z/OS Communications Server
3.2

SNA Programmer's LU 6.2 Guide

SC27-3669-70

Note:

Before using this information and the product it supports, be sure to read the general information under "Notices" on page 341.

This edition applies to 3.1 of z/OS® (5655-ZOS), and to subsequent releases and modifications until otherwise indicated in new editions.

Last updated: 2025-09-24

Contents

Figures

Tables

About this document

This document is intended to help customers write VTAM® application programs that use VTAM's logical unit (LU) 6.2 support.

This document describes the VTAM application programming interface's support for LU type 6.2. It describes both the APPCCMD macroinstruction that provides LU 6.2 services and the programming techniques for using the macroinstruction. It applies to programs that use only LU 6.2 sessions and to those that use LU 6.2 sessions with other session types. (VTAM-supported LU 6.2 application programs are those whose APPL definition statement includes APPC=YES.)

This book is intended to teach how to use VTAM's application programming interface (API) for LU 6.2 services. It should be used together with z/OS Communications Server: SNA Programmer's LU 6.2 Reference, which contains a complete list of all VTAM macroinstructions, return codes and DSECTs used for LU 6.2 programming.

This document must also be used with z/OS Communications Server: SNA Programming, which contains information common to all VTAM application programs not contained in this manual.

All of the exit routines described in this document are user exit routines. The function of user exit routines is described in Chapter 12, “Using exit routines,” on page 239. The user exit routines are also called *exit routines* and *exits* in this book.

Who should use this document

This document is for system programmers who code VTAM application programs that use VTAM's LU 6.2 support. This audience can include programmers who are modifying existing programs or writing new ones. The information can also be of use to planners who are estimating the amount of work required to use the VTAM interface.

You should be familiar with LU 6.2 architecture before you write LU 6.2 programs. The *SNA Transaction Programmer's Reference Manual for LU Type 6.2* provides this familiarity and is, therefore, a core requisite for using this document.

Typographic conventions used in this document

This publication uses the following typographic conventions:

- Commands that you enter verbatim onto the command line are presented in **bold**.
- Variable information and parameters that you enter within commands, such as file names, are presented in *italic*.
- System responses are presented in `monospace`.

Note: In this manual, examples of macroinstructions have blank spaces between macroinstruction operands. This convention is used to improve the formatting of the manual. When coding the macroinstructions, you must delete the blanks.

Figure 1 shows the conventions used in this book to illustrate the parts of a network.

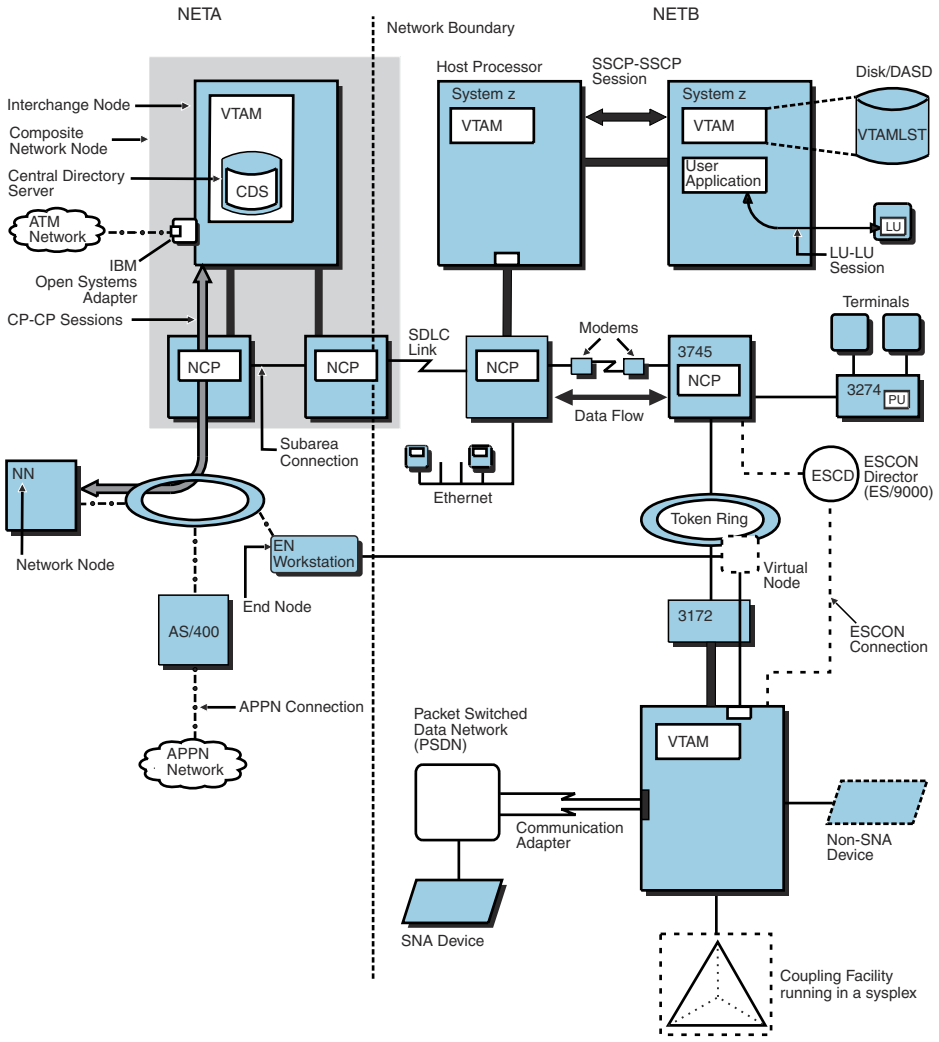


Figure 1. Conventions used in network illustrations

How this document is organized

This document is organized into the following topics:

- Chapter 1, “Understanding VTAM LU 6.2 application programs,” on page 1 discusses data-transfer application programs that conform to the rules of LU type 6.2 protocols.
- Chapter 2, “LU 6.2 and the VTAM API,” on page 11 describes the return codes.discusses the interface VTAM offers for LU 6.2 services and compares it to the interface offered for non-LU 6.2 session types.
- Chapter 3, “How VTAM implements LU 6.2 architecture,” on page 31 describes the functions that VTAM implements, the functions that the application program implements (pass-through verbs and mapped conversation verbs), and the optional LU 6.2 functions.
- Chapter 4, “Designing programs to use LU 6.2 services,” on page 47 describes the designing programs to use LU 6.2 services.
- Chapter 5, “Coding the APPCCMD macroinstruction,” on page 67 describes the coding the APPCCMD macroinstruction.
- Chapter 6, “Managing sessions,” on page 83 describes how a VTAM LU 6.2 application program can manage sessions.
- Chapter 7, “Allocating a conversation,” on page 149 describes how a VTAM LU 6.2 application program can allocate a conversation.

