restart
```

The chkconfig command should be used to verify installation:

```
# chkconfig  --list  snetex
```

Post Installation Considerations

Configuring H804

Once the software package installation has been successfully completed, Secure NetEx/IP must be configured prior to execution. The following instructions address editing the files associated with product activation (**/usr/share/nesi/snetex/prodconf** and **/usr/share/nesi/NESikeys**), the configuration file for Secure NetEx/IP (**/usr/share/nesi/snetex/snxmap.cfg**), and starting the Secure NetEx/IP process.

Configuring the H804IP Secure NetEx/IP software consists of the following steps:

Step 1 Edit the 'NESikeys' file

Step 2 Edit the snxmap.cfg file

Step 3 Create Secure NetEx/IP addressing information

Step 4 Start SNXMAP

Step 5 Verify that 'snxmap' Starts Automatically On Reboot

Step 1. Edit the 'NESikeys' file

A single NESi License Key file must reside on each system where one or more NESi products containing license support will be installed. An example of this file can be found at /usr/share/nesi/snetex. The following guidelines apply:

- The default file name is *NESikeys*.
- The LICPATH keyword/value pair in the product configuration file (see **/usr/share/nesi/snetex/prodconf**) specifies the full path name to this file. The default is: ***/usr/share/nesi/NESikeys***.
- A leading '#', '*', or ';' character, in a file record denotes a comment line.
- The systems programmer installing this software must edit this file to add a new encrypted Software Key each time such a key is obtained from NESi for H804 and/or other license-enabled NESi products. This should be done prior to installing the product, and must be done prior to any attempt to run the product successfully.
- To obtain a key from NESi, contact NESi support, supplying the hostname of the machine the software is to be installed on. The hostname may be obtained by issuing the Linux command "*hostname*" with no parameters.
- The file may contain multiple keys per product due to new product releases or a change to the platform's fingerprint (on UNIX this corresponds to the hostname for the target host). If there are multiple keys the

NESi product will use the first key found that matches the product name and system fingerprint starting at the top of the file. This makes it important to add new keys for an existing system to the top of the file. For example, if a new key is installed that provides a license date extension, or adds a new feature, adding this new key to the file before the old key ensures the new key will be used rather than the old key. If there are multiple keys the NESi product will use the first key found that matches the product name and system fingerprint starting at the top of the file. This makes it important to add new keys for an existing system to the top of the file. For example, if a new key is installed that provides a license date extension, or adds a new feature, adding

- this new key to the file before the old key ensures the new key will be used rather than the old key. To make the file easier to maintain over time, it is recommended that you precede each Key entry with a comment line that documents the product designator (e.g., H804) and release level of the product that the key is associated with. It will then be easier to delete older Keys that are no longer valid for the product.
- The following shows an example of what a *NESikeys* file might look like after adding several Keys to the file:

```
# Network Executive Software, Inc. Software License Key file
# Key for H804 R1.0:
CGGZ-4AAA-AAAE-IAO5-O5OJ-SBHX-AUZ5-PL4D
```

Step 2. Edit the snxmap.cfg file

See Appendix G: Secure NetEx/IP Configuration on page 135.

Step 3. Create Secure NetEx/IP addressing information

There are two methods of creating/updating the IP information on your system or network to allow for Secure NetEx/IP to operate properly.

3. **Update DNS nameserver information.**

   This method requires that you update the relevant DNS lookup tables with the IP addresses and Secure NetEx IP hostnames. If you want to isolate the Secure NetEx/IP hosts to specific IP Addresses on that host, the IP hostnames **must** be in the following format (case is important):

   NTX***IPHostname***

   Where ***IPHostname is the name of the host to be used***.

4. **Update local host table.(normally only for the local host)**

   This method requires that you update the local host entry in the local hosts table (usually located at '/etc/hosts') with the IP address and IP hostname (can use same format as in #1).

Step 4. Start SNXMAP

On an initial install SNXMAP will not start automatically nor will it start following an update. To Start SNXMAP use the following command:

