IRIS SNA LU 6.2 Programming Guide

Document Number 007-0874-020

Contributors

Engineering contributions by Jay Lan

IRIS SNA LU 6.2 Programming Guide
Document Number 007-0874-020

Silicon Graphics, Inc.
Mountain View, California

Contents

Figures and Tables

Introduction

This guide is designed for application programmers and end users who operate Silicon Graphics® IRIS® SNA LU 6.2, a specific implementation of the Systems Network Architecture (SNA) Logical Unit (LU) Type 6.2 protocols.

Using This Guide

The *IRIS SNA 6.2 Programming Guide* contains the following chapters and appendices:

Chapter 1	"Programming with the API Interface" contains an overview to the Application Programmers Interface, or verb functions, which are provided separately as manual pages in Appendix C. A diagnostic guide for applications is also included.
Chapter 2	"Sample Transaction Programs" presents two example programs: a transaction program to send a file and one to receive a file.
Chapter 3	"The IRIS Implementation of LU 6.2" describes the Silicon Graphics implementation of the SNA LU 6.2 architecture and the implemented LU 6.2 function sets.
Appendix A	"Major and Minor Return Codes" lists the major and minor error codes returned by the API verbs (that is, the SNA SERVER, LU 6.2, and LU 0-3 verbs).
Appendix B	"Verb Catalog" provides an index to all API verbs.

Appendix C “Man Pages” contains the category (3) manual pages related to the IRIS SNA LU 6.2.

Conventions

Within text, file names, parameters, commands, and command arguments are shown in *italics.*

Command syntax descriptions and examples appear in `typewriter font`.

User input and keyboard commands appear in **`bold typewriter font`**.

API verb names are shown in **bold** face.

Related Documentation

The following reference materials from Silicon Graphics and IBM® provide supplementary information on topics covered in this guide.

Silicon Graphics, Inc.

Token Ring Administration Guide

IRIS SNA SERVER Administration Guide

IRIS SNA SERVER Programming Guide

IRIS SNA SERVER VT100 Interface Guide

IRIS SNA 3270 Administrator's Guide

IRIS SNA 3770 Administrator's Guide

International Business Machines (IBM order numbers follow title)

Systems Network Architecture Concepts and Products (GC30-3072)

Systems Network Architecture Technical Overview (GC30-3073)

An Introduction to Advanced Program-to-Program Communication (GG24-1584)

Systems Network Architecture Transaction Programmer's Reference Manual for LU Type 6.2 (GC30-3084)

Systems Network Architecture Format and Protocol Reference Manual: Architecture Logic for LU Type 6.2 (SC30-3269)

Systems Network Architecture Network Product Formats (LY43-0081)

Synchronous Data Link Control Concepts (GA27-3093)

Systems Network Architecture Reference Summary (GA27-3136)

Token Ring Network Architecture Reference (SC30-3374)

Product Support

Silicon Graphics provides a comprehensive product support and maintenance program for IRIS products. For further information, contact the Technical Assistance Center.

x

Chapter 1

Programming with API

This chapter contains information about writing transaction programs using the verb library functions. It also has information to help you diagnose application errors. The material in this chapter provides an introduction and overview of the verbs. Appendix A lists the major and minor codes returned by the verb functions. Appendix B, "API Verb Catalog," lists all of the API verbs in alphabetical order, giving their full names and verb types.

The IRIS SNA SERVER incorporates verbs that are of interest to the programmer using IRIS SNA LU 6.2. The relevant server verbs are noted first in this chapter. The LU 6.2 Application Program Interface (API) overview and verb catalogs follow. A description of the LU 6.2 security features and a diagnostics guide for applications complete this chapter. See Appendix C, "Man Pages," for a more complete description of the LU 6.2 verbs.

1.1 SNA SERVER Verb Categories

This section describes the functions provided by the IRIS SNA SERVER that are of interest to the programmer using the LU 6.2 Application Program Interface (API) verbs. There are three general categories of IRIS SNA SERVER API verbs used with the IRIS SNA LU 6.2 API verbs:

- Implementation-specific verbs
- Configuration verbs
- Node operator verbs

Note: Details about the configuration and node operator verbs are contained in the *IRIS SNA SERVER Programming Guide*. Because the implementation-specific verbs listed below are used by all applications, man pages for them are included with this guide's man pages for convenience.

1.1.1 Implementation-specific Verbs

Although the implementation-specific verbs, also called the transaction program connection verbs, are not part of the IBM SNA, they are required for the IRIS SNA SERVER.

Before a program can issue any conversation, control-operator, or node-operator functions, it must establish a connection with the IRIS SNA SERVER. This is called attaching. Conversation verbs and control operator verbs attach by specifying the name of the configuration and the local LU that the transaction program wants to use.

The names specified in the attach request are called the context of the attach. Since a program can issue verbs to more than one configuration or LU, the program can issue multiple attach requests. Each of these attaches creates a new logical instance of the program. The **setctx** verb switches from one instance to the other before issuing verbs to the different LUs.

Because transaction program connection verbs establish or break the connection between the transaction program and the IRIS SNA SERVER, they are the first and last verbs the program issues. The verbs listed in Table 1-1 attach and detach the application to the IRIS SNA SERVER. All applications must use the **attach** and **detach** verbs. The **setctx** verb is optional.

Verb	Function
attach	Initiates communication between the local program and the IRIS SNA SERVER.
detach	Detaches the current context from the IRIS SNA SERVER.
rattach	Initiates communication between a remotely invoked transaction program and the IRIS SNA Scheduler.
setctx	Establishes the current context under which subsequent verbs are issued.

Table 1-1 Transaction Program Connection Verbs

1.1.2 Configuration Verbs

Configuration verbs define and display the resources of the node and logical unit. The following IRIS SNA SERVER verbs are useful to the programmer using the LU 6.2 API verbs and are divided into two groups: define verbs and display verbs. Table 1-2 lists the configuration define verbs.

Verb	Full Name
dfncp	Define Control Point
dfnline	Define Line
dfnllu	Define Local LU
dfnmode	Define Mode
dfnnode	Define Node
dfnrlu	Define Remote LU
dfnsta	Define Station
dfntp	Define Transaction Program

Table 1-2 Configuration Define Verbs

Table 1-3 lists the configuration display verbs.

Verb	Full Name
dspcp	Display Control Point
dspline	Display Line
dspllu	Display Local LU
dspmode	Display Mode
dspnode	Display Node
dsprlu	Display Remote LU
dspses	Display Session
dspsta	Display Station
dsptp	Display Transaction Program

Table 1-3 Configuration Display Verbs

1.1.3 Node Operator Verbs

The IRIS SNA SERVER node operator verbs (Table 1-4) control the links and activate, deactivate, and supervise the configured local resources of the server.

Verb	Full Name
actline	Activate Line
actlu	Activate Logical Unit
actpu	Activate Physical Unit
actsta	Activate Station
chgmsgq	Change Message Queue
dctline	Deactivate Line
dctlu	Deactivate Logical Unit
dctpu	Deactivate Physical Unit
dctsta	Deactivate Station
dspmsgq	Display Message Queue
rtvnmsg	Retrieve Node Message

Table 1-4 Node Operator Verbs

1.2 API Overview

Advanced Program-to-Program Communications (APPC) provides high-level access to data communications facilities. It defines a machine-independent programmatic interface that offers a standard solution to the problems of data communications. This high-level approach frees the programmer to spend time designing the features of distributed applications rather than laboring over the details of communications protocols.

An APPC application is a couplet of programs that exchange both data and control information using the LU 6.2 programmatic interface. The second program in the couplet is started at the remote site at the request of the first. IRIS SNA LU 6.2 enables the two programs to exchange information. This exchange, called a conversation, is transparent to low-level communications. The programs conduct the conversation by issuing verbs, which are high-level procedural calls that perform the tasks of starting the conversation (and the partner program at the remote site), sending and receiving data, controlling error handling, and stopping the exchange.

In the IRIS implementation, these verbs take the form of C-language function calls. The Application Program Interface (API) is a library containing verb functions and header files defining the structures and variables used by applications that call the verb functions. For more LU 6.2 protocol information, consult the IBM *Transaction Programmer's Reference Manual*, which is referred to in this guide as the *TPRM*.

1.3 IRIS SNA LU 6.2 Verb Categories

Verbs are divided into categories according to their function. These categories, in turn, are divided into groups. Two categories of LU 6.2 verbs exist: conversation verbs and control operator verbs. A description of the groups that comprise each category and their respective functions follows. See the man pages in Appendix C for individual verb details.

1.3.1 Conversation Verbs

Application programs use conversation verbs to exchange data. These verbs are divided into three groups:

- Mapped conversation verbs
- Basic conversation verbs
- Type-independent verbs

Mapped and basic verbs conduct peer-to-peer conversations. They differ in the amount of formatting the application program must do. Type-independent verbs are used on either mapped or basic conversations.

1.3.2 Control Operator Verbs

Control operator verbs define and control the resources of the logical unit. They are divided into two groups: change number of sessions (CNOS) verbs and session control verbs.

Both CNOS and session control verbs control the sessions between the local LU and the remote LU.

1.3.3 Verb Library

Verb functions and functions called by the verb are archived in */usr/lib/liblu62.a*. Programs that use verb functions are linked against this library. For most compilers, use the *-l* option. For example:

```
cc sample.c-llu62.a
```

See your compiler documentation for instructions on linking with IRIS libraries.

1.3.4 Header Files

The structures and variables used in IRIS SNA LU 6.2 verbs are defined in header files located in */usr/include/sna*. These header files are available:

basic.h Contains definitions of the verb parameter structures for the basic conversation verbs

mapped.h Contains definitions of the verb parameter structures for the mapped conversation verbs

cntrl.h Contains definitions of the verb parameter structures for the control operator verbs

global.h Contains definitions of the global variables used by all of the verb types

imp.h Contains definitions of the verb parameter structures for the implementation-specific verbs

mc_mcb.h Contains definition of the mapping control block structure

In addition to these header files, */usr/include/sna* has a number of files of type definitions and constant values used by the LU 6.2 verbs. These files need not be explicitly included but can be examined for their contents.

ddhvtyp.h Type definitions for the LU 6.2 verbs.

ddhverr.h Constants for major and minor return codes.

ddhvicn.h Constants used for verb parameter values.

ddhviex.h External variable declarations.

1.3.5 Data Type Definitions

The verb parameter structures use data types defined in the header file *ddhvtyp.h*. These data types are:

typedef unsigned char hex
(Used for strings or bytes where all bits are significant)

typedef unsigned short shex
(Used for values that must be two bytes long)

typedef unsigned long lhex
(Used for values that must be four bytes long)

1.3.6 Data Structures

Each member (referred to as both parameter and field) of the data structure is described as being Supplied, Returned, or Supplied / Returned.

- Supplied parameters are set by the application program.
- Returned parameters are set automatically by the successful operation of the verb.

- Supplied/Returned parameters are set by the application program when the verb is issued, but their value can change after the successful operation of the verb.

Initialization of every member of the structure is the responsibility of the application. Pointers not set to a specific address are set to null.

Note: For the character-string parameters in the Display and Get Attribute verbs, Returned and Supplied have a slightly different meaning. These verbs require the application program to allocate space for returned names and strings. If the pointer is nulled, the name is not returned. Thus, even though the value in a name field is returned by the verb, the pointer must still be supplied by the application program. If a name parameter is listed as Supplied/Returned in a display verb, a different name can be returned in the same space after the successful completion of the verb.

Supplied parameters are specified as Required, Conditional, or Optional.

- Required parameters must be set by the application program.
- Conditional parameters may have a value required, depending on the setting of another parameter.
- Optional parameters need not be set.

1.3.7 Global Variables

Information on the state of the conversation and feedback on the execution of verb calls is returned in three global variables. These variables are defined in the header file *global.h*. This file must be included in each program that uses APPC functions. Routines that query the values of these variables should refer to them as external variables.

Return Code: *snamaj* and *snamin*

After executing a verb function, return information is placed in global variables *snamaj* and *snamin*. The values carried here correspond to the verb return-code parameters defined in the *TPRM*. If the function completes normally (return code of OK), the function returns 0 and the major and minor codes also are 0.

A return code set in the major or minor fields returns the verb function -1. In general, the major code is sufficient for controlling program logic.

Table 1-5 describes the major codes used. See Appendix A for a complete list of the major and minor return codes.

Major Code	Description
S2_OK (0)	Function completed successfully.
S2_UNSUC (2)	Function completed unsuccessfully. (Set when a function, such as “Receive Immediate” or “Test,” completes normally but does not return data.)
S2_USAGE (1)	Function aborted, usage error. (The function was not performed because a parameter was in error or requested an unsupported function. The majority of the minor codes provide specific information on usage errors.)
S2_STATE (3)	Function aborted, state error. (The function was not performed because it is not allowed in the current conversation state.)
S2_ALCER (5)	Allocation error. (The program could not allocate a conversation for the reason specified in the minor code. The conversation is in deallocated state when an allocation errors occurs.)
S2_PGMER (7)	Program Error. (The partner program issued an error indication. A send state conversation is changed to a receive state.)
S2_DEALC (9)	Deallocation indication. (The conversation has been deallocated, normally or abnormally, for the reason specified in the minor code. The conversation is in deallocated state.)