- Chapter 8, “Deallocating a conversation,” on page 167 describes how a VTAM LU 6.2 application program can deallocate a conversation.
- “Sending information” on page 168 describes how a VTAM LU 6.2 application program can send information.
- Chapter 10, “Receiving information,” on page 193 describes how a VTAM LU 6.2 application program can receive information.
- Chapter 11, “Sending and receiving data using high performance data transfer,” on page 217 describes how a VTAM LU 6.2 application program can send and receive data using high performance data transfer.
- Chapter 12, “Using exit routines,” on page 239 discusses the use of exit routines for LU 6.2 application programs.
- Chapter 13, “VTAM's LU 6.2 security options,” on page 249 describes the VTAM's LU 6.2 security options.
- Chapter 14, “Handling errors,” on page 263 discusses how to analyze feedback from the VTAM program for errors and special conditions associated with the APPCCMD macroinstruction.
- Appendix A, “Conversation states,” on page 281, Appendix B, “APPCCMD macroinstruction overview,” on page 289, Appendix C, “Example of a sample LU 6.2 application program,” on page 305, Appendix D, “Example of retrieving information for a mode and any restored sessions,” on page 329, Appendix E, “Architectural specifications,” on page 335, and Appendix G, “Accessibility,” on page 339 provide additional information for this document.

How to use this document

How to provide feedback to IBM

We welcome any feedback that you have, including comments on the clarity, accuracy, or completeness of the information. See, How to send feedback to IBM® for additional information.

Conventions and terminology that are used in this information

Commands in this information that can be used in both TSO and z/OS UNIX environments use the following conventions:

- When describing how to use the command in a TSO environment, the command is presented in uppercase (for example, NETSTAT).
- When describing how to use the command in a z/OS UNIX environment, the command is presented in bold lowercase (for example, **netstat**).
- When referring to the command in a general way in text, the command is presented with an initial capital letter (for example, Netstat).

All the exit routines described in this information are *installation-wide exit routines*. The installation-wide exit routines also called installation-wide exits, exit routines, and exits throughout this information.

The TPF logon manager, although included with VTAM, is an application program; therefore, the logon manager is documented separately from VTAM.

Samples used in this information might not be updated for each release. Evaluate a sample carefully before applying it to your system.

z/OS no longer supports mounting HFS data sets (The POSIX style file system). Instead, a z/OS File System (zFS) can be implemented. The term hierarchical file system, abbreviated as HFS, is defined as a data structure that has a hierarchical nature with directories and files. References to hierarchical file systems or HFS might still be in use in z/OS Communications Server publications.

Network Express and Open Systems Adapter-Express (OSA-Express) terminology:

- The Network Express feature is introduced with the IBM z17 processor family. The Network Express feature is the next generation of Open Systems Adapter (OSA) technology. The term OSA (Open Systems Adapter) is carried forward with Network Express. The IBM z17 processor supports both the Network Express and the OSA-Express7S features. In this information, when a general reference is made to OSA that applies to all these features, then the term *OSA* is used, and the acronym will appear in italics. This formatting style and guideline for usage for the term OSA is used throughout this document. When a distinction is necessary, then the specific feature name is used such as the Network Express feature
- The Network Express feature is defined as channel (CHPID) type OSH (Open System Adapter for Hybrid networks) that might operate in either 10 GbE or 25 GbE link speed. When this term is used in this information, the processing being described applies to either link speed. If processing is applicable to only one link speed, the full terminology, for instance, IBM 25 GbE Network Express will be used.
- Network Express is defined with new system architecture called Enhanced Queued Direct I/O (EQDIO). In this information there are many references to QDIO or OSA/QDIO. When the reference applies to both QDIO and EQDIO the reference just indicates OSA. When the reference is specific to the QDIO or EQDIO architecture, then the specific architecture is referenced, for example, OSA/QDIO or OSA/EQDIO. Some OSA references also use or include the channel type for OSA such as OSD (QDIO). When the reference applies to both features, then the term OSA is used. When a distinction is necessary then the specific channel or architecture type is used, OSD/QDIO or OSH/EQDIO.

Shared Memory Communications over Remote Direct Memory Access (SMC-R) terminology

- *RoCE* , which is a generic term representing IBM® 10 GbE RoCE Express, IBM 10 GbE RoCE Express2, IBM 25 GbE RoCE Express2, IBM 10 GbE RoCE Express3, IBM 25 GbE RoCE Express3, IBM 10 GbE Network Express and IBM 25 GbE Network Express feature capabilities. When this term is used in this information, the processing being described applies to all of these features. If processing is applicable to only one feature, the full terminology, for instance, Network Express will be used.
- RoCE Express2, which is a generic term representing an IBM RoCE Express2 feature that might operate in either 10 GbE or 25 GbE link speed. When this term is used in this information, the processing being described applies to either link speed. If processing applies to only one link speed, the full terminology, for instance, IBM 25 GbE RoCE Express2 will be used.
- RoCE Express3, which is a generic term representing an IBM RoCE Express3 feature that might operate in either 10 GbE or 25 GbE link speed. When this term is used in this information, the processing being described applies to either link speed. If processing applies to only one link speed, the full terminology, for instance, IBM 25 GbE RoCE Express3 will be used.
- Network Express, which is a generic term representing an Network Express feature that might operate in either 10 GbE or 25 GbE link speed. When this term is used in this information, the processing being described applies to either link speed. If processing is applicable to only one link speed, the full terminology, for instance, IBM 25 GbE Network Express will be used. When configured with a CHPID type of NETH, the Network Express feature may operate as an RDMA network interface card.
- RDMA network interface card (RNIC), which is used to refer to the IBM 10 GbE RoCE Express, IBM 10 GbE RoCE Express2, IBM 25 GbE RoCE Express2, IBM 10 GbE RoCE Express3, or IBM 25 GbE RoCE Express3, IBM 10 GbE Network Express or IBM 25 GbE Network Express feature.
- Shared RoCE environment, which means that the *ROCE* feature can be used concurrently, or shared, by multiple operating system instances. The feature is considered to operate in a shared RoCE environment even if you use it with a single operating system instance.

Clarification of notes

Information traditionally qualified as Notes is further qualified as follows:

Attention
Indicate the possibility of damage

Guideline
Customary way to perform a procedure

Note
Supplemental detail

Rule
Something you must do; limitations on your actions

Restriction
Indicates certain conditions are not supported; limitations on a product or facility

Requirement
Dependencies, prerequisites

Result
Indicates the outcome

Tip
Offers shortcuts or alternative ways of performing an action; a hint

Prerequisite and related information

z/OS Communications Server function is described in the z/OS Communications Server library. Descriptions of those documents are listed in "Bibliography" on page 345, in the back of this document.

Required information

Before using this product, you should be familiar with TCP/IP, VTAM, MVS, and UNIX System Services.

Softcopy information

Softcopy publications are available in the following collection.

Titles	Description
IBM Z Redbooks	The IBM Z® subject areas range from e-business application development and enablement to hardware, networking, Linux®, solutions, security, parallel sysplex, and many others. For more information about the Redbooks® publications, see http://www.redbooks.ibm.com/ and http://www.ibm.com/systems/z/os/zos/zfavorites/.