```
service snetex start
```

Step 5. Verify that 'snxmap' Starts Automatically On Reboot

SNXMAP has been configured to automatically start for run levels 2, 3, and 5 after a system reboot. It will not work until the 'snxmap.cfg' file has been properly set up and placed in the correct location, and the key is correct and in the correct location.

Note: If SNXMAP is started/stopped manually, the following script should be used as it properly modifies some system parameters required by Secure NetEx/IP and detects common problems:

```
service  snetex  start | stop | restart | status
```

Debugging User Applications

Users may wish to see Secure NetEx/IP information when attempting to debug application problems. To enable messages set any of the following variables in the application runtime environment. No value is necessary, the existence of a variable is sufficient (ex. export SNXTRC=).

SNXERR	Display errors seen by Secure NetEx/IP.
SNXINF	Display informational messages seen by Secure NetEx/IP.
SNXDBG	Display debug messages seen by Secure NetEx/IP.
SNXTRC	Display the NRBs on input and completion.
SNXALL	Display all messages (includes all of the above).
SNXSSLTRC	Display SSL state messages (must be set independently).

Appendix F: H624 AIX Installation

Prerequisites

The following are hardware and software prerequisites for installing the H624 product.

- An IBM Power System or System p server running, AIX® 5.3 to 7.1 distributions.
- At least one other processor on the network running Secure NetEx/IP software. This processor should be connected with another Secure NetEx/IP (not required for intra-host test/evaluation).
- Customers must obtain a software KEY from NESi prior to running the H624 software. Customers must contact NESi customer support with the customer site name, hostname, and the Secure NetEx/IP product designator (e.g., H624). NESi customer support will supply the necessary key once this information has been received. The customer needs to place this key into the NESikeys file as discussed later in this section.
- Certificate and Key: Contact your Security Administrator to obtain PEM formatted keys and certificates. These must be place in the following files:

  /usr/share/nesi/snetex/key.pem

  usr/share/nesi/snetex/cert.pem

  If you would like to generate a self-signed certificate for testing, you can utilize OpenSSL tools to do this. Any self-signed server certificate will need to be added to the trusted CA store for Secure NetEx/IP in order for the secure connection to work.

  One example of this is the following command:

  ```
  openssl req -x509 -nodes -days 365 -newkey rsa:1024 -keyout
  /usr/share/nesi/snetex/key.pem -out /usr/share/nesi/snetex/cert.pem
  ```

All requirements for the equipment listed above must be met before proceeding with the installation.

Hardware Installation

Install and verify proper operation of the AIX® system.

Accessing the H624 software distribution

The H624 Secure NetEx/IP software is available as an RPM which may be downloaded from NESi. Contact NESi Customer Support to request the download link.

Software Installation

All installation steps must be completed by a user logged on as root.

Version 1.0 of H624 Secure NetEx/IP installs from an RPM package.

Upgrading H624

If you are upgrading an RPM installation of H624, it is strongly recommended that any running Secure NetEx/IP process be stopped. Using the “rpm -U” command preserves any customized files in this package

and the replacement files are installed with extensions of ".rpmnew". Any files that are not in the package but in package directories will also be preserved. Upgrade the software as super user with the command:

```
# rpm –U H624-1.0.-1.ppc.rpm
```

Removing H625 RPM

If H625 BFX was installed, this RPM for BFX must be removed first.

If you wish to remove the rpm, you may use the following command using your superuser id.