Table 1-5 Major Return Codes

Major Code	Description
S2_COERR (10)	Control-operator function error. (A control-operator function has ended abnormally. Since the control operator verb may not have been using a conversation, the conversation state does not apply.)
S2_NPERR (11)	Operation function error. (An operation function was not accepted. Since operation verbs do not use conversations, the conversation state does not apply.)

Table 1-5 (continued) Major Return Codes

Conversation State: *snastat*

After the execution of any of the conversation verb functions, the conversation state is set in the variable *snastat*. Table 1-6 describes the values. The states are referred to by their constant identifier throughout the conversation verb documentation.

Constant Identifier	Description
S2_NONE (0)	No state. (The program has not allocated a conversation.)
S2_RESET (1)	Reset state. (The conversation has not been fully allocated.)

Table 1-6 Conversation State Constant Identifiers

Constant Identifier	Description
S2_SEND (2)	Send state. (The conversation can send data and confirmation requests.)
S2_RECV (3)	Receive state. (The conversation can receive information from the partner program.)
S2_CNFRM (4)	Confirm state. (The conversation can respond to a confirmation request from the partner program.)
S2_CSEND (5)	Confirm send state. (The partner has issued a prepare-to-receive of type SYNC. When the program issues a confirm, it is in send state.)
S2_CDELC (6)	Confirm deallocate state. (The partner has issued a deallocate of type SYNC or type CONFIRM. When the program issues a confirm, it is in deallocate state.)
S2_DLCED (7)	Deallocate state. (The partner has ended the conversation. The program must deallocate the conversation locally.)

Table 1-6 (continued) Conversation State Constant Identifiers

1.4 Conversation Verbs

Full connectivity among programs requires that all transaction programs interpret the records they transfer in the same way. Data is transferred between SNA components or sublayers via message units (MUs), which are any bit-strings that contain an SNA-defined format. The type of format depends on the type of conversation.

The basic conversation protocol boundary, usually used by service transaction programs, is implemented in a low-level language, such as assembler. Basic conversations format MUs into logical records, which consist of a two-byte length prefix (LL) followed by data. A transaction program sending data over a basic conversation must include the LL field in its data, and complete the formatting of the logical record it is sending before leaving the send state.

The mapped conversation protocol boundary, usually used by application transaction programs, is implemented in a high-level language, such as C. Mapped conversations transform MUs into a standard format called a general data stream, or GDS. The basic structural unit in the GDS is a two-byte length prefix (LL), a two-byte GDS identifier (ID), and a variable length data field. A transaction program sending data over a mapped conversation is responsible for providing only the data and a map name. The receiving LU automatically converts the data into its original form.

Figure 1-1 illustrates the significant difference between basic and mapped conversation formats.

Figure 1-1 Basic and Mapped Conversation Formats

Because the LL prefix of a logical record has the same format as the LL field in a GDS variable segment, a GDS variable segment is also a logical record. Application transaction programs that use basic conversations do not need to supply ID fields. If they are supplied, the basic conversation treats everything following the LL prefix of the logical record as user data.

Conversation verbs are divided into three categories: mapped, basic, and type independent. Mapped verbs communicate between user-written transaction

programs. Basic verbs are normally used only by service programs, such as the Change Number of Sessions program. Type-independent verbs provide functions that span both mapped and basic verbs.

Note: Since a transaction program that uses basic verbs is responsible for all data formatting, it can also use basic verbs to format mapped conversation flows. This feature is useful in implementations that provide only basic-verb support.

1.4.1 Mapped Conversation Verbs

All mapped conversation verb functions and functions called by a mapped conversation verb are in */usr/lib/liblu62.a*. Data structures for the mapped conversation verbs are in */usr/include/sna/mapped.h*. The global variables are defined in the header file */usr/include/sna/global.h*.

The mapped conversation verbs listed in Table 1-7 are supported.

Verb	Function
malcnv	Allocates a mapped conversation connecting the local transaction program to a remote transaction program.
mcnfrm	Sends a confirmation request to the remote transaction program and waits for a reply so that the two programs can synchronize their processing.
mcnfrmed	Sends a confirmation reply to the remote transaction program so that the two programs can synchronize their processing.
mdalcnv	Deallocates a mapped conversation resource from the transaction program.
mflush	Transmits all information that the LU has buffered.
mgetatr	Returns information pertaining to a mapped conversation.
mprprcv	Changes the mapped conversation from send state to receive state in preparation to receive data.
mpstrct	Requests posting of the specified mapped conversation when information is available for the program to receive.
mrcvim	Receives any information available from the specified mapped conversation, but does not wait for the information to arrive.
mrcvwt	Waits for information to arrive on the mapped conversation and then receives the information.

Table 1-7 Mapped Conversation Verbs

Verb	Function
mrqssnd	Notifies the remote program that the local program is requesting to enter send state for the mapped conversation.
msnddta	Sends one logical record to the remote transaction program.
msnderr	Informs the remote transaction program that the local program has detected an application error.
mtestcv	Tests the mapped conversation to determine whether or not it has been posted.

Table 1-7 (continued) Mapped Conversation Verbs

Conversation States

Certain verbs are only issued in certain states. An application can be designed to use the state variable to determine which verb to issue. For example, the program may be designed with a receive loop that issues **mrcvim** while in S2_RECV state and a switch statement that issues **msnddta**, **mdalcnv**, or **mcnfrm** when the state changes from S2_RECV. (See the sample transaction programs in Chapter 2 for examples of the use of the state variable.)

Table 1-8 lists the states in which each mapped verb are issued.The states listed in the table heading are described in Section 1.3.7, “Global Variables.”

	S2_ REQUEST	S2_ SEND	S2_ ECV	S2_ CONFIRM	S2_ CSEND	S2_ SDELC	S2_ DLCED
malcnv	yes	no	no	no	no	no	no
mcnfrm	no	yes	no	no	no	no	no
mcnfrmed	no	no	no	yes	yes	yes	no
mdalcnv							
Type:							
flush	no	yes	no	no	no	no	no
sync	no	yes	no	no	no	no	no
abend	no	yes	yes	yes	yes	yes	no
local	no	no	no	no	no	no	yes
confirm	no	yes	no	no	no	no	no
mflush	no	yes	no	no	no	no	no
mgetatr	no	yes	yes	yes	yes	yes	yes
mprprcv	no	yes	no	no	no	no	no
mpstrct	no	no	yes	no	no	no	no
mrcvwt	no	yes	yes	no	no	no	no
mrcvim	no	no	yes	no	no	no	no
mrqssnd	no	no	yes	yes	no	no	no
msnddta	no	yes	no	no	no	no	no
msnderr	no	yes	yes	yes	yes	yes	no
mtestcv							
Type:							
posted	no	no	yes	no	no	no	no
rqssnd	no	no	yes	yes	no	no	no

Table 1-8 Mapped Verb Conversation States and Verb Validity

Data Mapping

At its simplest level, the information sent between two transaction programs consists of a stream of data bytes formatted into logical records. At the logical-record level, however, the data may require further transformation before the

transaction program can receive it. Map names provide this function. A map name is a non-null user-defined name that identifies the format of the logical record and describes the mapping performed before the data is sent in a manner transparent to the transaction program.

The map name specified by the local program and the map name received by the remote program can be different. For example, the local LU can translate a map name with a meaning known locally into a global map name known to the remote LU. The remote LU in turn can translate the received map name into one that is known locally to the remote transaction program.

Data mapping is optional on a mapped conversation. That is, the logical record can be sent without being mapped. A null map name specifies that no data mapping should be done. A null map name is never translated into a non-null map name, although a non-null map name may be translated into a null map name, which would disable mapping.

Mapper: The Mapping Utility Interface

When data mapping is specified, the programmer is responsible for supplying it. A defined interface exists between the LU Mapped-conversation Component (MC) and the user-supplied Mapping Utility (Mapper). The Mapper must adhere to this interface to ensure correct operation of the LU mapped-conversation component.

Under the LU 6.2 architecture, the Mapper is responsible for map-name transmission as well as data transformation. Under this implementation of data mapping, the Mapper handles only data transformation. The MC handles the transmission of map names between the local and remote LUs.

The function that performs mapping, *mc_map*, is defined in a man page. To use data mapping, an application programmer creates a function with the name *mc_map* and links it to the executable object.

1.4.2 Type-independent Conversation Verbs

The type-independent conversation verbs described in Table 1-9 are used on either mapped or basic conversations and are supported in IRIS SNA LU 6.2.

Verb	Function
gtype	Returns the type of resource to which the specified resource ID is assigned.
waitcv	Waits for posting to occur on any basic or mapped conversation from among a list of conversations.

Table 1-9 Type-independent Conversation Verbs

Conversation States

After the execution of any of the conversation verb functions, the conversation state is set in the variable *snastat*.

Table 1-10 lists the states in which each type-independent verb can be issued. (The states listed in the table header are defined in Section 1.3.7, “Global Variables.”)

	S2_RESET	S2_SEND	S2_RECV	S2_CNFRM	S2_CSEND	S2_CDELC	S2_DLCED
gtype	no	yes	yes	yes	yes	yes	no
waitev	no	no	yes	no	no	no	no

Table 1-10 Conversation States and Verb Validity

All type-independent conversation verb functions, and functions called by a type-independent conversation verb are in */usr/liblu62.a*. The data structures for the type-independent verbs are in */usr/include/lu62/basic.h*.

The global variables are defined in the header file *global.h*.

1.4.3 Basic Conversation Verbs

Basic conversation verbs can conduct basic or mapped conversations. The application is responsible for correctly formatting the data that is sent using

basic verbs. That is, the data must be packed into logical records, including the GDS variables if a mapped conversation is being conducted. Basic verbs are intended for use by LU service programs, such as CNOS or Document Interchange Architecture (DIA). Mapped verbs also use basic verbs; the mapped verbs format the data and the verb request and then call the basic verb to perform the action.

The basic conversation verbs listed in Table 1-11 are supported.

Verb	Function
alcnv	Allocates a conversation connecting the local transaction program to a remote transaction program.
cnfrm	Sends a confirmation request to the remote transaction program and waits for a reply, so that the two programs can synchronize their processing.
cnfrmed	Sends a confirmation reply to the remote transaction program, so that the two programs can synchronize their processing.
dalcnv	Deallocates a conversation resource from the transaction program.
flush	Transmits all information that the LU has buffered.
getatr	Returns information pertaining to a conversation when information is available for the program to receive.
prprcv	Changes the conversation from send state to receive state in preparation to receive data.
pstrct	Requests posting of the specified conversation when information is available for the program to receive.
rcvim	Receives any information that is available from the specified conversation, but does not wait for the information to arrive.
rcvwt	Waits for information to arrive on the conversation and then receives the information.
rqssnd	Notifies the remote program that the local program is requesting to enter send state for the conversation.

Table 1-11 Basic Conversation Verbs

Verb	Function
snddta	Sends information to the remote transaction program.
snderr	Informs the remote transaction program that the local program has detected an application error.
testcv	Tests the conversation to determine whether it has been posted.

Table 1-11 (continued) Basic Conversation Verbs

Conversation States

Some verbs are issued only in certain states (see Table 1-12). An application can use the state variable to determine which verb to issue. One example is a program designed with a receive loop that issues **rcvim** while in S2_RECV state and a switch statement that issues **snddta**, **dalcnv**, or **cnfrm** when the state changes from S2_RECV.

	S2_ RESET	S2_ SEND	S2_ RECV	S2_ CONFIRM	S2_ CSEND	S2_ CDELC	S2_ DLCED
alcnv	yes	no	no	no	no	no	no
cnfrm	no	yes	no	no	no	no	no
cnfrmed	no	no	no	yes	yes	yes	no
dalcnv							
Type:							
flush	no	yes	no	no	no	no	no
sync	no	yes	no	no	no	no	no
abend	no	yes	yes	yes	yes	yes	no
local	no	no	no	no	no	no	yes
confirm	no	yes	no	no	no	no	no
flush	no	yes	yes	yes	yes	yes	yes
getatr	no	yes	yes	yes	yes	yes	yes
prprcv	no	yes	no	no	no	no	no
pstrct	no	no	yes	no	no	no	no
rcvwt	no	yes	yes	no	no	no	no
rcvim	no	no	yes	no	no	no	no
rqssnd	no	no	yes	yes	no	no	no
snddta	no	yes	no	no	no	no	no
snderr	no	yes	yes	yes	yes	yes	no
testcv							
Type:							
posted	no	no	yes	no	no	no	no
rqssnd	no	yes	yes	no	no	no	no

Table 1-12 Conversation States and Verb Validity

1.5 Control Operator Verbs

Control operator verbs, which define and control LUs, modes, and sessions, are divided into two categories: change number of session verbs and session control verbs.