Other documents

This information explains how z/OS references information in other documents.

When possible, this information uses cross-document links that go directly to the topic in reference using shortened versions of the document title. For complete titles and order numbers of the documents for all products that are part of z/OS, see z/OS Information Roadmap (SA23-2299). The Roadmap describes what level of documents are supplied with each release of z/OS Communications Server, and also describes each z/OS publication.

To find the complete z/OS library, visit the z/OS library in IBM Documentation (https://www.ibm.com/docs/en/zos).

Relevant RFCs are listed in an appendix of the IP documents. Architectural specifications for the SNA protocol are listed in an appendix of the SNA documents.

The following table lists documents that might be helpful to readers.

Title	Number
DNS and BIND, Fifth Edition, O'Reilly Media, 2006	ISBN 13: 978-0596100575
Routing in the Internet, Second Edition, Christian Huitema (Prentice Hall 1999)	ISBN 13: 978-0130226471
sendmail, Fourth Edition, Bryan Costales, Claus Assmann, George Jansen, and Gregory Shapiro, O'Reilly Media, 2007	ISBN 13: 978-0596510299
SNA Formats	GA27-3136

Title	Number
TCP/IP Illustrated, Volume 1: The Protocols, W. Richard Stevens, Addison-Wesley Professional, 1994	ISBN 13: 978-0201633467
TCP/IP Illustrated, Volume 2: The Implementation, Gary R. Wright and W. Richard Stevens, Addison-Wesley Professional, 1995	ISBN 13: 978-0201633542
TCP/IP Illustrated, Volume 3: TCP for Transactions, HTTP, NNTP, and the UNIX Domain Protocols, W. Richard Stevens, Addison-Wesley Professional, 1996	ISBN 13: 978-0201634952
TCP/IP Tutorial and Technical Overview	GG24-3376
Understanding LDAP	SG24-4986
z/OS Cryptographic Services System SSL Programming	*SC14-7495*
z/OS IBM Tivoli Directory Server Administration and Use for z/OS	*SC23-6788*
z/OS JES2 Initialization and Tuning Guide	*SA32-0991*
z/OS Problem Management	*SC23-6844*
z/OS MVS Diagnosis: Reference	*GA32-0904*
z/OS MVS Diagnosis: Tools and Service Aids	*GA32-0905*
z/OS MVS Using the Subsystem Interface	*SA38-0679*
z/OS Program Directory	*GI11-9848*
z/OS UNIX System Services Command Reference	*SA23-2280*
z/OS UNIX System Services Planning	*GA32-0884*
z/OS UNIX System Services Programming: Assembler Callable Services Reference	*SA23-2281*
z/OS UNIX System Services User's Guide	*SA23-2279*
z/OS C/C++ Runtime Library Reference	*SC14-7314*
OSA-Express Customer's Guide and Reference	*SA22-7935*

Redbooks publications

The following Redbooks publications might help you as you implement z/OS Communications Server.

Title	Number
IBM z/OS Communications Server TCP/IP Implementation, Volume 1: Base Functions, Connectivity, and Routing	SG24-8096
IBM z/OS Communications Server TCP/IP Implementation, Volume 2: Standard Applications	SG24-8097
IBM z/OS Communications Server TCP/IP Implementation, Volume 3: High Availability, Scalability, and Performance	SG24-8098
IBM z/OS Communications Server TCP/IP Implementation, Volume 4: Security and Policy-Based Networking	SG24-8099
IBM Communication Controller Migration Guide	SG24-6298
IP Network Design Guide	SG24-2580
Managing OS/390 TCP/IP with SNMP	SG24-5866

Title	Number
Migrating Subarea Networks to an IP Infrastructure Using Enterprise Extender	SG24-5957
SecureWay Communications Server for OS/390 V2R8 TCP/IP: Guide to Enhancements	SG24-5631
SNA and TCP/IP Integration	SG24-5291
TCP/IP in a Sysplex	SG24-5235
TCP/IP Tutorial and Technical Overview	GG24-3376
Threadsafe Considerations for CICS	SG24-6351

Where to find related information on the Internet

z/OS

This site provides information about z/OS Communications Server release availability, migration information, downloads, and links to information about z/OS technology

http://www.ibm.com/systems/z/os/zos/

z/OS Internet Library

Use this site to view and download z/OS Communications Server documentation

http://www.ibm.com/systems/z/os/zos/library/bkserv/

z/OS Communications Server product

The page contains z/OS Communications Server product introduction

https://www.ibm.com/products/zos-communications-server

IBM Communications Server product support

Use this site to submit and track problems and search the z/OS Communications Server knowledge base for Technotes, FAQs, white papers, and other z/OS Communications Server information

https://www.ibm.com/mysupport

IBM Communications Server performance information

This site contains links to the most recent Communications Server performance reports

http://www.ibm.com/support/docview.wss?uid=swg27005524

IBM Systems Center publications

Use this site to view and order Redbooks publications, Redpapers, and Technotes

http://www.redbooks.ibm.com/

z/OS Support Community

Search the z/OS Support Community Library for Techdocs (including Flashes, presentations, Technotes, FAQs, white papers, Customer Support Plans, and Skills Transfer information)

z/OS Support Community

Tivoli® NetView for z/OS

Use this site to view and download product documentation about Tivoli NetView for z/OS

http://www.ibm.com/support/knowledgecenter/SSZJDU/welcome

RFCs

Search for and view Request for Comments documents in this section of the Internet Engineering Task Force website, with links to the RFC repository and the IETF Working Groups web page

http://www.ietf.org/rfc.html

Internet drafts

View Internet-Drafts, which are working documents of the Internet Engineering Task Force (IETF) and other groups, in this section of the Internet Engineering Task Force website

http://www.ietf.org/ID.html

Information about web addresses can also be found in information APAR II11334.

Note: Any pointers in this publication to websites are provided for convenience only and do not serve as an endorsement of these websites.

DNS websites

For more information about DNS, see the following USENET news groups and mailing addresses:

USENET news groups
comp.protocols.dns.bind

BIND mailing lists
https://lists.isc.org/mailman/listinfo

BIND Users

- Subscribe by sending mail to bind-users-request@isc.org.
- Submit questions or answers to this forum by sending mail to bind-users@isc.org.

BIND 9 Users (This list might not be maintained indefinitely.)

- Subscribe by sending mail to bind9-users-request@isc.org.
- Submit questions or answers to this forum by sending mail to bind9-users@isc.org.

The z/OS Basic Skills Information Center

The z/OS Basic Skills Information Center is a web-based information resource intended to help users learn the basic concepts of z/OS, the operating system that runs most of the IBM mainframe computers in use today. The Information Center is designed to introduce a new generation of Information Technology professionals to basic concepts and help them prepare for a career as a z/OS professional, such as a z/OS systems programmer.