```
# rpm –e H625
```

During RPM removal, any customized files and log files (e.g., snxmap.cfg, prodconf, NESikeys, etc.) will not be deleted.

Starting, Stopping & Verifying Install of SNXMAP

The startsrc command or SMIT should be used to start SNXMAP:

```
# startsrc -s snetex
```

The stopsrc command or SMIT should be used to stop SNXMAP:

```
# stopsrc -s snetex
```

The lssrc and lsitab commands or SMIT can be used to verify installation:

```
# lssrc -S -s snetex
# lsitab snetex
```

Post Installation Considerations

Configuring H624

Once the software package installation has been successfully completed, Secure NetEx/IP must be configured prior to execution. The following instructions address editing the files associated with product activation (**/usr/share/nesi/snetex/prodconf** and (**/usr/share/nesi/NESikeys**), the configuration file for Secure NetEx/IP (**/usr/share/nesi/snetex/snxmap.cfg**), and starting the Secure NetEx/IP process.

Configuring the H624 Secure NetEx/IP software consists of the following steps:

Step 1 Edit the 'NESikeys' file

Step 2 Edit the SNXMAP.CFG file

Step 3 Create Secure NetEx/IP addressing information

Step 4 Starting / Stopping

Step 5 Verify that 'snxmap' Starts Automatically On Reboot

Step 1. Edit the 'NESikeys' file

A single NESi License Key file must reside on each system where one or more NESi products containing license support will be installed. An example of this file can be found at /usr/share/nesi/snetex. The following guidelines apply:

- The default file name is **/usr/share/nesi/*NESikeys***.
- The LICPATH keyword/value pair in the product configuration file (see **/usr/share/nesi/snetex/prodconf**) specifies the full path name to this file. The default is: ***/usr/share/nesi/NESikeys***
- A leading '#', '*', or ';' character, in a file record denotes a comment line.
- The actual key must reside on a single line by itself and start in column 1.
- The systems programmer installing this software must edit this file to add a new encrypted Software Key each time such a key is obtained from NESi for H624 and/or other license-enabled NESi products. This should be done prior to installing the product, and must be done prior to any attempt to run the product successfully.
- To obtain a key from NESi, contact NESi support, supplying the hostname of the machine the software is to be installed on. The hostname may be obtained by issuing the Linux/AIX command "*hostname*" with no parameters.
- The file may contain multiple keys per product due to new product releases or a change to the platform's fingerprint (on UNIX this corresponds to the hostname for the target host). If there are multiple keys the NESi product will use the first key found that matches the product name and system fingerprint starting at the top of the file. This makes it important to add new keys for an existing system to the top of the file. For example, if a new key is installed that provides a license date extension, or adds a new feature, adding this new key to the file before the old key ensures the new key will be used rather than the old key. To make the file easier to maintain over time, it is recommended that you precede each Key entry with a comment line that documents the product designator (e.g., H624) and release level of the product that the key is associated with. It will then be easier to delete older Keys that are no longer valid for the product.

- The following shows an example of what a *NESikeys* file might look like after adding a key to the file. The first two lines are comments with the key on the third line. Additional comments and keys could be added.

```
# Network Executive Software, Inc. Software License Key file
# Key for H624 R1.0:
CGGZ-4AAA-AAAE-IAO5-O5OJ-SBHX-AUZ5-PL4D
```

Step 2. Edit the SNXMAP.CFG file

See Appendix G: Secure NetEx/IP Configuration on page 135.

Step 3. Create Secure NetEx/IP addressing information

There are two methods of creating/updating the IP information on your system or network to allow for Secure NetEx/IP to operate properly.

5. **Update DNS nameserver information.**

   This method requires that you update the relevant DNS lookup tables with the IP addresses and Secure NetEx IP hostnames. If you want to isolate the Secure NetEx/IP hosts to specific IP Addresses on that host, the IP hostnames **must** be in the following format (case is important):

   NTX***IPHostname***

   Where ***IPHostname is the name of the host to be used***.

6. **Update local host table.(normally only for the local host)**

   This method requires that you update the local host entry in the local hosts table (usually located at '/etc/hosts') with the IP address and IP hostname (can use same format as in #1).

Step 4. Starting / Stopping SNXMAP

On an initial install SNXMAP will not start automatically nor will it start following an update. To Start SNXMAP you may use the SMIT tool on AIX. SNXMAP is defined as an AIX subsystem, and may be started or stopped through the SMIT interface. It may also be manually started or stopped with:

```
startsrc –s snetex
stopsrc –s snetex
```

Step 5. Verify that 'snxmap' Starts Automatically On Reboot

SNXMAP has been configured to automatically start for run level 2 after a system reboot. It will not work until the 'snxmap.cfg' file has been properly set up and placed in the correct location, and the key is correct and in the correct location. To permanently disable this, the super user may issue the command "rmitab snetex".

Debugging User Applications

Users may wish to see Secure NetEx/IP information when attempting to debug application problems. To enable messages set any of the following variables in the application runtime environment. No value is necessary, the existence of a variable is sufficient (ex. export SNXTRC=).

SNXERR	Display errors seen by Secure NetEx/IP.
SNXINF	Display informational messages seen by Secure NetEx/IP.
SNXDBG	Display debug messages seen by Secure NetEx/IP.
SNXTRC	Display the NRBs on input and completion.
SNXALL	Display all messages (includes all of the above).
SNXSSLTRC	Display SSL state messages (must be set independently).

Appendix G: Secure NetEx/IP Configuration File

Edit the snxmap.cfg file

The *snxmap.cfg* file contains default values for Secure NetEx/IP parameters.

Note for UNIX only: After an update, if a snxmap.cfg.rpmnew exists there may be updated defaults that should be reviewed.

Edit this file with the following recommendations:

1. The LCLHOST defines the name of your local host. (This is analogous to the NetEx hostname in the NCT of legacy NetEx products.)

2. Edit or modify any other parameters for your host and site. The following table lists all of the parameters and their default values.

3. Certificates and CAs must be configured for secure transfers to operate. Secure NetEx/IP implements client and server certificate verification. Server certificate will be verified by the client and the client certificate will be verified by the server. Certificates and/or CAs will need to be configured on all hosts utilizing a secure connection.

Lines proceeded by an '*', '#', '"', '!', '.'or '/' are comments. In the distributed file, these comments indicate the use of provided program defaults. To override these default values you should duplicate the entry and uncomment, remove the *, and supply your override value. All keywords and Boolean values are case-insensitive. Boolean values: on, off, yes, no, true, false, 1, 0. The parameters are defined in four categories:

1. SNXMAP Logging Keywords
2. Common keywords for all Secure NetEx/IP applications
3. IBM GSK keywords
4. OpenSSL keywords

SNXMAP logging keywords	**Default**	**Definition**
DEBUG	Off	Boolean: enable/disable extra debug logging for SNXMAP
LOG	On	Boolean: enable logging to stdout/stderr
LOGFILE	Default	string: redirect stdout/stderr to named file (may be ignored on some platforms)
SYSLOG	Off	Boolean: enable syslog (where available)
SYSLOGFAC	local3	string: syslog facility name (where available)

Common keywords for all Secure NetEx/IP applications	Default	Definition
CNVERIFY	Off	Boolean: SSL certificate Common Name verification UNIX: Client will verify the remote server hostname matches SubjectName or SubjectAltName in server certificate OR will verify the remote IP address matches the SubjectAltName in the server certificate. Unisys: Ignore (N/A) zOS: Client will verify the remote server hostname matches the SubjectName in the server certificate.
DEFBI	32768	integer: size of default input block (0-65535)
DEFBO	32768	integer: size of default output block (0-65535)
DNSRR	Off	Boolean: DNS round-robin (randomly select IP)
IDLETO	6	integer: Utilizes TCP keepalive to implement idle time in seconds (0-disable) Valid range 1-7200 Unisys: Actual time is rounded to the up to the nearest 60 seconds.
LCLHOST	LCLHOST	string: local host name (max 8 characters)
MAXBI	32768	integer: maximum size of input block (2048-size of integer-1)
MAXBO	32768	integer: maximum size of output block (2048-size of integer-1)
PORTNUM	none	integer: number of ports used for offers (1-65535)
PORTSTART	none	integer: starting port used for offers (1-65535)
SMWAIT	15	integer: wait time between offer lookup and connect

IBM GSK keywords	Default	Definition
KEYFN	snx.kdb	string: certificate database file
KEYLBC	ClientLabel	string: client certificate label
KEYLBS	ServerLabel	string: server certificate label
KEYPW	kdbpw	string: certificate database password

OpenSSL keywords	Default	Definition
CAFILE	none	string: trusted CA certificate file containing PEM formatted certificates Used to verify certificates. Server will send client certificate request with CA names from this file.
CAPATH	none	string: trusted CA certificate directory containing PEM formatted certificate files using hash names Used to verify certificates.
CERTFILE	/usr/share/nesi/snetex/snx.crt	string: certificate file in PEM format
KEYFILE	/usr/share/nesi/snetex/snx.key	string: certificate private key file in PEM format

Notes:

- Some of the parameters documented above may not be included in the sample snxmap.cfg file (“***/usr/share/nesi/snetex/snxmap.cfg***”). It is the responsibility of the user to enter these values as necessary into the installation-specific copy of “snxmap.cfg” prior to starting Secure NetEx/IP.
- For Unix OS systems, after a re-install or upgrade install, it is possible to have a newer snxmap.cfg file. If so, it will be named snxmap.cfg.rpmnew. It is the responsibility of the user to merge or update snxmap.cfg to reflect any additions or deletions.

Appendix H: Secure NetEx/IP Tools

This section documents the Secure NetEx/IP tools shipped with the product.

If you have any questions on running these tools please contact support@netex.com

SNXMVGEN

This tool will generate data for testing purposes. It will prompt the user for parameters.

The prompt will look like this:

```
SNXMVGEN  V 3.0   12/10/12   65535 Max data
ENTER:
#SESS #BLOCKS SIZE  ODATA LOOPS DMODE VALIDATE HOSTNAME OFFRNAME
NNN   NNNNN   NNNNN NNN   NNNN  HHHH  Y/N      HHHHHHHH OOOOOOOO
```

Sessions: The number of concurrent sessions to process. This must be equal to or less than the number of sessions used by SNXMVEAT. No default.

Blocks: The number of blocks of data to generate. No default.

Size: The size of the blocks of data to generate in bytes. No default

OData: The number of bytes of ODATA to generate. Typically, this parameter can be set to 0.

Loops: The number of times to send all of the blocks. No default.

Dmode: The DATAMODE to use when sending the blocks. No default. (See NRBDMODE)

Sample data modes of 0202 would be for an ASCII to ASCII character transfer.

0203 would be for an ASCII to EBCDIC character transfer.

0302 would be for an EBCDIC to ASCII character transfer.

0 would be for a binary to binary transfer. The systems must have the same number of bits per byte for this test to work.

Validate: Should the content of each received block be validated. No default.

Hostname: The Secure NetEx/IP hostname to send to. No default.

OffrName: The SNXMVEAT Offer to connect to send the data. No default.

SNXMVEAT

This tool will read data generated by SNXMVGEN. It will prompt the user for parameters.

The prompt will look like this:

```
SNXMVEAT   V3.1   04/18/14   65535 max data
Enter:
#Sessions Validate OffrName HostName
NNN       Y/N      OOOOOOOO HHHHHHHH
```

Sessions: The number of concurrent sessions to process. This must be equal to or greater than the number of sessions used by SNXMVGEN.

Validate: Should the content of each block be validated. No default.

OffName: The OffrName SNXMVEAT will connect to for the test. No default.

Hostname: The Secure NetEx/IP hostname to receive from. No default.

Running SNXMVEAT and SNXMVGEN:

1. On the receiving side, execute SNXMVEAT (this MUST be started before SNXMVGEN).

   When you start the SNXMVEAT application, you will be prompted to specify the number of concurrent sessions and whether you want the application to validate those sessions. Enter the values separated by a space character, then hit ENTER.

   You will need to use CTRL-C (or let the offer(s) time out) to stop the SNXMVEAT application when your testing is completed.

   Example:

   ```
   shell_prompt# SNXMVEAT