CNOS verbs establish the number of sessions allowed between two LUs over a particular mode. Session control verbs activate and deactivate sessions after the session limits have been established.

The verb functions and functions called by the verb are archived in */usr/lib/liblu62.a*. Programs that use verb functions are linked with this library. The data structures for the CNOS and session control verbs are in */usr/lib/cntrl.h*. The global variables are defined in the header file *global.h*.

Note: Feedback on the execution of control operator verb calls is returned in two global variables: *snamaj* and *snamin*. The third global variable, *snastat*, is not used by the control operator verbs.

1.5.1 Change Number of Session Verbs

The CNOS verbs set the allowed number of sessions between the local LUs and the remote LUs. The limits are set for each mode defined between the LUs. When the node is first activated, the session limits on all modes is 0; that is, no sessions can be activated. Limits are raised by the **initsl** verb. If pre-bound sessions were defined for the mode, sessions can also be activated as a result of raising the limits. If not, sessions can be started using the session control verbs. Once limits have been raised initially, they can be changed by the **chgsl** verb. All session activity can be terminated by using the **rstsl** verb, which reduces the session limits to 0.

When parallel sessions are supported by the LUs (that is, the defined session limit can be greater than 1), the two LUs must agree on the number of sessions allowed. This agreement is negotiated between the LUs by an LU 6.2 conversation over a special mode. The mode, named SNASVCMG, is defined by default for all LUs that support parallel sessions. The session limit on this mode is always set to 2 so the LUs need not negotiate these limits. Nevertheless, these modes must be initialized by both LUs before any other mode can be initialized. The LU starts the negotiation when the operator issues **initsl** (initialize session limits), **chgsl** (change session limits), or

rstsl (reset session limits) for a parallel-session mode. The LU then allocates a conversation with the partner LU, requesting the CNOS model as its target program, and sends a defined message that contains the requested limits. The CNOS model (*s2_cnos*) issues the **procsl** verb to handle the target side of the negotiation. This verb determines the defined limits for the mode and returns a message containing limits that are the lesser of those defined for the mode or requested by the source. The target returns the limits which the mode will observe.

Note: When LUs try to initiate the CNOS exchange at the same time, the LU with the "greater" network name prevails.

When parallel sessions are not supported, the **initsl** verb is still issued, although no CNOS negotiation takes place. Do not use the **chgsl** verb (limits are either 0 or 1). See the man pages in Appendix C for detailed information on the verbs listed in Table 1-13.

Verb	Function
chgsl	Changes the session limit and contention-winner polarities for parallel-session connections.
dspsl	Provides information on the mode's current session limit.
initsl	Establishes the initial session limit for single-session or parallel-session connections.
procsl	Processes the session limit, contention-winner polarities, and related CNOS parameters from the source LU and, if necessary, negotiates them to values acceptable to the target LU.
rstsl	Resets to 0 the session limit for single-session or parallel-session connections, and the contention winner polarities for the parallel-session connections.

Table 1-13 CNOS Verbs

1.5.2 Session Control Verbs

Session control verbs explicitly activate and deactivate sessions. Sessions are activated by any one of three ways:

1. By issuing the **actses** verb.
2. By issuing a CNOS verb if pre-bound (also called “auto-activated”) sessions are defined for the mode.
3. By issuing the **alcnv** verb if session limits have been raised, but no sessions are active.

Sessions can be deactivated by issuing either a **dctses** verb or a CNOS verb to reduce the number of allowed sessions. In fact, if pre-bound sessions are defined, a session may be activated to replace one brought down by a **dctses** verb, leaving the same number of sessions active. The session control verbs listed in Table 1-14 are supported. See Appendix C, “Man Pages,” for additional information about these verbs.

Verb	Function
actses	Activates a session with the specified mode name to the target LU. The session is activated as a contention winner for either the source or target LU.
dctses	Deactivates the specified LU-LU session.

Table 1-14 Session Control Verbs

1.6 Security Features

Three levels of security are defined for IRIS SNA LU 6.2:

- LU-LU security
- Conversation-level security
- Resource-level security

LU-LU security at session activation verifies the identity of the remote LU.

Conversation-level security verifies access to the remote system (that is, it determines whether or not the requesting user is authorized to allocate a conversation to the remote system).

Resource-level security verifies the user's authority to access the requested resources on the remote system; for example, whether the user is authorized to access the requested transaction program.

This section explains the three types of security and how they are implemented Following the explanation is a comparison of this implementation to the specifications in the *TPRM*.

1.6.1 LU-LU Security

LU-LU security verifies the identity of the remote LU when a session is activated between local and remote LUs using passwords configured at both LUs. Both LUs verify the identify of the other by using the following exchange protocol during session activation.

The local LU transmits random text to the remote LU, which encrypts the text using its password. The remote LU sends the encrypted data back to the local LU. The local LU then encrypts the original random data with its password and checks that the two encrypted versions match. The same exchange takes place in the opposite direction with the remote LU transmitting random data to the local LU, which then sends back the encrypted form to the remote LU.

To initiate LU-LU security, define an LU password for the remote LU with the **dfnrlu** verb by specifying the *pswd* and *pswdop* parameters. The hex characters must match the password of your partner. A partner using LU 6.2 specifies the same characters in defining the remote LU for your site. If your partner is an IBM system, consult the relevant IBM manual for information on how to specify security information. The encryption algorithm makes only the first seven bits of each byte significant. Therefore, passwords 0x000000 and 0x01010101 are identical.

Note: Because of federal export regulations, the LU-LU security feature is not available on all systems.

1.6.2 Conversation-level Security

Conversation-level security verifies that a user requesting a program start on a remote site is authorized to that site. When allocating a conversation, the user of the transaction program specifies a user ID, password, and profile. If the user does not specify this information, the source LU captures the information from the sign-on ID under which the program is being run. The security information is transmitted with the conversation-initiation information to enable the remote site to validate the request for access.

The security information is specified on the **alcnv** verb parameters *sec, user, pass,* and *prof.*

The parameter *sec* controls how security information is specified:

- SEC_NONE indicates that no security information should be transmitted.
- SEC_SAME specifies the use of the sign-on ID of the user running the transaction program. In this case, the password is flagged as being already verified and is not transmitted.
- SEC_PGM indicates that the information provided in the *user, pass,* and *prof* parameters is to be transmitted to the remote site.

Note: Since data is not encrypted, any password sent on the allocate request is transmitted in readable form and captured with a line trace. For this reason, the SEC_SAME option is preferred.

The acceptance of the security information is configured with the **dfnrlu** verb at the site receiving the allocation request (remote site). The ***s****ecacc* parameter indicates the type of security information that can be accepted for each of the remote LUs.

The following values are defined:

- NO_SEC indicates that security information is not accepted. Allocation requests received from this LU that carry security information are rejected. This means that the initiation program can specify only SEC_NONE on the **alcnv** request.
- USERID indicates that security information is accepted from this LU, but the password-verified option is not allowed. Therefore, the initiating program cannot specify SEC_SAME on the **alcnv** request.

- VERIFIED indicates that security information is accepted and that the verified option can be used. This means the initiation program can use any valid setting for the *sec* parameter of the **alcnv** request.

After sessions have been established between the LUs, each reads the security-acceptance level of its partner from the *psecacc f*ield returned in the **dsprlu** verb. The three values specify only the acceptable level of security information, and not the level required.

Accepted security information is verified against the IRIX™ system file */etc/passwd* when the allocation request is received at the remote site. The user is authorized if listed in */etc/passwd*. The password is verified against the encrypted password in */etc/passwd* (through the IRIX login program). If the */etc/passwd* entry for the user lacks a password, the user is verified without checking the allocation request password. Also, if SEC_SAME was specified, indicating a verified allocation request password, no password check is performed. The transaction program initiator *s2_tpi* performs the check against the IRIX password file.

Note: The **dfnllu** and **dfntp** verbs define a list of authorized users for the LU and the transaction program. This defined security information is never used in this implementation. The IRIX security information is used in its place. Defining users has no adverse effect on operations, although it increases memory resource usage.

1.6.3 Resource-level Security

Resource-level security is defined for IRIS SNA LU 6.2 to add another level of access authorization above conversation-level security. Resources, such as transaction programs, can be restricted to a small group of users. Resource security is provided in two ways. First, when a transaction program is defined, specify the level of security information required to run the program. Second, when the program is started, it is initiated under the IRIX user ID specified in the allocation request, so that IRIX security can control access to other resources, such as data files.

The **dfntp** verb's *secrq* parameter specifies whether or not security is required. Several values are provided for this parameter, indicating whether conversation-level security is required and giving the level of resource security required. However, since this implementation verifies security against the

IRIX password file, not against the user information specified on the **dfntp** verb, the *secrq* parameter functions as though it were a Boolean value. Specifying SQ_NONE indicates that security information is not required. Security information can be specified in the allocation request, but is not required. Any other value indicates that security information must be provided in the allocation request. The user and password fields are verified against */etc/passwd*. If a profile is specified, it is verified against */etc/group*.

There is a connection between the *secacc* parameter of the **dfnrlu** verb and the *secrq* parameter of **dfntp**. The requirement for security information is determined by the transaction program configuration, but the acceptance of security information is determined by the remote LU configuration. Mismatched configurations can lead to situations where no allocation request is accepted. For example, if the transaction program requires security, but security information cannot be accepted from the remote LU, allocation requests for the transaction program from the remote LU are always rejected. Requests that contain security information are rejected because *secacc* is SEC_NONE. Requests that do not specify security are rejected because *secrq* is not SQ_NONE.

After an allocation request has passed all edits and the user has been verified, the transaction program initiator *s2_tpi* invokes the transaction program process and changes the process group ID and process user ID values to those in the */etc/passwd* entry for the user. The transaction program runs under the IRIX security limitations of the group ID and user ID and is, in effect, logged in as the requesting user. When no security information is specified, the transaction runs by default under the group ID and user ID of the *s2_tpi* process. Underlying IRIX security enhances the LU 6.2-defined user verification by limiting system resources available to the transaction program to those resources available to the specified user.

In addition to user and password, LU 6.2 implements a special use of the *profile* field. A profile specified in the allocation request identifies an IRIX group name. The IRIX system file */etc/group* is checked to see if the group is available to the user. If so, the transaction program runs under the group ID indicated in */etc/group* for the group name (specified by the *profile* parameter) as well as under the user ID specified for the user in */etc/passwd*. This way a given user has multiple groups available under which to run the transaction program. The group is specified by the *profile* parameter in the allocation request.

The *s2_tpi* process requires the effective user ID of a superuser to enable it to set the group ID and the user IDs for the transaction program processes. This is accomplished either of two ways:

1. Start *s2_tpi* from a superuser logon.

   This is dangerous since transaction programs run by default under the user ID of the *s2_tpi* process if no security information is specified in the allocation request. If no security information is specified, the evoked program runs as *root*.

2. Run the *s2_tpi* program under an effective user ID of a superuser.

   The *s2_tpi* program owned by a superuser has its file mode set to run under an effective user ID of the file's owner (for example, mode 04111) and runs under the effective user ID of the superuser. *s2_tpi* always removes the process-effective user ID from invoked transaction program processes. This is the preferred method.

1.6.4 Comparison of TPRM and LU 6.2 Security

This section compares the methods used by IRIS SNA LU 6.2 to provide LU to LU security, conversation-level security, and resource level security to those described in the IBM *System Network Architecture Transaction Programmer's Reference Manual for LU 6.2 (TPRM).*

LU-LU Security

The IBM *TPRM* configures LU-LU security via the *DEFINE_REMOTE_LU* verb. The *LU_LU_PASSWORD* parameter defines the 64-bit password used during session activation for the LU-LU verification.

LU 6.2 configures LU-LU security with the **dfnrlu** verb. The *pswd* and *pswdop* parameters define the LU-LU password.

Conversation-level Security

In the IBM *TPRM*, the *DEFINE_LOCAL_LU* verb provides a list of users and their associated passwords to the local LU. This list identifies the user specified in the allocation request; or, if the user is not already identified in the request, the list provides passwords to verify that the user has system access. In LU 6.2, the list of users is provided by the IRIX system file */etc/passwd*. User and password information can be specified by the **dfnllu** verb, but this information is not used.

In the *TPRM*, the *DEFINE_REMOTE_LU* verb defines the acceptable remote LU allocation request security information. In LU 6.2, security acceptance is defined by the **dfnrlu** verb in a fashion similar to *TPRM*.

Resource-level Security

In the *TPRM*, the *DEFINE_TP* verb indicates the security required by a specific transaction program. In LU 6.2, the security required is defined similarly by the **dfntp** verb. However, the various flavors of security have been collapsed into one in the LU 6.2 implementation.