Specifically, the z/OS Basic Skills Information Center is intended to achieve the following objectives:

- Provide basic education and information about z/OS without charge
- Shorten the time it takes for people to become productive on the mainframe
- Make it easier for new people to learn z/OS

To access the z/OS Basic Skills Information Center, open your web browser to the following website, which is available to all users (no login required): https://www.ibm.com/support/knowledgecenter/zosbasics/com.ibm.zos.zbasics/homepage.html?cp=zosbasics

Summary of changes for SNA Programmer's LU 6.2 Guide

This document contains terminology, maintenance, and editorial changes, including changes to improve consistency and retrievability. Technical changes or additions to the text and illustrations for the current edition are indicated by a vertical line to the left of the change.

Summary of changes for z/OS 3.2

The following content is new, changed, or no longer included in z/OS 3.2.

New

The following content is new.

September 2025 release

- None.

Changed

The following content is changed.

September 2025 release

- None.

Deleted

The following content is deleted.

September 2025 release

- None.

Changes made in z/OS Communications Server 3.1

This information contains no technical change for this release.

Chapter 1. Understanding VTAM LU 6.2 application programs

About this chapter

VTAM supervises telecommunications activity in a Systems Network Architecture (SNA) network. VTAM helps direct data transfer between application programs and devices, such as terminals, and between different application programs.

This manual discusses data-transfer application programs that conform to the rules of LU type 6.2 protocols, also known as advanced program-to-program communication (APPC) protocols. LU 6.2 protocols and services are defined by SNA to support communication between type 6.2 logical units. (For a definition of logical units, see “Logical units” on page 1.)

Advantages of LU 6.2 application programs

VTAM LU 6.2 support enables programmers to write or modify host application program protocols with less programming effort, encouraging the use of peer-to-peer protocols throughout a network. This, in turn, enhances the processing power of smaller systems used in developing distributed processing application programs.

VTAM LU 6.2 application programs run in a host processor. These application programs communicate with other host programs and devices in the network that support LU 6.2 protocols. VTAM's support for host LU 6.2 application programs enhances the connectivity between the host and non-host points in the network.

A personal computer or AS/400, for example, cannot be a peer to a host application program without LU 6.2 support in VTAM or without its own implementation of the LU 6.2 architecture. Otherwise, it appears to the host as a terminal rather than as another application program. Many host application programs do not provide a complete LU 6.2 implementation. In such cases, non-host systems must emulate a terminal to communicate with the host programs that does not take full advantage of the processing power available in the nonhost system.

The role of LU 6.2 in SNA networks

LU 6.2 application programs are an important part of an SNA network. Their function in the network can be understood by comparing them to other logical units and the session types associated with these logical units.

Logical units

A logical unit is a device or application program by which an end user (an application program, a terminal user, or an input/output mechanism) gains access to a SNA network. Any device or application program that implements LU 6.2 protocols appears as an LU 6.2 in the network. All application programs using SNA sessions, including LU 6.2 application programs, are considered logical units by SNA. To the network, a logical unit is the source of a request coming into the network, although the logical unit might not be the original source. The contents of the request or the information on which the request is based might have originated at a device controlled by the logical unit.

For example, in a 4702 Finance Controller, the logical unit is an application program that handles input and output for one or several finance terminals attached to the controller. Input actually originates at one of the terminals, but the logical unit (the application program) in the 4702 uses the input to create and transmit the request. Similarly, the network sees a logical unit as the destination of a request, but the logical unit might actually pass the data to a device for recording, printing, or displaying.

A VTAM application program is also a logical unit. VTAM sees an application program as an originator of and destination for requests. Other programs in the host processor can interact with a VTAM application program and use it to communicate with other parts of the network. In such cases, these programs are the application LU's end users.

SNA categorizes LUs into different types depending on the set of SNA functions the LU supports. SNA defines LU types of 1, 2, 3, 4, 6.1, 6.2, and 7. SNA does not define an LU type of 0. However, type 0 is used to describe communication between LUs that do not conform to SNA protocols.

Figure 2 on page 2 shows an SNA network composed of type 6.2 LUs.

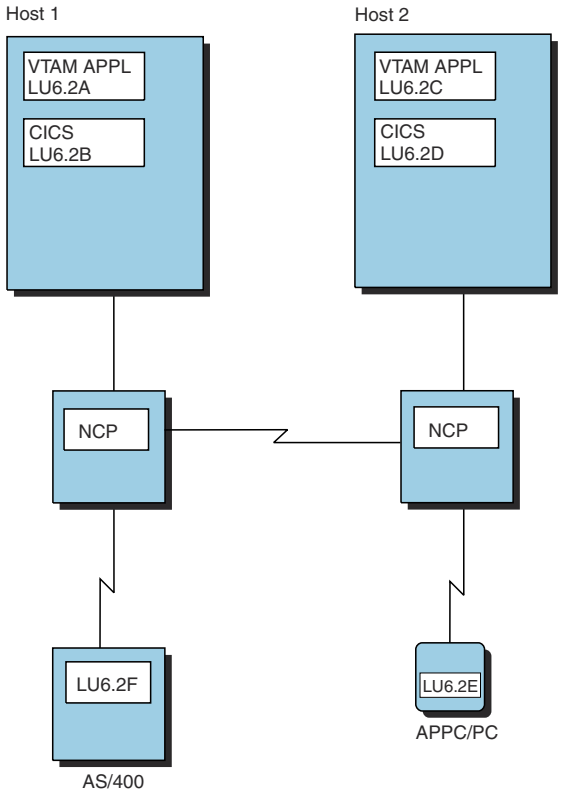


Figure 2. SNA network of LU 6.2s

Session types

When two logical units in the network want to exchange data, VTAM establishes a communication connection between the two LUs. The logical connection between the LUs is called a session in SNA. Figure 3 on page 2 shows APPL1, running on LU 6.2A, in session with APPL2, running on LU 6.2C.

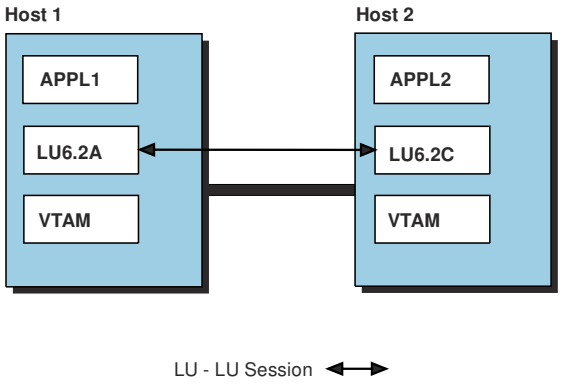


Figure 3. Session between APPL1 and APPL2

When two LUs communicate over a session, the protocols used on the session correspond to the type of LUs communicating. Consequently, protocols on the session can be described as an LU session type.

The following list gives the characteristics of the LU session types. LU types 1 through 4 and LU type 7 describe a host program supporting a device with limited function. LU types 6.1 and 6.2 describe program-to-program protocols.