   SNXMVEAT   V3.1   04/18/14   65535 max data
   Enter:
   #Sessions Validate OffrName HostName
   NNN       Y/N      OOOOOOOO HHHHHHHH
   1 n SNXMVEAT sunrise

   Making 1 offers of SNXMVEAT , validate 0, hostname SUNRISE
   ```

   OS2200 Example

   ```
   @xqt H304*release000.SNXMVEAT
   ```

2. On the sending host, execute SNXMVGEN.

   The SNXMVGEN application will prompt you to enter a suite of values to use during the test. Enter the values separated by a space character, then hit ENTER.

   For this example, we specified one (1) session of 99995 blocks of 32000 bytes with zero (0) ODATA, one (1) loop and specify zero (0) for the DMODE. There is no validation required. The following is an example of an execution of SNXMVGEN (user input is *italicized*).

   ```
   shell_prompt# SNXMVGEN
   SNXMVGEN  V 3.0   12/10/12   65535 Max data
   ENTER:
   #SESS #BLOCKS SIZE  ODATA LOOPS DMODE VALIDATE HOSTNAME OFFRNAME
   NNN   NNNNN  NNNNN NNN   NNNN  HHHH  Y/N      HHHHHHHH OOOOOOOO
   1 99995 32000 0 1 0 n sunrise SNXMVEAT
   ```

   OS2200 Example

   ```
   @xqt H304*release000.SNXMVEAT
   ```

Once both processes are up and running, on the SNXMVGEN side, after specifying the desired parameters and hitting the <Enter> key, you will see:

```
1 ses, 99995 blocks, 32000 bytes/blk, 0 odata bytes,
 1 loops, datamode 0, validate 0, to SNXMVEAT  at SUNRISE
Connect: Status: 0,Ind: 0, Session: 1 Try: 1
```

On the SNXMVEAT side you should see output similar information to:

```
COffer: Status: 0,Ind: 1, Session: 1 Try: 0
```

When each loop completes, the SNXMVGEN side will output the stats for the finished loop:

```
Session 1:
325.5054 Mbits/s,  40.6882 Mbytes/s, 1333.3199 OPs/s, 75 Sec, 99999 Blks, 3199872256 Bytes
```

On the SNXMVEAT side, the output when the test completes is similar to:

```
CDisc: Status: 0,Ind: 0, Session: 1
Session 1:
325.5052 Mbits/s,  40.6881 Mbytes/s, 1333.3199 OPs/s, 75 Sec, 99999 Blks, 3199872256 Bytes
```

Appendix I: Unisys SSL TRACING

In the event of SSL connection problems, please gather the following information:

JOBLOG (If you are connecting to a remote system)

COMAPI PRINT FILE

- Before the job is submitted, enter using your comapi keyin:
  - <*keyin*> log high (Turns on logging)
  - <*keyin*> log close (Starts a new print cycle)
  - Run the job
  - <*keyin*> log close (Close the print cycle This is the file with the data)
  - <*keyin*> log off (Turns off logging)

CPCOM TRACE FILE

- Before the job is submitted, enter using your cpcomm keyin:
  - <*keyin*> trace api-ssl,medium (Trace these records)
  - <*keyin*> trace api-tcp,medium (Trace these records)
  - <*keyin*> trace network,medium (Trace these records)
  - <*keyin*> trace ssl,medium (Trace these records)
  - <*keyin*> trace ip,medium (Trace these records)
  - <*keyin*> trace tcp,medium (Trace these records)
  - <*keyin*> trace close (Starts a new trace cycle)
  - Run the job
  - <*keyin*> trace close (Close the trace cycle This is the file with the data)
  - <*keyin*> trace off (Turns off tracing)
- Print the trace file.
  - Execute SYSLIB*CPCOMM.LTA (To print the trace)
  - The trace file name was displayed at close time
    - I (analyze interactively)
    - !HEX (print data in HEX)
    - !STATUS (Do a STATUS)
    - !ALL (Print all trace Records)
    - !QUIT (End)
    - Send in the printed trace file. The name is displayed

Index