In the *TPRM*, the *DEFINE_TP* verb defines the users, passwords, and profiles authorized to access the program. In LU 6.2, the authorities can be defined, but the list of authorized users is not checked at program-initiation time. Instead, IRIX security is contained in */etc/passwd* and */etc/group* files. This provides unauthorized access protection for resources, such as files, not directly under the control of LU 6.2.

1.7 Application Diagnostics Guide

Table 1-15 provides information to help locate the source of problems encountered when creating an application with IRIS SNA LU 6.2.

Problem	Solution
Compile Errors:	
Include files not found	Use the *-I* option of the *cc* command to direct the compiler to load in */usr/lib*.
Unresolved references	The program must be linked against the LU 6.2 library */usr/lib/liblu62.a*.
Execution Errors:	
LU name and / or mode are not recognized	Your program must know the configuration's names for the local LU, remote LU, and mode names to attach and to allocate a conversation.
Verb function returns a usage error (Major Code 01)	A usage error indicates that a verb parameter is being use incorrectly. Refer to the Message Guide in the *IRIS SNA SERVER Administration Guide* for an exact description of the error received and how to change the parameter settings.
Remote program not known (Major Code 05) (Minor Code 08)	If a fully qualified pathname is not specified, the program must be located in one of the directories in the standard path. Either relocate the target program or specify a pathname.
Remote program starts, but does not begin to receive data.	The remote program must issue an LU **rattach** verb connect itself to the conversation. It then must re-issue a **rcvwt** verb to begin to receive data. Consult the man pages on these verbs for more information.

Table 1-15 Application Error Codes

Problem	Solution
When communicating with an IBM system, PIP arrive with unexpected values.	Parameters sent to an IBM system must be sent in EBCDIC. Use the routines **atoe** and **etoa** to perform the translation. (CICS does not support Program Initialization Parameter (PIP).
When communicating with an IBM System/36, transaction error is received. (Major Code 07) (Minor Code 11)	The RPG support on the System/36 required mapped conversations. The data service must be packaged in GDS variable 0x12ff. See the sample transaction programs in Chapter 2 for an example of a mapped conversation.

Table 1-15 (continued) Application Error Codes

Chapter 2

Sample Transaction Programs

These sample transaction programs are for descriptive purposes only and are not considered part of the production system. Two sample programs are illustrated. The first describes a transaction program to send a file, and the second describes a program to receive a file.

2.1 Sample Program: Send a File

OBJECT: *s2_fsnd.c*

FULL NAME: Send a file

TYPE: End-user Package Main

FUNCTION: Sends a file to another computer, either UNIX® or IBM. The program will confirm delivery.

Without any option flags, the program uses a basic conversation, performs no transformation on the file, and evokes the remote program *s2frcv*. With option flags, a user can request the program to translate the file into EBCDIC, use a mapped conversation, or evoke a different remote program.

INPUTS: *Required:*

- Pathname of local node
- Local LU name
- Remote LU name

- Mode name
- Pathname of the file to send
- Name(s) of the file on the remote side

If the remote system requires multiple names to identify a file (for example, library name, data set name), enter them here separately. Each name is then passed to the remote program as a separate parameter.

INPUTS: *Optional:*

-m Use a mapped conversation

-e Translate the file to EBCDIC before sending

This option assumes text is being sent. New lines are stripped and text is shipped in packets of eighty characters or less. This option cannot be used when document mode is specified.

-r Evokes “Program name”

-d Document mode

All flags must come on the command line before the required names. The required names must be in the order given. All parameters are translated to EBCDIC before sending, regardless of whether the *-e* option is taken. It is the responsibility of the remote program to transform them if necessary.

OUTPUTS: An exit code of 0 if the remote system confirms successful delivery; otherwise, an exit code of -1.

```
/*..... ..................................................*/
/* DATA DEFINITION SECTION */
/*..... ..................................................*/

#include <stdio.h>
#include <fcntl.h>
#include "global.h"/ *SNA62 global variables*/
#include "basic.h"/ *SNA62 Basic Verb Header*/

long cnvid;*/ Conversation Identifier*/
#define BUFFSIZE 802
#define RECSIZE 84
#define REMOTPGM "s2_frcv"

/*..... ..................................................*/
/* MAINLINE                    */
/*..... ..................................................*/

main(argc, argv)
int   argc;
char**argv;
      {
      extern interrno;
      extern intoptind;
      extern intopterr;
      extern char*optarg;
      int rc;
      int fildes;
      FILE*stream;
      char*ch;
      int nbyte;
      int c;
      int start = 2;/* Default to basic*/
      int mapped = 0;/* Default to basic*/
      int ebcdic = 0;/* Default to ascii*/
      int doc = 0/* Default to not a document*/

      char**names;
      char*rpgm;

      structsnddta_dssnddta_ds;

      union{
         short II
         hex buff[BUFFSIZE];
         } s;

      union{
         short II
         hex buff[RECSIZE];
         } e;
```

```
/*..... ..................................................*/
/* Find Option flags -m, -e, -d, and -r */
/*..... ..................................................*/

opterr = 1;/* Turn off option error */
rpgm = REMOTPGM;

while(   (c=getopt(argv, "mer:d")   )!=EOF)

       {
       switch (c)

          {
          case 'm':
             mapped = 1;
             start = 4;
             break;

          case 'e':
             if (doc)
                 {
                 printf("-e for ebcdic assumes documentode./n");
                 printf("-e and -d cannot be specified
                        simultaneously.\n");
                 exit(-1);
                 }
             ebcdic = 1;

             break;
          case 'r':
             rpgm = optarg;
             if( !rpgm )
                 {
                 printf("A program name isrequired with the -r
                        option.\n");
                 printf("s2_fsnd exiting.\n");
                 exit(-1);
                 }
                 break;
          case 'd':
             if ( ebcdic )
                 {
                 printf("-e for ebcdic assumes documentode.\n");
                 printf("-e and -d cannot be specified
                        simultaneously.\n");
                 exit(-1);
                 }
             doc = 1;
             break;

             }
          }
```

```
      names = (argv + optind);


/*..... ..................................................*/
/* Six parameters must be entered.  Exit if not */
/*..... ..................................................*/
if ( (argc - optind) <6 )
      {
      printf("These parameters must be entered to
                  s2_fsnd:\n");
      printf("The configuration file name, the local LU,
                     the partner LU, the mode,
                     \n");
      printf("the file you wish to transfer, and its name
                     on the remote system.\n");
      printf("The -m -e and -r flags are optional, but
                     must come first if present.
                     \n");
      printf("s2_fsnd exiting.\n");
      exit(-1);
      }


/*..... ..................................................*/
/* Open a local file.  Exit if it cannot be opened */
/*..... ..................................................*/

   if ( (fildes = open( *(names + 4), O_RDONLY )  ) == -1 )
      {
      printf("File %s open unsuccessful, errno %d\n",
                        *(names + 4),errno);
      printf("s2_fsnd exiting.\n"); exit(-1);
      }

   else if ( doc || ebcdic)
      {
      if(   !(stream = fdopen(fildes, "r") )  )
         {
         printf("File %s open unsuccessful, errno %d\n",
            *(names + 4), errno);
         printf("s2_fsnd exiting.\n"); exit(-1);
         }
      }
```

```
/*..... ..................................................*/
   LOCALLY ATTACH THE PROGRAM

   LATTACH requires the node name, local LU name, and transaction
program name to be present as parameters.
/*..... ..................................................*/

   rc = lattach( *(names + 0), *(names + 1), argv[0] );

   if( rc == S2_ERR )
          {
          prterr(names);
          exit( -1 );
          }


/*..... ..................................................*/
   ALLOCATE THE CONVERSATION

   Pass mapped and ebcdic values, the remote pgm name, and the
pointer to the array of pointers that contain the names.
/*..... ..................................................*/


   rc = lconv( mapped, ebcdic, rpgm, names); /* Allocate
                                Conversation*/

   if (rc == S2_ERR )
          {
          prterr(names);
          exit( -1 );
          }
/*..... ..................................................*/
   READ AND SEND DATA

   If ebcdic, call sndibm - else call sndunix.
/*..... ..................................................*/

if ( ebcdic )
      {
      sndibm(snddta_ds, mapped, stream, start, names);
      }

else
      {
      sndunix(snddta_ds, mapped, stream, start, names,
                       doc, fildes, ebcdic);
      }
```

```
/*..... ..................................................*/
   DEALLOCATE, FLUSH

   If the deallocate returns OK, detach from the LU and end the
program. If the deallocate fails, (a negative response was
received), deallocate locally, detach, and end program with an
abend code.
/*..... ..................................................*/

if ( snamaj != S2_OK )
       {
       cleanup( );
       }

else
       {
       rc = ldealloc( DC_SYNC );

       if ( rc == S2_NOERR )
           {
           rc = detach( );
           rc = 0;
           }

       else
           {
           prterr(names);
           cleanup();
           }
       }

exit( rc );
       /* END SPEEDL  */


/*..... ..................................................*/
   LOCAL ATTACH
/*..... ..................................................*/

lattach( pfile, llu, tpn )
char *pfile, *llu, *tpn;

       {
       structattach_ds attach_ds;
       int rc;

       attach_ds.type= AT_LU
       attach_ds.path= pfile;
       attach_ds.name= llu;
       attach_ds.tpn= tpn;
       attach_ds.wait= NO;

       rc = attach(&attach_ds);
       return( rc );
       }
```

```
/*...... ..................................................*/
   LALLOC
/*...... ..................................................*/

lconv(mapped, ebcdic, rpgm, names)
int mapped;
int ebcdic;
char *rpgm;
char *names[ ];
       {
       int rc, i, j;
       char wrkrlu[8];
       char wrkmode[8];
       struct alcnv_ds alcnv_ds;

       alcnv_ds.rlu = names[2];
       alcnv_ds.mode= names[3];alcnv_ds.tpn = rpgm;

          alcnv_ds.when = AC_WHEN;/* Delay*/
          alcnv_ds.type = mapped;/* Basic or mapped   */
          alcnv_ds.sync = 1;

          alcnv_ds.user = NULL;
          alcnv_ds.pass = NULL;
          alcnv_ds.sec = 0;

          alcnv_ds.pipused = 1;
          if ( ebcdic )
             alcnv_ds.pipa[0] = "EBCDIC";
          else
             alcnv_ds.pipa(0) = "ASCII";

       atoe( alcnv_ds.pipa[0], strlen(alcnv_ds.pipa[0]) );

       strcpy(wrkrlu, alcnv_ds.rlu);

       alcnv_ds.pipa(1) = wrkrlu;

       atoe( alcnv_ds.pipa{1}, strlen(alcnv_ds.pipa[1]) );

       strcpy(wrkmode, alcnv_ds.mode);

       alcnv_ds.pipa(2) = wrkmode;

       atoe( alcnv_ds.pipa{2}, strlen(alcnv_ds.pipa[2]) );

       for( i = 5, j = 3; names[i]; i++, j++ )
          {
          alcnv_ds.pipa[j] = names[i];
          atoe( alcnv_ds.pipa[j],
                    strlen(alcnv_ds.pipa[j]) );
          }
```

```
       alcnv_ds.pipa[j] = NULL;
       rc = alcnv(&alcnv_ds);
       cnvid = alcnv_ds.cnvid;
       return( rc );
       }

/*..... ..................................................*/
   LDEALLOC
/*..... ..................................................*/

ldealloc(type)
int type;
       {
       int rc;

       if( snastat == S2_DLCED )
          type = DC_LOCAL;

       rc = dalcnv(cnvid,type,NULL);

       return( rc );
       }

/*..... ..................................................*/
   PRTERR
/*..... ..................................................*/

prterr(names)
char *names[];

          {
          if( !(snamaj == S2_DEALC && snamin == S2_DNORM ) )
             {
             printf("Major code: %-4d %s\n", snamaj,
                    dspmaj(snamaj));
             printf("Minor code: %-4d %s\n", snamin,
                    dspmin(snamaj,snamin));
             printf("Error in s2_fsnd, file '%s' not sent
                    to '%s'\n", names[4], names[2] );
             }
          }


/*..... ..................................................*/
   CLEANUP
   Called if a conversation terminates abnormally.
/*..... ..................................................*/

   cleanup()
       {
       ldealloc( DC_AB_PGM );
       detach( );
       exit(-1);
       }
```

```
/*..... ..................................................*/
   READ FROM THE FILE, SEND THE DATA, UNIX TO IBM

   Leave the first two bytes for the logical record length field.
   Read eighty bytes of data or less at one time.
/*..... ..................................................*/

sndibm(snddta_ds, mapped, stream, start, names)

       structsnddta_dssnddta_ds;
       int mapped;
       FILE*stream;
       int start;
       char**names;

       {

       int nbyte;
       int rc;
       union {
          short  II
          hex    buff[RECSIZE];
          } e;

snddta_ds.cnvid = cnvid;
snddta_ds.data = e.buff;

/* while snastat is in send state, and read is successful,
   is successful, perform the send data */

   while( snastat == S2_SEND &&
   (fgets(e.buff + start, RECSIZE - start, stream)) )
       {

       nbyte = strlen( e.buff +start );

       if( e.buff[nbyte + start -1] =='\n' )nbyte--;

       atoe( e.buff + start, nbyte );

       e.II = nbyte + start;

       snddta_ds.length = nbyte + start ;

       if( mapped )
          {
          *( e.buff + 2 ) = 0x12;
          *( e.buff + 3 ) = 0xFF;
          }

       rc = snddta( &snddta_ds );

       if( rc == S2_ERR )
          prterr(names);

       }
}
```

```
/*...... ..................................................*/
   READ FROM THE FILE, SEND THE DATA, UNIX TO UNIX

Leave the first two bytes for the logical record length field.
/*...... ..................................................*/

sndunix(snddta_ds, mapped, stream, start, names, doc, fildes,
                              ebcdic)

       structsnddta_dssnddta_ds;
       int mapped;
       FILE*stream;
       int start;
       char**names;
       int doc;
       int fildes;
       int ebcdic;

       {

       int nbyte;
       int rc;
       union {
           short  II
           hex    buff[BUFFSIZE];
           } s;

       snddta_ds.cnvid = cnvid;
       snddta_ds.data = s.buff;

       /* while snastat is in send state, and either of
          the read types are successful, perform the
          send data */

       while( snastat == S2_SEND && ( (!doc && (nbyte =
           read(fildes, s.buff + start,
           BUFFSIZE - start ) )
           && (nbyte != -1)) ||
           (doc&& (fgets(s.buff + start,
           BUFFSIZE - start,stream)) ) ) )

           {

           if( doc )   nbyte = strlen( s.buff + start );

           if( ebcdic )
           atoe( s.buff + start, nbyte );

           s.II = nbyte + start;
           snddta_ds.length = nbyte F+ start ;

           if( mapped )
```

```
        {

        *( s.buff + 2 ) = 0x12;
        *( s.buff + 3 ) = 0xFF;

        }
        rc = snddta( &snddta_ds );
        if( rec == S2_ERR )
        prterr(names);
        }

    }
```