LU 0
: These protocols are not defined by SNA. The LUs that are communicating must implement a set of protocols. For example, the protocols that VTAM uses to support non-SNA 3270 binary synchronous communication (BSC) terminals are LU 0 protocols.

LU 1
: LU 1 protocols provide access to nondisplay I/O devices such as printers and keyboard printer terminals.

LU 2
: LU 2 protocols provide access to display terminals with the IBM 3270 data stream.

LU 3
: LU 3 protocols provide access to printers with a subset of the IBM 3270 data stream.

LU 4
: LU 4 protocols provide access to terminals that are similar to LU 1 terminals but have more functions.

LU 6.1
: LU 6.1 protocols provide communication access to other application programs. These are the original protocols SNA defined for communication between application programs. Only application programs in a host processor can use LU 6.1 protocols.

LU 6.2
: LU 6.2 protocols support sessions between two application programs in a distributed data processing environment. LU 6.2 provides a connection between its transaction programs and network resources. These protocols can be implemented by application programs running on non-host systems and by programmable hardware devices in the network. (This manual, however, is concerned only with LU 6.2 application programs that are in the host and that can issue VTAM APPCCMD macroinstructions.)

LU 7
: LU 7 protocols provide access to a single display station.

VTAM application programs can function as many different LU types simultaneously. For example, when an application program exchanges data with a terminal that supports only LU type 2, the session between the two logical units uses only LU type 2 protocols; during that session, the application program is an LU type 2. When the application program exchanges data with another application program and LU protocols of 6.1 or 6.2 are used, the LU is one of those types during that session.

An application program can function as a type 6.2 LU and can still communicate as any of the other SNA LU types. It can have sessions with printers or terminals and still request LU 6.2 services from VTAM when it communicates with another LU 6.2 application program.

Refer to *SNA Technical Overview* for more details on SNA protocols for non-LU 6.2 application programs.

Important LU 6.2 concepts

The discussion of the VTAM LU 6.2 application programming interface (API) assumes that the reader understands certain concepts that are fundamental to LU 6.2 architecture. This section reviews those fundamental concepts. For more information, see the LU 6.2 architecture manuals: the *SNA Transaction Programmer's Reference Manual for LU Type 6.2* and the *SNA Format and Protocol Reference Manual: Architecture Logic for LU Type 6.2*.

Peer-to-peer protocol

LU 6.2 protocols control synchronous communication between peers. Both of the points communicating have similar communication processing capability. They might not have equal processing power, but they

are capable of fulfilling the basic communication functions of a type 6.2 LU. A PC, for example, does not have the processing capabilities of a mainframe.

LU 6.2 peers are program processes running on one or more computers. For example, a personal computer can use software such as Communications Server for OS/2 to appear to programs in the host as another application program instead of a terminal. LU 6.2 can also be implemented in microcode (or even in hardware) on devices such as line printers or terminals. The LU 6.2 protocol is equally applicable to line printers and to complicated, distributed database operations.

Throughout this book, the term application program is used to describe the LU 6.2 peers that are communicating. These peers can be anything capable of implementing the LU 6.2 protocol, including microcode and hardware logic.

LU protocol boundary

To accommodate LU 6.2 implementations on different processors and in different languages, SNA LU 6.2 architecture defines how application programs request LU 6.2 services in generic terms. This generic interface is called an LU protocol boundary. Protocol boundaries offered to user application programs are defined in terms of verbs.

For example, the function of sending data somewhere else in the network is defined in the LU 6.2 architecture by the verb SEND_DATA. The SNA definition for this verb further specifies the format the data must be in, what parameters can be supplied on the request, and what parameters can be returned to the application program. The APPCCMD CONTROL=SEND, QUALIFY=DATA macroinstruction conforms to this definition by allowing application programs to specify the input parameters required by the architecture and by returning the required output parameters.

Through LU 6.2 support, VTAM provides an API that enables its application programs to implement an LU protocol boundary. The application program takes advantage of the VTAM API by using the APPCCMD macroinstruction. The various forms of the APPCCMD macroinstruction offer the functions of verbs defined in the architecture.

By using these verb functions, an application program on one end of the network can initiate a transaction with another application program or device on the network, send and receive data as required by the transaction, and terminate the transaction. All of this can be done without the application program being aware of the configuration of the network or even of the SNA protocols used to establish connections and transmit data across the network.

VTAM offers its application programs both conversation verbs and control operator verbs. VTAM does not support all the conversation verbs defined in the architecture. However, it does support a large subset of the base set and many key optional layers. For more information on VTAM verbs, see “LU 6.2 verbs” on page 31.

Control operator verbs are used to request services to control the LU. These services include establishing limits for the number and types of sessions the LU can have.

Transaction programs

LU 6.2 treats a session as a relatively long-lived reusable connection between two LUs. Sessions can be compared to pipes through which data flows between the LUs.

LU 6.2 breaks down the data that is exchanged between two LUs into units called transactions. Transactions are the units of work done between the LUs.

For example, an interaction between an automatic teller machine and a host database program for a given customer could constitute a transaction. The interaction could involve several exchanges of data but all for the purpose of serving the customer using the teller machine. The following sequence could make up the transaction:

1. The customer inserts a card, alerting the teller machine to contact the host database program. The logical connection is established and the transaction begins. This transaction is on behalf of one customer. No exchange of data involving another customer can take place until this one is finished.

2. The customer requests a cash withdrawal. The teller machine reports the request to the host program, which checks the database, reports to the teller machine that the withdrawal can take place, and updates the database to reflect the new balance.
3. The teller machine dispenses the money. The customer then signals the teller machine that the transaction is being terminated. The teller machine reports this to the host program, and the logical connection that is set up to serve the customer is disconnected.

The teller machine and the host program would have many transactions each day. These transactions cannot be interrupted and are executed serially on each session. One transaction must finish before another one can use the session resource to exchange data.

Transactions are governed in a host by the VTAM application program involved. The application program supports an associated set of execution threads, called transaction programs. A subroutine, for example, might make up a transaction program. The application program issues VTAM macroinstructions to request LU 6.2 services for these processing threads.

Note: The application program must implement transaction programs.

Conversations

LUs are connected by sessions, and transaction programs are connected by conversations. A conversation is a logical connection between two transaction programs that uses a session between two LUs to transport data. The conversation can be thought of as a time slice of a session. One session can support only one conversation at a time, but one session can support many conversations in sequence. Figure 4 on page 5 shows a session between LU 6.2A and LU 6.2B and a conversation between TP1 and TP2.