2.2 Sample Program: Receive a File

OBJECT: *s2_frcv.c*

FULL NAME: Receive a file

TYPE: End-user Package Main

FUNCTION: Program reads the open conversation and writes to the file name input as a parameter. If the file is sent in EBCDIC, this program translates it to ASCII. This program can read from either a basic or mapped conversation: it simply checks to see what type of conversation has evoked it and acts accordingly.

INPUTS: Node name, conversation ID, EBCDIC or ASCII, (depending on which language the file is in), the RLU name, mode name, and pathname of the file. Node name and conversation ID are provided by the LU; a programmer wishing to write a different source program provides the EBCDIC or ASCII string and the pathname of the file. This program does not use RLU and mode, a part of the PIP data required by IBM System/36 programs.

OUTPUTS: A return code of 0 if the file is successfully written to disk; otherwise, -1.

```
..... .....
***/

#ifndef GOLD static char SCCSID[] = "@(#)s2_frcv.c 1.4";#endif


/*..... .....*/
/* DATA DEFINITION SECTION   */
/*..... .....*/

#include <fcntl.h>
#include <string.h>
#include "global.h"/* SNA62 global variables*/
#include "basic.h"/* SNA62 Basic Verb Header*/

#defineBUFFSIZE   802 #define
       RECSIZE     80

union {
       short II;
       hex buff[BUFFSIZE];
       } r;

long   cnvid; /* Conversation Identifier*/


/*..... .....*/
/* MAINLINE                    */
/*..... .....*/

main(argc, argv)
int   argc;
char  *argv[];
       {
       extern  interrno;
       int rc;
       structrcvwt_ds rcvwt_ds;
       int fildes;
       char*fspc;
       int start = 2;
       int mapped = 0;/* Default to basic*/
       int ebcdic = 0;/* Default to ascii*/



/*..... .....*/
/* REMOTELY ATTACH THE PROGRAM       */
/*..... .....*/

       rc = lrattach( argv[1], argv[2] );
       if( rc == S2_ERR )
          {
          cleanup();
          exit(-1);
          }
```

```
/*..... ..................................................*/
/* Determine the type of conversation, set LL offset  */
/*..... ..................................................*/

          mapped = gtype( cnvid );
          if( mapped == S2_ERR )
             {
             cleanup();
             exit(-1);
             }
          if( mapped )
             start = 4;
          else
             start = 2;



/*.........................................................*/
/*Check parameters, which are in EBCDIC.  Abend if    /* /*parms
not here.  Determine if EBCDIC to ASCII       /*
/*transform is needed.                                  /*
/*.....................................................*/

          if( !argv[3] || !argv[6] )
          if( mapped == S2_ERR )
             {
             cleanup();
             exit(-1);
             }

          etoa( argv[3], strlen(argv[3] ) );
          if( !(strcmp( argv[3], "EBCDIC" ) ))
             ebcdic = 1;
          etoa( argv[6], strlen( argv[6] ) );
             fspc = strchr(argv[6], ' ');
          if( *fspc != '\0';


/*..... ..................................................*/
/* Open local file.  Exit if it cannot be open.  */
/*..... ..................................................*/

          if ( (fildes = open( argv[6], O_WRONLY | O_CREAT,
             0777 )) == -1 )
                 {
                 cleanup();
                 exit(-1);
                 }



/*..... ..................................................*/
/*          Receive the data and write it out.  */
/*..... ..................................................*/

       rcvwt_ds.cnvid = cnvid;
       rcvwt_cs.fill = RW_LL;
```

```
        rcvwt_ds.data = r.buff;

        while( snastat == SW_RECV && rc !=S2_ERR )

                {
                rcvwt_ds.length = BUFFSIZE;

                rc = rcvwt( &rcvwt_ds );

                if (rc != S2_ERR && snastat == S2_RECV &&
                          rcvwt_ds.what == RW_CMPL )
                    {
                    if( ebcdic )
                    {
                     etoa( r.buff + start, rcvwt_ds.length - start);
rc = ctus(r.buff + start, rcvwt+ds.length - start, fildes);
                       }
                    else
                    {
                    rc = write( fildes. r.buff + start,
rcvwt_ds.length - start );
                    }

                if ( rc == -1)
                    (
                    cleanup();
                    exit(-1);
                    }
        }


/*..... ..................................................*/
   CONFIRM DEALLOCATE
If a confirm deallocate arrives, send back a positive
response.  A negative response would have been sent prior to this.
/*..... ..................................................*/

if( snastat == S2_CDELC )
        {
        cnfrmed( cnvid );
        }


/*..... ..................................................*/
   DEALLOCATE

   If the deallocate returns OK, detach from the LU and end the
program.  Ifthedeallocatefails,(anegativeresponsewasreceived),
deallocate locally, detach, and end program with an abend code.
/*..... ..................................................*/
```

```
if (snastat == S2_DLCED )
       {
       rc = ldealloc( DC_LOCAL );
       rc = detach();
       rc = 0;
       }

else
       {
       cleanup();
       rc = -1;
       }
       exit( rc );

       }  /* END FRCV   */
/*..... ..................................................*/
/* REMOTE ATTACH             */
/*..... ..................................................*/

lrattach(pfile,cconv)
char   *pfile;
char   *cconv;

       {
       int rc;

       cnvid = rattach(pfile, cconv, AT_NOMAX);

       return( cnvid );
       }
/*..... ..................................................*/
/*       LDEALLOC  */
/*..... ..................................................*/

ldealloc(type)

int    type;

       {
       int rc;

       if ( snastat == S2_DLCED )
          type = DC_LOCAL;

       rc = dalcnv(cnvid,type,NULL);

       return( rc );
       }



/*..... ..................................................*/
   CLEANUP
   Called if a conversation terminates abnormally.
/*..... ..................................................*/

cleanup()
```

```
      {
      ldealloc( DC_AB_PGM );
      detach();
      }


/*..... ..................................................*/
   CTUX
Converts EBCDIC source file to UNIX.  Adds new line termination
and strips spaces at ends of line.  Called if a conversation
terminates abnormally.
/*..... ..................................................*/

ctux(buf, noc, fd)
char*buf;    /* pointer to buffer */
int noc;     /* number of characters */
int fd;      /* file descriptor */
{
char *a, *b;

   a = buf;

   while( noc > 0 )
      {
      b = a + RECSIZE - 1;
      while( *b-- == ' ' )
         ;

      if( write(fd, a, (unsigned)(b - a + 2)) == -1 )
         return( -1 );

      if( write(fd, "\n", 1) == -1 )
         return( -1 );

      a += RECSIZE;
      noc -= RECSIZE;
      }
   return( 0 );
}


/*..... ..................................................*/
   CLEANUP
   Called if a conversation terminates abnormally.
/*..... ..................................................*/

cleanup()
      {
      ldealloc( DC_AB_PGM );
      detach();
      }
```

```
/*..... ................................................*/
   CTUX
Converts EBCDIC source file to UNIX.  Adds new line termination
and strips spaces at ends of line.  Called if a conversation
terminates abnormally.
/*..... ................................................*/

ctux(buf, noc, fd)
char*buf;     /* pointer to buffer */
int noc;      /* number of characters */
int fd;       /* file descriptor */
{
char *a, *b;

   a = buf;

   while( noc > 0 )
      {
      b = a + RECSIZE - 1;
      while( *b-- == ' ' )
         ;

      if( write(fd, a, (unsigned)(b - a + 2)) == -1 )
         return( -1 );

      if( write(fd, "\n", 1) == -1 )
         return( -1 );

      a += RECSIZE;
      noc -= RECSIZE;
      }
   return( 0 );
}
```

Chapter 3

The IRIS LU 6.2 Implementation

This chapter provides additional information about Silicon Graphics implementation of SNA Logical Unit Type 6.2 (LU 6.2) architecture. Also contained in this chapter are the base set of functions and the option sets of LU 6.2, as defined in the IBM *Systems Network Architecture Transaction Programmer's Reference Manual for LU Type 6.2* and *Systems Network Architecture Format and Protocol Reference Manual: Architecture Logic for LU Type 6.2*.

3.1 How the IRIS Implementation Differs from IBM SNA

Each verb is implemented as a C-language library function. After the execution of any verb function, return information is placed in the global variables *snamaj* and *snamin*. This equates to the architected RETURN_CODE. The conversation state is set in the global variable *snastat*. While the architecture does not mandate that the state be explicitly returned to the transaction program, doing so enables the programmer to write the transaction program in a convenient, state-driven fashion.

The IRIS implementation of LU 6.2 from Silicon Graphics maps on an almost one-for-one basis to the IBM architected verbs and their specified parameters. Exceptions are described in this section.

3.1.1 Basic Conversation Verbs

The basic conversation verbs are described below.

GET_TYPE (**gtype**)

A return value of 0 indicates that the conversation type is basic; 1 indicates that the conversation type is mapped. A return value of -1 indicates an error. There is no variable corresponding to ***t****ype* to receive the value.

SEND_ERROR (**snderr**)

An additional value for *type*, SE_ALC (2), indicates an allocation error. If the type returned is SE_ALC, the ***s****ense* field (also not architected) contains the allocation-error sense data. These parameters are reserved for the Transaction Program Initiator (TPI).

WAIT (**waitcv**)

Passes an additional parameter, *count*, that indicates the number of conversation IDs on the following list. All posted conversation IDs are checked when *count* is 0.

3.1.2 Implementation-specific Verbs

Two additional verbs are included in LU 6.2 to control the initial connection between the transaction program and the IRIS SNA SERVER: **attach** and **rattach**. **attach** establishes the connection between the local transaction program and the IRIS SNA SERVER. **rattach** establishes the connection between a remotely evoked transaction program and the conversation that evoked it.

3.1.3 Control Operator Verbs

RESET_SESSION_LIMITS (**rstsl**)

MODE_NAME is handled as two parameters: the first, *all*, uses 0 to indicate that a single mode is to be reset, or 1 to indicate that all modes are to be reset.

The second, *mode*, contains the name of the mode to be reset if *all* is 0. It is ignored if *all* is 1.

PROCESS_SESSION_LIMIT (**procsl**)

LU_NAME and MODE_NAME are not returned. They are handled internally by *s2_cnos*.

DEACTIVATE_SESSION (**dctses**)

Requires two parameters not specified in the architecture: remote (the local name of the remote LU) and mode (the name of the mode for the session).

3.2 Implemented LU 6.2 Function Sets

The following basic conversation verbs are implemented:

- **alcnv** Allocate
- **cnfrm** Confirm
- **cnfrmed** Confirmed
- **dalcnv** Deallocate
- **getatr** Get Attributes
- **rcvwt** Receive-and-Wait
- **rqssnd** Request-to-Send
- **snddta** Send Data
- **snderr** Send Error
- The following control operator verbs are implemented:
- **initsl** Initialize Session Limit
- **rstsl** Reset Session Limit

These option sets are supported:

- Conversations between programs located at the same LU
- Delayed allocation of a session
- Immediate allocation of a session
- Session-level LU-LU verification
- User ID verification
- Profile verification and authorization
- Profile pass-through
- Program-supplied profile
- PIP data
- Logging of data in a system log
- Flush the LU's send buffer
- Prepare-to-Receive
- Long locks
- Post-on-Receipt with wait
- Get Attributes
- Get Conversation Type
- Mapped conversation LU services component
- CHANGE_SESSION_LIMIT verb
- MIN_CONWINNERS_TARGET parameter
- RESPONSIBLE(TARGET) parameter
- DRAIN_TARGET(NO) parameter
- FORCE parameter
- ACTIVATE_SESSION verb
- DEACTIVATE_SESSION verb
- LU parameter verbs
- LU-LU session limit
- Locally known LU names

- Uninterpreted LU names
- Single-session reinitiation
- Maximum RU size bounds
- Contention-winner automatic activation limit

Appendix A

Major and Minor Return Codes

This appendix lists and defines the return codes that may be displayed in the message line area of the Information Panel in the IRIS SNAView main window. Table A-1 lists return codes for functions that complete normally. Table A-2 lists return codes for functions that are aborted. Table A-3 lists return codes for functions that do not complete normally. Table A-4 lists return codes for functions that terminate abnormally with state errors. Table A-5 lists return codes for allocation errors. Table A-6 lists return codes for program errors. Table A-7 lists return codes for deallocation errors. Table A-8 lists return codes for node operator errors.

Major Code	Minor Code	Code Meaning
00	0000	Completed normally. Function completed normally.
00	0001	Completed as negotiated. Function completed as negotiated.
00	0003	Data available. Returned by test conversation if data has arrived on the posted conversation.
00	0004	Control information available. Returned by test conversation if control information has arrived on the posted conversation.
00	0086	Logic error.

Table A-1 Major Code 00 (S2_OK): Function Completed Normally

Major Code	Minor Code	Code Meaning
01	0001	Program not attached. The transaction issued another verb before issuing an **attach** verb. The program must attach before calling other verbs.
01	0002	Duplicate attach attempted. After a previous **attach** verb and before a **detach** verb, the transaction program attempted to reissue the **attach** verb.
01	0003	Invalid conversation ID (rsrc unknown). The conversation specified in the ***cnvid*** parameter is not a valid conversation.
01	0004	Context not set.
01	0009	Null structure pointer parameter passed. A verb was called with a null structure pointer as a parameter.
01	0010	Attach: path is a required parameter. The configuration file path is required for **attach** and **rattach**.
01	0011	Attach: LU name is a required parameter. The LU name is required for attach type AT_LU(0).
01	0012	Attach: Invalid attach type parameter. Valid attach types for a transaction program are AT_LU(0) AT_NODE(1).

Table A-2 Major Code 01 (S2_USAGE): Function Aborted, Usage Error

Major Code	Minor Code	Code Meaning
01	0013	Attach: LU unknown. The specified LU is not configured.
01	0014	Attach: LU not available, limits reached. The LU specified in the attach is not activated, or has already reached the configured limit of attached transaction programs.
01	0015	Attached: TP name is not configured. The transaction program name specified in the request is not configured.
01	0016	Attach, rattach: Configuration file not authorized or invalid. Cannot authorize attach. The attach failed because the transaction program process does not have access permission to the node, the node is not active, or the **attach** verb configuration file parameter was incorrect.
01	0017	Rattach: Conversation ID required. Conversation ID required for remote attach.
01	0018	Rattach: Invalid conversation ID for rattach. The conversation specified in the remote attach is invalid or cannot be remotely attached.
01	0019	Attach, rattach: Invalid wait-time parameter. Invalid wait-time. Valid wait parameters are 0, 1, 2, ..., or -1 to specify no maximum wait time.

Table A-2 (continued) Major Code 01 (S2_USAGE): Function Aborted, Usage Error

Major Code	Minor Code	Code Meaning
01	0020	Attach: Not enough space. Attach rejected. Not enough space to support the transaction program.
01	0021	Attach: SNA Scheduler TP limit reached. Attach rejected. The SNA Scheduler process limit has been reached.
01	0022	Attach: Not enough space in TP.
01	0023	Attach: Pathname is too long.
01	0030	Alcnv: Partner (rlu) is a required parameter. Remote LU name is required.
01	0031	Alcnv: Mode name is a required parameter. Mode name is required.
01	0032	Alcnv: Program name (tpn) is a required parameter. Remote transaction program name is required.
01	0033	Alcnv: Partner LU (rlu) not found. Remote LU unknown. The specified remote LU is not configured for the attached LU.
01	0034	Alcnv: Mode not found. Mode unknown. The specified mode is not configured between the remote LU and the attached LU.

Table A-2 (continued) Major Code 01 (S2_USAGE): Function Aborted, Usage Error

Major Code	Minor Code	Code Meaning
01	0035	Alcnv: Invalid parameter (type). Invalid **alcnv** type. Valid conversation types for **alcnv** are 0 for basic conversation and 1 for mapped conversation.
01	0036	Alcnv: Mapped verb interface not allowed. Mapped conversations not available. Either the attached LU or the remote LU is not configured to support mapped conversations, or the transaction program name specified in the previous attach is not configured to support mapped conversations.
01	0037	Alcnv: Basic verb interface not allowed. Basic conversations not available. The transaction program name specified in the previous attach is not configured to support basic conversations.
01	0038	Alcnv: Invalid parameter (when). Invalid session allocation parameter. Valid session allocation parameters for **alcnv** are AC_WHEN(0), AC_DELAY(1), and AC_IMMED(2).
01	0039	Alcnv: Delayed allocation not allowed. No delayed session allocation. The attached LU is not configured to support delayed session allocation.

Table A-2 (continued) Major Code 01 (S2_USAGE): Function Aborted, Usage Error

Major Code	Minor Code	Code Meaning
01	0040	Alcnv: Immediate allocation not allowed. No immediate session allocation. The attached LU is not configured to support delayed session allocation.
01	0041	Alcnv: Invalid parameter (sync). Invalid sync-level parameter. Valid sync level parameters for **alcnv** are 0 for NONE and 1 for CONFIRM processing.
01	0042	Alcnv: Requested sync level not allowed. Requested sync level not available. The requested sync-level support must be configured for the requested mode and for the transaction program name specified in the previous attach request.
01	0043	Alcnv: PIP not allowed. No PIP data. PIP data-support must be configured for the transaction program name specified in the previous attach request.
01	0044	Alcnv: Invalid PIP length. PIP data too large. Too many PIP parameters were passed in the PIP data, or the PIP data exceeds the maximum length.
01	0045	Alcnv: Invalid parameter (sec). Invalid security-level parameter. Valid security-level parameters are SEC_NONE(0), SEC_SAME(1), and SEC_PGM(2).

Table A-2 (continued) Major Code 01 (S2_USAGE): Function Aborted, Usage Error

Major Code	Minor Code	Code Meaning
01	0046	Alcnv: Requested security not allowed. Requested security level not available. The requested security-level support is not configured. SEC_SAME(1) must be configured for the attached LU and the remote LU. SEC_PGM(2) must be configured for the attached LU, the remote LU, and for the transaction program name specified in the previous attach request.
01	0047	Alcnv: Invalid format for user parameter.
01	0048	Alcnv: Invalid format for password parameter.
01	0049	Invalid mode name. Mode name SNASVCMG cannot be specified when using a mapped conversation.
01	0060	Cnfrm, cnfrmed: Conflict with conversation sync level. Sync level conflict. A **cnfrm** or **cnfrmed** verb was attempted on a conversation that was allocated sync level NONE.

Table A-2 (continued) Major Code 01 (S2_USAGE): Function Aborted, Usage Error

Major Code	Minor Code	Code Meaning
01	0070	Dalcnv: Invalid parameter (type). Invalid **dalcnv** type. Valid **dalcnv** types are: DC_SYNC(1) DC_FLUSH(2) DC_AB_PGM(3) DC_AB_SVC(4) DC_AB_TMR(5) DC_LOCAL(6) DC_CNFRM(7) Invalid **mdalcnv** types are: DC_AB_PGM(3) DC_AB_SVC(4) DC_AB_TMR(5) Valid **mdalcnv** types are: DC_SYNC(1) DC_FLUSH(2) DC_LOCAL(6) DC_CNFRM(7) DC_ABEND(8)
01	0080	Prprcv: Invalid parameter (type). Invalid **prprcv** type. Valid **prprcv** types are 0 for FLUSH and 1 for SYNC.

Table A-2 (continued) Major Code 01 (S2_USAGE): Function Aborted, Usage Error

Major Code	Minor Code	Code Meaning
01	0081	Prprcv: Invalid parameter (lock). Invalid **prprcv** lock parameter. Valid **prprcv** lock parameters are 0 for Short lock, and 1 for Long lock. This parameter is significant for **prprcv** type SYNC.
01	0090	Rcvwt, pstrct: Invalid parameter (fill). Invalid fill parameters for **rcvwt** and **pstrct** are RW_BUFF(0) and RW_LL(1).
01	0091	Invalid parameter (length) cannot be negative. Negative length is invalid. The length parameter is negative.
01	0100	Snddta: Invalid LL. Invalid LL field. The data passed to **snddta** contains an invalid LL field of 0x0000, 0x8000, or 0x8001.
01	0101	Snddta: Invalid LL within the data stream. PS headers not supported. The data passed to **snddta** contains an LL field of 0x0001, which indicates a PS header. This is not supported.
01	0102	Mapping Error: Map name not found.
01	0103	Mapping Error: Map execution failure.
01	0104	Mapping Error: Duplicate map name.

Table A-2 (continued) Major Code 01 (S2_USAGE): Function Aborted, Usage Error

Major Code	Minor Code	Code Meaning
01	0110	Snderr: Invalid parameter (type). Invalid **snderr** type. Valid **snderr** types for transaction programs are SE_PGM(1) and SE_SVC(0).
01	0111	Snderr: SE_ALC(2) is reserved.
01	0120	Waitcv: Negative count value is invalid. Negative **waitcv** count is invalid.
01	0121	Waitcv: Invalid conversation ID on list. Invalid conversation for **waitcv**. The list parameter passed to **waitcv** contained an invalid conversation ID.
01	0122	Waitcv: Listed conversation ID does not have posting active.
01	0123	Waitcv: No conversation with posting is active. If a list of conversations was passed to **waitcv**, none of the conversations had posting active. If no list was passed, then no conversations for the TP had posting active.
01	0124	Waitcv: List size (count) must be positive. Waitcv requires list when non-zero count specified. When the **waitcv** request specifies one or more conversations in the count parameter, those conversations must be passed in the list parameter. No list parameter was passed.

Table A-2 (continued) Major Code 01 (S2_USAGE): Function Aborted, Usage Error

Major Code	Minor Code	Code Meaning
01	0125	Waitcv: List size (count) too large. Waitcv list too large. The number of conversations specified in the **waitcv** request exceeds the ability of the verb interface layer to process the request.
01	0130	Testcv: Invalid parameter (type).
01	0131	Testcv: Posting not active for conversation.
01	0140	Setctx: Path required if *tcbid* is supplied.
01	0141	Setctx: Requested context not found.
01	0300	Not authorized to CO verbs. TP not authorized to the type of Control Operator verb issued.
01	0301	Requested limits are invalid. Limits requested in CNOS verb are invalid.
01	0302	Service Manager Mode not initialized. CNOS verb issued before service manager mode is initialized.
01	0310	Session limit of 0 is invalid. Session limit must be greater than 0.
01	0311	Responsible value invalid. Invalid responsible parameter.

Table A-2 (continued) Major Code 01 (S2_USAGE): Function Aborted, Usage Error

Major Code	Minor Code	Code Meaning
01	0312	C_maxs must be greater than c_minf + c_minb. Sum of the minimum first-speaker sessions and the minimum bidder sessions cannot exceed session limit.
01	0313	Dctses: Invalid parameter (immediate).
01	0314	Rstsl: Invalid parameter (dtrg).
01	0315	Rstsl: Invalid parameter (dsrc).
01	0316	Rstsl: Invalid parameter (force).
01	0320	Next session identifier not found.
01	0321	Invalid session identifier.
01	0400	Name must be less than 20 characters. Node Operator: String parameter too long. Maximum name length is 20 characters.
01	0401	Line is a required parameter. Node Operator: Line name is required.
01	0402	LU is a required parameter. Node Operator: LU name is required.
01	0403	PU is a required parameter. Node Operator: PU name is required.