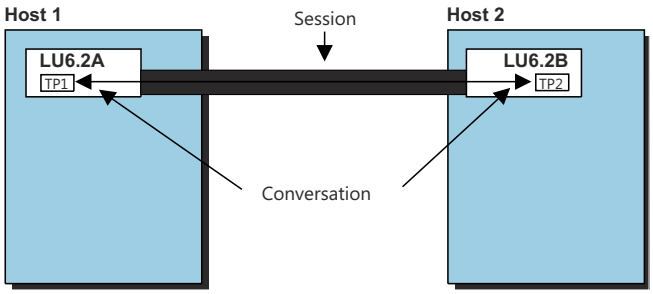


Figure 4. Conversation flowing on a session

When a transaction program requests a conversation, it uses the function of the ALLOCATE verb. The transaction program specifies the name of a partner LU and the session mode name, which identifies certain session characteristics. The application program also specifies a name that identifies the partner transaction program that is to be started. (This function is provided in VTAM by the APPCCMD macroinstruction with CONTROL=ALLOC.)

When a transaction program requests a conversation, VTAM allocates the session to the requested conversation if the following conditions are present:

- A session exists between the two LUs.
- The session has the specified mode name.
- The session is not being used for another conversation.

If a session is not available, VTAM starts a new session (if possible) using the specified mode name. This session is then used for the conversation.

After the conversation is initiated, the two transaction programs use LU 6.2 verb functions to send and receive data as necessary for the transaction. When the transaction is finished, the conversation is deallocated (ended) by using the DEALLOCATE verb. The session is now available for another conversation between transaction programs using the same LUs.

Conversation states

A conversation is the communication mechanism between two transaction programs. The conversation may appear different to each transaction program. Although the appearance of the conversation differs, the views are complementary. For example, one conversation partner on a half-duplex conversation would be receiving while the other partner is sending. The appearance of a conversation for a transaction program is described by the *state* in which the conversation exists for that transaction program. Whenever the term *conversation state* is used in this publication, it refers to the local view of the conversation.

Logical records and buffers

Data is transmitted across conversations in logical records. A logical record consists of a 2-byte length field followed by up to 32765 bytes of data. The value in the length field specifies the length of the data and the length field together.

The transaction program can receive data one logical record at a time or in one input/output buffer at a time. Both buffer size and logical record size are independent of the SNA RU size used on the session.

Full-duplex and half-duplex protocols

During BIND negotiation, each LU indicates its support for full-duplex or half-duplex conversations.

Full-duplex protocols

Full-duplex protocols allow an application program to send and receive information concurrently. This allows the application sending information to continue without waiting for a response from the partner LU. The partner LU can also send information without waiting for an appropriate response. Neither LU has to wait for the partner LU to surrender its right to receive or send information.

Full-duplex conversations are especially suited for transactions in which the data sent and received is either independent or indirectly related (for example, process control). This is advantageous for transactions in which an immediate response is desired. For example, sensor data (temperature, for example) is continuously received in a process control environment and corrective information is continuously sent. A full-duplex conversation allows the controlling application to detect an abnormal condition and respond to correct the condition immediately. Fewer system resources are used, and the integrity of the environment is maintained.

Because full-duplex conversations are not suited for tightly coupled transactions such as sync point and inquiry-reply transactions, sync point is not supported for full-duplex conversations.

Half-duplex protocols

Half-duplex protocols do not allow LUs to exchange information concurrently. Only one LU at a time can send or receive data. One LU is designated as the sender and the other LU as the receiver. The receiver can request permission to become the sender and can take over that role under certain error conditions, but in general, the receiver cannot send data until the sender surrenders that right.

Half-duplex conversations are well-suited to tightly coupled transactions, such as sync point or inquiry and reply transactions.

Single and parallel sessions

An LU might have more than one session at the same time with another LU. Such sessions are called parallel sessions. LUs that can support parallel sessions are parallel-session-capable LUs. When the first session is established between LUs, part of the session establishment process defines whether the partners are capable of parallel sessions. If either LU is not parallel-session capable, the session is said to be a single session. If both are parallel-session capable, the session is called a parallel session, even if it is the only one in existence at the time.

Parallel and single sessions use different protocols. Establishing a session as a single session restricts the LUs to only one session at a time. LUs capable of only a single session can support only one transaction

program at a time. LUs that are capable of parallel sessions can support more than one session at a time and more than one transaction program at a time. Figure 5 on page 7 shows a comparison of type 6.2 LUs capable of only a single session and type 6.2 LUs that are parallel-session capable.

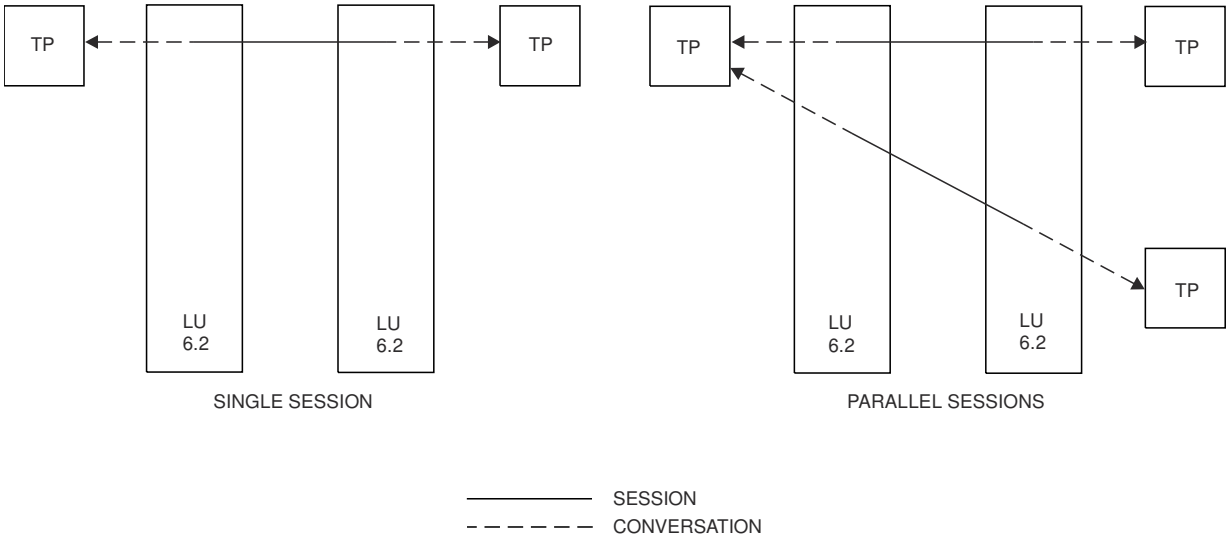


Figure 5. Comparison of single and parallel sessions

In parallel sessions, an LU can have more than one TP at a time, and a TP can have more than one session at a time.

LUs that are parallel-session-capable can also support conversations between transaction programs located at the same LU (option set 204).

Mode name groups

LU 6.2 associates each session with a set of characteristics for a type of device. This set of characteristics is called a *mode name*. A mode name defines characteristics such as pacing level and class of service. When transaction programs request a conversation, they usually do not specify which session to use for the conversation, but they can specify the mode name of the session. Each mode name is defined in the logon mode table, which describes the session parameters. For additional information on logon mode tables, refer to z/OS Communications Server: SNA Resource Definition Reference and z/OS Communications Server: SNA Programming.