Table A-2 (continued) Major Code 01 (S2_USAGE): Function Aborted, Usage Error

Major Code	Minor Code	Code Meaning
01	0404	Actsta: Invalid parameter (dial). Node Operator: Invalid dial parameter. Valid dial parameters are 0, 1, 2.
01	0410	Rspque must be NULL. Node Operator: Response queue name unsupported this release.
01	0411	Chgmsgq: Severity must be between 0 and 99.
01	0501	Invalid local LU. The fully qualified LU name of the local LU specified on the **dfnllu** is invalid.
01	0502	Invalid remote LU. The value specified in the local LU session-limit parameter of the **dfnllu** verb is less than the sum of the currently defined LU mode session limits.
01	0503	Invalid mode.
01	0504	Invalid ALS.
01	0505	Invalid Send Pacing Window.
01	0506	Invalid Receive Pacing Window.
01	0507	Invalid Max RU Upper Bound.
01	0508	Invalid Max RU Lower Bound.

Table A-2 (continued) Major Code 01 (S2_USAGE): Function Aborted, Usage Error

Major Code	Minor Code	Code Meaning
01	0509	Invalid Sync Level option.
01	0510	Invalid Single Session Reinitialization option.
01	0511	Invalid Cryptography option.
01	0512	Invalid Maximum Number Sessions. Parallel-session support (YES) is specified on the **dfnrlu** verb but the local LU session limit is 1.
01	0513	Invalid Minimum Number First Speaker. Parallel-session support (YES) and CNOS support (NO) are specified on the **dfnrlu** verb.
01	0514	Invalid Minimum Number First Prebound. CNOS support (YES) and parallel-session support (NO) is not specified on the **dfnrlu** verb.
01	0516	Invalid Blank Mode option.
01	0517	Invalid Network Name operator.
01	0518	Invalid Network Qualifier operator.
01	0519	Invalid Network Name.
01	0520	Invalid Network Qualifier.
01	0521	Invalid Init Type.

Table A-2 (continued) Major Code 01 (S2_USAGE): Function Aborted, Usage Error

Major Code	Minor Code	Code Meaning
01	0522	Invalid Parallel Session option. The fully qualified LU name of the remote LU specified on the **dfnmode** verb is not currently defined for the local LU.
01	0523	Invalid CNOS ALS.
01	0524	Invalid LU Password.
01	0525	Invalid Security Acceptance option.
01	0526	Invalid Password operation.
01	0527	Invalid LU Session Limits.
01	0528	Invalid Conversation Security.
01	0529	Invalid Security operation.
01	0530	Invalid User ID.
01	0531	Invalid Password.
01	0532	Invalid Profile.
01	0533	Invalid Wait.
01	0534	Invalid Max Number of TPs.

Table A-2 (continued) Major Code 01 (S2_USAGE): Function Aborted, Usage Error

Major Code	Minor Code	Code Meaning
01	0535	Invalid LU ID Number.
01	0536	Invalid TP Name.
01	0537	Invalid Status.
01	0538	Invalid Conversation Type.
01	0539	Invalid Security Required.
01	0540	Invalid PIP option.
01	0541	Invalid PIP Number.
01	0542	Invalid PIP check.
01	0543	Invalid Data Mapping.
01	0544	Invalid FMH.
01	0545	Invalid Privilege.
01	0546	Invalid LUW Indicator.
01	0550	Invalid LU.
01	0551	Invalid Partner.

Table A-2 (continued) Major Code 01 (S2_USAGE): Function Aborted, Usage Error

Major Code	Minor Code	Code Meaning
01	0552	Invalid ALS.
01	0553	Not enough space to complete operation.
01	0554	Invalid Remote PU.
01	0555	Invalid Line.
01	0556	Logic error: LSCB not found.
01	0557	Dfnsta: Associated Line not inactive.
01	0558	Logic error: ALCB not found.
01	0560	Dfnmode: Invalid Parameter.
01	0561	Dfnmode: Lower Bound exceeds Upper Bound.
01	0562	Dfnmode: Single Session Reinit not compatible with Partner.
01	0564	Defend: Minf cannot be greater than maxs.
01	0565	Dfnmode: Minpf cannot be greater than minf.
01	0566	Dfnmode: Maxs not compatible with partner.
01	0567	Dfnmode: Blank mode already exists.

Table A-2 (continued) Major Code 01 (S2_USAGE): Function Aborted, Usage Error

Major Code	Minor Code	Code Meaning
01	0570	Dfnrlu: Cnosals and parallel session support not compatible.
01	0571	Dfnrlu: Parallel session support not compatible with maxs.
01	0574	Dfnrlu: Invalid reinit option for parallel-session support.
01	0575	Dfnrlu: CNOS also unknown.
01	0576	Dfnrlu: Null net name already exists.
01	0577	Dfnrlu: Parallel support not compatible with PU type.
01	0580	Logic error: drcb for local PU does not exist.
01	0581	Dfnllu: Session limits exceed LU session limits specified.
01	0582	Dfnllu: Network name specified after initsl.
01	0583	Dfnllu: Security parameters to be deleted not found.
01	0584	Dfnllu: LU ID already specified.
01	0585	Dfnllu: LU ID must be specified when lucb is initialized.
01	0586	Dfnllu: LU ID cannot be updated when session limit initialized.

Table A-2 (continued) Major Code 01 (S2_USAGE): Function Aborted, Usage Error

Major Code	Minor Code	Code Meaning
01	0590	Dfnllu: Security access parameters conflict with security required.
01	0591	Dfnllu: Security parameters to be deleted not found.
01	0600	Too many resources defined—no internal address available.
01	0601	Invalid Exchange ID.
01	0602	Invalid Master Device.
01	0603	Invalid Monitor Timer.
01	0604	Invalid NOOP Messages.
01	0605	Invalid LOG Messages.
01	0606	Invalid Debug Messages.
01	0607	Invalid PU Name.
01	0608	Invalid CPU ID.
01	0609	Invalid Line Name.
01	0610	Invalid Line Type.

Table A-2 (continued) Major Code 01 (S2_USAGE): Function Aborted, Usage Error

Major Code	Minor Code	Code Meaning
01	0611	Invalid Device Name.
01	0612	Invalid SDLC Role.
01	0613	Invalid Connection Type.
01	0614	Invalid NRZI.
01	0615	Invalid Half-duplex.
01	0616	Invalid Max BTU.
01	0617	Invalid Max Retries.
01	0618	Invalid Idle.
01	0619	Invalid Nonprod Rcv Time.
01	0620	Invalid Max I-Frames.
01	0621	Invalid Rate.
01	0622	Invalid SDLC Address.
01	0623	Invalid Phone Name.
01	0624	Dfnsta: Exceeded maximum number of stations (8).

Table A-2 (continued) Major Code 01 (S2_USAGE): Function Aborted, Usage Error

Major Code	Minor Code	Code Meaning
01	0626	Dfnline: Multidrop specified and role not primary.
01	0627	Dfnsta: One station already defined for leased line.
01	0628	Dfnline: Line already exists.
01	0630	Dfnsta: Line not specified.
01	0631	Dfnsta: PU not specified.
01	0632	Dfnsta: Station already exists.
01	0634	Dspline: Invalid line name.
01	0635	Dspline: End of line list.
01	0636	Dspline: No lines defined.
01	0637	Dspcp: Invalid control point name.
01	0638	Dspcp: End of PU list.
01	0639	Dspcp: No PUs defined.
01	0640	Dltcbl: Mode unknown.
01	0641	Dltcbl: TP Name unknown.

Table A-2 (continued) Major Code 01 (S2_USAGE): Function Aborted, Usage Error

Major Code	Minor Code	Code Meaning
01	0642	Dltcbl: Object is in use.
01	0646	Dltcbl: Local LU name not specified.
01	0647	Dltcbl: Remote LU name not specified.
01	0648	Dltcbl: No parameters specified.
01	0650	Dfnrpu: Neither XID or CPID specified.
01	0651	Dfnrpu: Both XID and CPID specified.
01	0659	Control point is not a host.
01	0660	Dspllu: Userid invalid.
01	0661	Dspllu: End of security list.
01	0662	No userid found.
01	0663	Dspllu: Invalid profile.
01	0664	Dspllu: Invalid LU name.
01	0665	Dspllu: End of LU list.

Table A-2 (continued) Major Code 01 (S2_USAGE): Function Aborted, Usage Error

Major Code	Minor Code	Code Meaning
01	0666	Dspllu: No LU defined.
01	0667	Dsprlu: Invalid Remote LU name.
01	0668	Dsprlu: No more Remote LUs.
01	0669	Dsprlu: No Remote LUs defined.
01	0670	Dspmode: Invalid mode name.
01	0671	Dspmode: End of mode list.
01	0672	Dspmode: No modes defined.
01	0673	Dsptp: Invalid TP name.
01	0674	Dsptp: End of TP list.
01	0675	Dsptp: No TP defined.
01	0676	Dsptp: No Network Name.
01	0677	Invalid NEXT parameter value.
01	0691	Control point name is required.
01	0692	No more remote LUs or secondary LUs.

Table A-2 (continued) Major Code 01 (S2_USAGE): Function Aborted, Usage Error

Major Code	Minor Code	Code Meaning
01	0693	LU_ID value does not match any LU.
01	0830	User crash code not described.
01	0850	*s2_schd* is not active. The SNA Scheduler is not active.
01	0860	*s2_schd* has terminated abnormally. The SNA Scheduler is no longer active. The transaction program has been detached.
01	0870	Time out—request not recovered.
01	0880	Time out—request recovered.
01	0890	Time out—message queue full.
01	0900	Verb logic error. An internal logic error occurred while processing the verb.
01	0901	Duplicate *s2_tpi*. Duplicate TPI attach attempted.
01	1000	Invalid NAU name. The NAU must be no greater than eight characters and all characters must be of symbol-string Type A.

Table A-2 (continued) Major Code 01 (S2_USAGE): Function Aborted, Usage Error

Major Code	Minor Code	Code Meaning
01	1001	Invalid NAU type. Valid values are P_PU and P_LU.
01	1002	Invalid when parameter. Valid values for **sndru** are P_WHEN and P_NONE. Valid values for **alnau** are P_WHEN and P_IMMED.
01	1003	Invalid block parameter. Valid values are P_BLOCK and P_NBLOCK.
01	1004	Invalid bind parameter. Buffer length is greater than zero and no bind pointer supplied.
01	1005	Resources temporarily unavailable. Retry.
01	1006	Not enough TP space to allocate another NAU.
01	1008	Not enough resources. An attempt was made to send an RU that exceeds the maximum send size when **sndru** is issued in non-blocking mode.
01	1009	Null structure pointer supplied to a PI verb.
01	1010	Invalid structure type supplied for **initpi**.

Table A-2 (continued) Major Code 01 (S2_USAGE): Function Aborted, Usage Error

Major Code	Minor Code	Code Meaning
01	1011	NAU identifier invalid. The NAU identifier supplied on the verb call is not the ***nauid*** of a currently allocated NAU.
01	1012	Specified NAU not allocated. An attempt was made to use the ***nauid*** of an NAU that is pending allocation.
01	1013	Invalid svcid. Service identifier is not one of the valid values or is inconsistent with the NAU type specified on the **alnau** verb.
01	1014	The pointer to the pru structure is NULL. This is a required field.
01	1015	The RU command is zero.
01	1016	The pointer to the RU buffer is NULL. In this verb call, the RU buffer is required.
01	1017	The pointer to the ufsm structure is NULL. This is a required field for the **gfsm** verb.
01	1018	RU sense data is required. Zero is an invalid value.

Table A-2 (continued) Major Code 01 (S2_USAGE): Function Aborted, Usage Error

Major Code	Minor Code	Code Meaning
01	1019	The flow parameter of the **ssync** verb is invalid. Valid values are P_SND, P_RCV, and P_FLOW.
01	1020	The wait parameter is invalid. Valid values are P_WAIT, P_NOWAIT, or a positive numeric value representing the time, in seconds, that the verb should wait.
01	1021	The loc_pac parameter is invalid. This is an optional field, but if specified, valid values are P_PACE and P_NPACE.
01	1022	Required UFSM pointer is NULL. The pointer to the ***ufsm*** structure is NULL. This field is required on the **rcvru** verb and on the **alnau** verb issued with block = P_BLOCK.
01	1023	svcid inconsistent with NAU type. The ***svcid*** parameter is compatible with the nau type of the specified NAU.
01	1024	The **dfnslu** verb was issued while LU 0-3 was not configured by the configuration manager.
01	1025	LU not defined. A secondary LU with the name specified in ***naunm*** is not currently defined.

Table A-2 (continued) Major Code 01 (S2_USAGE): Function Aborted, Usage Error

Major Code	Minor Code	Code Meaning
01	1026	LU not available. The LU specified on the **alnau** verb is already allocated.
01	1027	PU not defined. A PU with the name specified in ***naunm*** is not currently defined.
01	1028	PU not available. The PU specified on the **alnau** verb is already allocated.
01	1030	Attach type is not specified. The Transaction Program that issued the verb did not attach with attach type AT_ANODE or AT_ALU.
01	1031	Send check. The reply to the **sndru** verb received from the SNA Scheduler indicated a send check.
01	1032	Negative response. The reply to the **sndru** verb received from the SNA Scheduler indicated a negative response.
01	1033	Receive check. The reply to the **sndru** verb received from the SNA Scheduler indicated a receive check.
01	1034	Session lost.

Table A-2 (continued) Major Code 01 (S2_USAGE): Function Aborted, Usage Error

Major Code	Minor Code	Code Meaning
02	0001	No information for conversation.
02	0002	Conversation not posted.
02	0003	Request-to-send not received.

Table A-3 Major Code 02 (S2_UNSUC): Completed Unsuccessfully

Major Code	Minor Code	Code Meaning
03	0001	Conversation State error. Request is illegal in the current conversation state.
03	0002	Logical Record State error. Request is illegal because the current logical record has not been completed.
03	0003	Conv for waitcv not S2_RECV. A conversation specified in the **waitcv** request is not in receive state.

Table A-4 Major Code 03 (S2_STATE): Function Aborted, State Error

Major Code	Minor Code	Code Meaning
05	0001	Remote TP not available—no retry. The transaction program could not be started on the remote system because of a lack of resources, which is not temporary. Retry is not suggested.
05	0002	Remote TP not available—retry. The transaction program could not be started on the remote system because of a temporary resource shortage. Retry is suggested.
05	0003	Remote LU does not support conversation type. The remote LU does not support the requested conversation type.
05	0004	PIP data not supported by remote. Program initialization data is not supported by the remote LU.
05	0005	PIP specification error. Program initialization data was specified incorrectly.
05	0006	Invalid access security information. Security information was not specified correctly.
05	0007	Remote program does not support requested sync level.
05	0008	Remote TP not recognized by remote LU. The transaction program requested was not recognized at the remote LU.
05	0009	Allocation failure—no retry.

Table A-5 Major Code 05 (S2_ALCER): Allocation Error

Major Code	Minor Code	Code Meaning
05	0010	Allocation failure—retry.
05	0011	First Speaker session not available. The session is not immediately available.
05	0012	Local resource allocation failure.
05	0013	Cannot authorize security access.
05	0022	Allocation rejected by Resource Manager.
05	0099	Unknown sense code data received.

Table A-5 (continued) Major Code 05 (S2_ALCER): Allocation Error

Major Code	Minor Code	Code Meaning
07	0001	Program Error: Current logical record not truncated. Partner has reported a program error; the current logical record was not truncated.
07	0002	Program Error: Current logical record truncated. Partner has reported a program error; current logical record was truncated.
07	0003	Program Error: Data may have been purged. Partner has reported a program error; current logical record may have been purged.
07	0011	Service Error: Current logical record not truncated. Service Transaction Program has reported an error; current logical record was not truncated.
07	0012	Service Error: Current logical record was truncated. Service Transaction Program has reported an error; current logical record was truncated.
07	0013	Service Error: Data may have been purged. Service transaction program has reported a program error; current logical record may have been purged.
07	0020	Mapping Error: FMH not supported.
07	0021	Mapping Error: Mapping not supported.

Table A-6 Major Code 07 (S2_PGMER): Program Error

Major Code	Minor Code	Code Meaning
09	0000	Normal deallocation by partner. Partner has deallocated the conversation.
09	0001	Abnormal deallocation by partner. Partner has terminated the conversation abnormally.
09	0002	Abnormal deallocation by service program. A service transaction program has terminated the conversation abnormally.
09	0003	Abnormal deallocation verb time-out. Time-out has occurred.
09	0004	Abnormal deallocation, Session Failure. The session has failed.
09	0005	Abnormal deallocation by partner.

Table A-7 Major Code 09 (S2_DEALC): Deallocated

Major Code	Minor Code	Code Meaning
10	0063	Limits not zero.
10	0064	Requested limits exceed configuration. This error should be interpreted as ACTIVATION_FAILURE_RETRY.
10	0065	Sum of minimums exceed maximum session. The sum of the minimum first-speaker and bidder sessions exceeds the requested maximum.
10	0066	Invalid SNASVCMG limits. SNASVCMG limits must be 2: 1 bidder and 1 first speaker.
10	0067	SNASVCMG mode not initialized.
10	0068	Mode limits are closed.
10	0069	Chgsl not valid for this mode.
10	0070	SNASVCMG mode not reset, other modes still not reset. SNASVCMG mode cannot be reset because user modes are still open.
10	0071	CNOS race with remote—remote won.
10	0072	Partner does not recognize mode.
10	0073	CNOS is in process locally.

Table A-7 (continued) Major Code 09 (S2_DEALC): Deallocated

Major Code	Minor Code	Code Meaning
10	0074	CNOS allocation error.
10	0075	CNOS resource failure. The SNASVCMG session with the partner LU either could not be started or failed.
10	0076	Insufficient space for a new session.
10	0077	Partner LU is not active.
10	0078	Modes incompatible: session not started.

Table A-7 (continued) Major Code 09 (S2_DEALC): Deallocated

Major Code	Minor Code	Code Meaning
11	0001	Node operator verb failure.
11	0002	Node operator verb not recognized.
11	0003	Attach of type AT_NODE. Transaction Program must attach with type AT_NODE to use node operator verbs.
11	0010	Link name not recognized.
11	0011	Adjacent link station not recognized.
11	0012	Physical unit not recognized.
11	0013	Earlier request still active.
11	0014	Dial in or dial out required.
11	0015	Logical unit name not recognized.
11	0016	Message queue name not recognized.
11	0017	Message queue not enabled.
11	0018	No message in message queue.

Table A-8 Major Code 11 (S2_NPERR): Node Operator Error

Major Code	Minor Code	Code Meaning
11	0100	Actpu failure.
11	0110	Actlu failure.
11	0120	Dctpu failure.
11	0130	Dctlu failure.
11	0140	Chgmsgq failure.
11	0150	Dspmsgq failure.
11	0190	Rtvnmsg failure.

Table A-8 (continued) Major Code 11 (S2_NPERR): Node Operator Error

Appendix B

API Verb Catalog

This appendix lists all of the API verbs in alphabetical order. These verbs and their supporting man pages are used with this guide (referred to as *LU 6.2* in the table) and the *IRIS SNA SERVER Administration Guide* (referred to as *SNA* in the table).

Verb	Verb's Full Name	Verb Type	Guide
accru	Accept Request Unit	PI	SNA
actline	Activate Line	Node Operator	SNA
actlu	Activate Logical	Node Operator Unit	SNA
actpu	Activate Physical Unit	Node Operator	SNA
actses	Activate Session	Session Control	LU 6.2
actsta	Activate Station	Node Operator	SNA
alcnv	Allocate	Basic Conversation	LU 6.2
alnau	Allocate NAU	PI	SNA
atoe	ASCII to EBCDIC Translation	Implementation-specific	SNA
attach	Local Attach	Implementation-specific	SNA
chgmsgq	Change Message Queue	Node Operator	SNA
chgsl	Change Session Limit	CNOS	LU 6.2
cnfrm	Confirm	Basic Conversation	LU 6.2
cnfrmed	Confirmed	Basic Conversation	LU 6.2
dalcnv	Deallocate	Basic Conversation	LU 6.2
dalnau	Deallocate NAU	PI	SNA
dctline	Deactivate Line	Node Operator	SNA
dctlu	Deactivate Logical Unit	Node Operator	SNA
dctpu	Deactivate Physical Unit	Node Operator	SNA
dctses	Deactivate Session	Session Control	LU 6.2
dctsta	Deactivate Station	Node Operator	SNA
detach	Detach	Implementation-specific	SNA
dfncp	Define Control Point	Define	SNA

Verb	Verb's Full Name	Verb Type	Guide
dfnline	Define Line	Define	SNA
dfnllu	Define Local LU	Define	SNA
dfnmode	Define Mode	Define	SNA
dfnnode	Define Node	Define	SNA
dfnrlu	Define Remote LU	Define	SNA
dfnslu	Define Secondary LU	Define	SNA
dfnsta	Define Station	Define	SNA
dfntp	Define Transaction Program	Define	SNA
dltcbl	Delete LU Control Block	Define	SNA
dltcbu	Delete Control Block	Define	SNA
dspcp	Display Control Point	Display	SNA
dspcph	Display Host Control Point	Display	SNA
dspline	Display Line	Display	SNA
dspllu	Display Local LU	Display	SNA
dspmaj	Display Major Code	Implementation-specific	SNA
dspmin	Display Minor Code	Implementation-specific	SNA
dspmode	Display Mode	Display	SNA
dspmsgq	Display Message Queue	Node Operator	SNA
dspnode	Display Node	Display	SNA
dsprlu	Display Remote LU	Display	SNA
dspses	Display Session	Display	SNA
dspslu	Display Secondary LU	Display	SNA
dspsta	Display Station	Display	SNA

Verb	Verb's Full Name	Verb Type	Guide
dsptp	Display Transaction Program	Display	SNA
etoa	EBCDIC to ASCII Translation	Implementation-specific	SNA
flush	Flush	Basic Conversation	LU 6.2
getatr	Get Attributes	Basic Conversation	LU 6.2
gfsm	Get Finite State Machine	PI	SNA
gsync	Get Sync Point	PI	SNA
gtype	Get Type	Type-independent Conversation	LU 6.2
initcbl	Initialize LU Definition Structure	Define	SNA
initcbu	Initialize Node Definition Structure	Define	SNA
initpi	Initialize Verb Data Structure	PI	SNA
initsl	Initialize Session Limit	CNOS	LU 6.2
malcnv	MC Allocate	Mapped Conversation	LU 6.2
mcnfrm	MC Confirm	Mapped Conversation	LU 6.2
mdalcnv	MC Deallocate	Mapped Conversation	LU 6.2
mflush	MC Flush	Mapped Conversation	LU 6.2
mgetatr	MC Get Attributes	Mapped Conversation	LU 6.2
mprprcv	MC Prepare to Receive	Mapped Conversation	LU 6.2
mpstrct	MC Post on Receipt	Mapped Conversation	LU 6.2
mrcvim	MC Receive Immediate	Mapped Conversation	LU 6.2

Verb	Verb's Full Name	Verb Type	Guide
mrcvwt	MC Receive and Wait	Mapped Conversation	LU 6.2
mrqssnd	MC Request to Send	Mapped Conversation	LU 6.2
msnddta	MC Send Data	Mapped Conversation	LU 6.2
msnderr	MC Send Error	Mapped Conversation	LU 6.2
mtestcv	MC Test	Mapped Conversation	LU 6.2
procsl	Process Session Ldimit	CNOS	LU 6.2
prprcv	Prepare to Receive	Basic Conversation	LU 6.2
prtnmsg	Print Node Message	Implementation-specific	SNA
pstrct	Post on Receipt	Basic Conversation	LU 6.2
rattach	Remote Attach	Implementation-specific	SNA
rcvim	Receive Immediate	Basic Conversation	LU 6.2
rcvru	Receive Request Unit	PI	SNA
rcvwt	Receive and Wait	Basic Conversation	LU 6.2
rejru	Reject Request Unit	PI	SNA
rqssnd	Request to Send	Basic Conversation	LU 6.2
rstsl	Reset Session Limit	CNOS	LU 6.2
rtvnmsg	Retrieve Node Message	Node Operator	SNA
setctx	Set Context	Implementation-specific	SNA
snddta	Send Data	Basic Conversation	LU 6.2
snderr	Send Error	Basic Conversation	LU 6.2
sndru	Send Request Unit	PI	SNA
ssync	Set Sync Point	PI	SNA
testcv	Test	Basic Conversation	LU 6.2
waitcv	Wait	Type-independent Conversation	LU 6.2

Appendix C

Man Pages

This appendix contains the following category (1M) and (3X) man pages related to the IRIS SNA LU6. These man pages are listed in alphabetical order.

Core Programs

- s2_cnos (1M)
- s2_lucp (1M)
- s2_tpi (1M)

Conversation Mapped

- malcnv (3X)
- mc_map (3X)
- mcnfrm (3X)
- mcnfrmed (3X)
- mdalcnv (3X)
- mflush (3X)
- mgetatr (3X)
- mprprcv (3X)
- mpstrct (3X)
- mrcvim (3X)
- mrcvwt (3X)
- mrqssnd (3X)

- msnddta (3X)
- msnderr (3X)
- mtestcv (3X)

Conversation/Type-independent

- gtype (3X)
- waitcv (3X)

Conversation Basic

- alcnv (3X)
- cnfrm (3X)
- cnfrmed (3X)
- dalcnv (3X)
- flush (3X)
- getatr (3X)
- prprcv (3X)
- pstrct (3X)
- rcvim (3X)
- rcvwt (3X)
- rqssnd (3X)
- snddta (3X)
- snderr (3X)
- testcv (3X)

Control Operator

- chgsl (3X)
- dsps l (3X)
- initsl (3X)
- procsl (3X)
- rstsl (3X)

Session Control

- actses (3X)
- dctses (3X)

Index

T

U

V