Each mode name can be used by several sessions. In an LU type 6.2, these sessions form a group that is treated as a pool of sessions, all of which share the same mode name characteristics. An application program can control the size of a pool, but VTAM handles the individual sessions that make up the pool. You cannot specify pool size for non-LU 6.2 sessions.

LUs can define a number of different mode name groups for sessions with another LU. For example, an LU might have a FILESERV mode name defined for sessions with a partner LU used by file-server programs. FILESERV denotes sessions with a large request unit (RU) size, which aids bulk transmission of data. This LU also might have other mode names, such as INTERACT, defined for sessions with the same LU used for database queries.

By using different mode groups, you can avoid the possibility of database queries getting backed up behind mass file-server requests. To determine whether different mode name groups are beneficial, consider the following factors:

- Importance of the data exchange
- Nature of the data exchange
- Desired response time

As part of its support for LU 6.2, VTAM maintains a data structure for the application program that:

- Lists possible partner LUs

- Determines valid names for the partner LU
- Associates valid mode names defined for each partner LU
- Contains information about the mode names

This data structure is the LU-mode table. It is unique in VTAM to LU 6.2 support. For more information on the LU-mode table, see "Data structures" on page 43.

Some SNA-defined modes are used for special purposes. For example, the SNASVCMG mode is used by VTAM to exchange control information. Application programs should not normally use SNASVCMG sessions for a conversation. The CPSVCMG mode is defined but it is also reserved.

Session contention

Two LUs in different hosts can request a conversation at the same time, and the VTAM in each host can choose the same session for both conversations. Such a situation is called *contention*. For a specific session, one of the partner LUs is the contention winner and the other partner LU is the contention loser. Contention winner versus contention loser is decided during BIND negotiation. The choice of contention winner versus contention loser is primarily due to information learned during the change number of sessions (CNOS) negotiation.

A contention loser can request use of the session from the contention winner. For VTAM-supported LU 6.2 application programs, VTAM requests use of sessions on the contention-loser side and grants or denies use on the contention-winner side.

LUs capable of parallel sessions can divide the contention-winner role between them for their sessions. For example, if two LUs have five parallel sessions, one LU might be designated the contention winner for three sessions and the other LU the contention winner for two sessions. The number of contention winners can be established at system definition time or dynamically. VTAM-supported LU 6.2 application programs are involved in setting the numbers of contention-winner and contention-loser sessions.

Session limits

The LU 6.2 architecture does not permit an unlimited number of sessions between parallel-session-capable LUs. Limits are imposed on the number of sessions an LU can have for a given mode name. Limits also are imposed on the number of contention-winner sessions an LU can have for a given mode name. These limits are called *session limits*. Before a session can be activated between two LUs on a given mode, the session limits on that mode must be negotiated between the two LUs.

Because each mode name has its own session limits, LUs have multiple session limits. An application program might be limited, for example, to five sessions with another LU using a mode name of EXAMPLE and be able to have 10 sessions with the same LU using a mode name of TESTCASE. Session limits can be defined dynamically while the application programs are executing. LU 6.2 architecture defines control operator verbs that can be used to change session limits.

VTAM-supported LU 6.2 application programs

VTAM-supported LU 6.2 application programs have APPC=YES coded on their APPL definition statements. (For details on defining an application program to VTAM, refer to z/OS Communications Server: SNA Resource Definition Reference.) These LU 6.2 application programs can issue the VTAM APPCCMD macroinstruction, which provides LU 6.2 services. They also can support non-LU 6.2 sessions by using the macroinstructions provided with the record API. The set of assembler-language macroinstructions VTAM offers to its application programs is called the *application program interface* (API). The application program uses the API by issuing the macroinstructions. Once the system programmer defines the application program to VTAM, the application program is authorized to issue these types of macroinstructions.

LU 6.2 support is a specialized enhancement to the interface that VTAM offers to all its application programs.

In general, an application program uses the VTAM API by:

- Issuing VTAM macroinstructions to request VTAM services.
- Manipulating storage that contains control information that VTAM needs to process macroinstructions.
- Providing exit routines for VTAM to schedule when a request completes or when an external event requires the application program's attention. (The exit routines are optional, but some are highly recommended.)

For a detailed explanation of the VTAM record API, and of concepts pertaining to all VTAM application programs, refer to z/OS Communications Server: SNA Programming.

Responsibilities for implementing LU 6.2

VTAM and the application program share responsibility for implementing a type 6.2 LU.

VTAM is responsible for the following actions:

- Providing a higher-level API that assists an application program in building an LU protocol boundary, as described in the LU 6.2 architecture. This protocol boundary, largely implemented through the APPCCMD macroinstruction, enables application programs to request LU 6.2 services.
- Associating network communication resources with the application program.
- Terminating LU 6.2 communication services with an application program's partners if the application program ends prematurely.

The application program is responsible for the following actions:

- Implementing the LU 6.2 concept of transaction programs
- Using VTAM's APPCCMD macroinstruction to perform LU 6.2 functions
- Maintaining internal resources needed to support a type 6.2 LU

Figure 6 on page 9 shows that VTAM and the application program share the responsibility for implementing a type 6.2 LU.

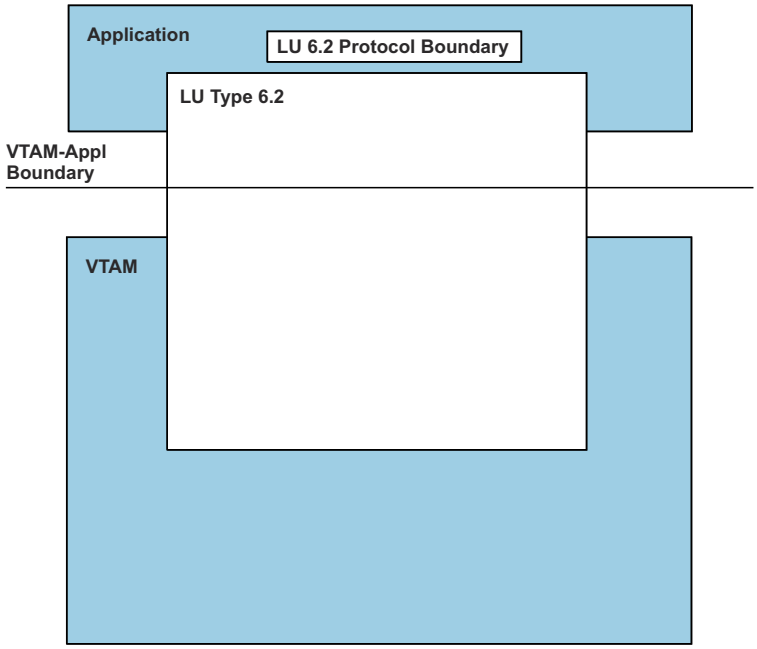


Figure 6. VTAM and application program implementation responsibilities

Chapter 2. LU 6.2 and the VTAM API

About this chapter

VTAM application programs can support LU 6.2 and non-LU 6.2 session types. This chapter discusses the interface VTAM offers for LU 6.2 services and compares it to the interface offered for non-LU 6.2 session types.

Application programs that use VTAM's implementation of LU 6.2 must have APPL definition statements coded with APPC=YES and must let VTAM establish and terminate their LU 6.2 sessions. The application programs manage conversations and implement transaction programs.

The discussion of the similarities and differences of the VTAM LU 6.2 interface covers features that all VTAM application programs use and features that only LU 6.2 application programs use.

Many of the concepts of the interface common to both LU 6.2 functions and other session types are documented in z/OS Communications Server: SNA Programming. This book discusses them briefly, but the full explanation and details are found in z/OS Communications Server: SNA Programming.

Standard features of the API

All VTAM application programs perform some common tasks regardless of the types of sessions being used. Even if an application program uses LU 6.2 services exclusively, it does much of the same processing as a program using other session types.

All VTAM application programs do the following actions:

- Use the VTAM macroinstruction language to request VTAM services
- Use control blocks to identify themselves and their requests to VTAM
- Use many of the same macroinstructions
- Have access to vector lists that describe the level of VTAM on the system and that specify resource identifiers that otherwise might be unknown to the application program
- Provide VTAM with a list of special-purpose exit routines to handle external events
- Use general-purpose registers
- Use the same operating system environment

Unique LU 6.2 features of the API

Although similarities exist in the way LU 6.2 and other application programs use the VTAM API, some features of the interface are unique to LU 6.2 support. The primary differences are in how the LU 6.2 interface treats sessions and how the LU 6.2 application program uses conversations. VTAM initiates and terminates LU 6.2 sessions. The LU 6.2 application programs initiate and terminate conversations. Some other differences follow.

- The following VTAM macroinstructions are normally used to establish and terminate sessions by applications whose APPL statement specifies APPC=NO. These macroinstructions are not valid for use with LU 6.2 sessions in LU 6.2 applications whose APPL statement specifies APPC=YES:
  - OPNDST
  - CLSDST
  - TERMSESS
  - OPNSEC
- The LU 6.2 application program is not concerned with the node initialization block (NIB), a control block used by non-LU 6.2 applications in session initiation and termination.

- VTAM handles many SNA request units for the application program for LU 6.2 sessions. This means that:
  - Network services request units for LU 6.2 sessions are VTAM's responsibility. Therefore, LU 6.2 application programs do not need an NSEXIT exit routine for LU 6.2 sessions. However, if an LU 6.2 APPL initiates a non-LU 6.2 session (that is, REQSESS) and a failure prevents a BIND, notification could be received in the NSEXIT.
  - Most of the conditions that cause application program SYNAD exits to be scheduled are instead handled by VTAM for LU 6.2 sessions.
  - Only the CONTROL=BIND option is valid on the SESSIONC macroinstruction for LU 6.2 sessions.
  - APPCCMD SEND and RECEIVE macroinstructions are simpler than their counterparts for non-LU 6.2 sessions. The options and RPL fields required to process SNA RUs, such as data-flow-control requests, do not concern the application program.

Other features of the LU 6.2 interface are:

- Additional control blocks and fields in the RPL
- A special-purpose exit, the ATTN exit, for LU 6.2 support
- Finite state machine concepts to restrict the varieties of the APPCCMD macroinstruction that an application program can issue at any given time

VTAM as session manager

To ensure that application programs defined with APPC=YES do not perform their own session management for LU 6.2 sessions, VTAM monitors the use of the session-establishment macroinstructions used to establish non-LU 6.2 sessions and rejects any that would cause such an LU 6.2 session.

VTAM manages LU 6.2 sessions *only* for those application programs that have APPC=YES coded on their APPL definition statement. However, application programs can implement LU 6.2 support entirely on their own. When they do, they do *not* code APPC=YES on their APPL definition statement. These application programs establish and terminate sessions according to the rules of LU 6.2 architecture, but do so using the application program macroinstructions described in z/OS Communications Server: SNA Programming.

VTAM-supported LU 6.2 application programs can communicate with application programs that implement LU 6.2 on their own. In such a case, VTAM still establishes and terminates sessions for the application program that is using VTAM support.

The LU 6.2 application program controls the session limits for a given mode name for sessions between itself and a partner LU. An application program can indirectly start and end sessions by asking VTAM to allocate a conversation or change the session limits between itself and another LU. VTAM handles the details of starting or ending the sessions.

VTAM macroinstruction language

VTAM macroinstructions are assembler macroinstructions and follow the same rules. (Refer to z/OS Communications Server: SNA Programmer's LU 6.2 Reference for details on coding rules for these macroinstructions.) With only a few exceptions, the operands on VTAM and VTAM APPCCMD macroinstructions can be entered in any order.

The four categories of VTAM macroinstructions are:

Declarative
These macroinstructions build control blocks during assembly.

RPL-Based
These macroinstructions use the RPL to describe requests to VTAM. Most of the macroinstructions this manual discusses are RPL-based, including APPCCMD.

ACB-Based
These macroinstructions tell VTAM that the application program is beginning or ending its use of VTAM services. They are required for VTAM to recognize the application program as active.

Manipulative
These macroinstructions build and manipulate control block values during application program execution. They do not support the LU 6.2-unique control block fields, but they do support some control block fields that are common to both LU 6.2 and other application programs.

Control blocks and mappings

To use VTAM macroinstructions, the application program must reserve and manipulate storage for the control information that VTAM needs to process the application program's request. These storage areas are called *control blocks*. (Some of the VTAM macroinstructions can be used to build the control blocks and initialize them to appropriate values. In addition, IBM supplies assembler-language DSECTs with VTAM that help the application program manipulate control block storage fields.)

When an application program requests VTAM services, it must ensure that the appropriate control block fields are initialized. In many cases, the application program can do so with parameters on the VTAM macroinstruction. When VTAM finishes processing a macroinstruction request, it passes back completion information to the application program in these control blocks and in general-purpose registers.

Common control blocks and mappings

Some control blocks can be used by both LU 6.2 and non-LU 6.2 application programs. The manipulative macroinstructions VTAM provides to create control blocks and gain access to control block fields are only of limited use for LU 6.2 functions. They do not support the unique LU 6.2 control blocks. IBM does, however, supply DSECTs that enable application programs to refer to control block fields symbolically.

Access method control block

Application programs must use an access method control block (ACB) to identify themselves to VTAM. The ACB points to a location in the program that contains the name of the application program as specified in an APPL definition statement during VTAM definition. It can also point to an EXLST control block containing the addresses of exit routines that are to be associated with the application program.

The application program name pointed to by the ACB corresponds to the LU name that another VTAM application program would specify if requesting a conversation with one of the application program's transaction programs.

The application program must create the ACB and initialize the fields within it. VTAM provides a macroinstruction, ACB, that allows application programs to build ACBs during program assembly.

Opening an ACB

The application program uses the OPEN macroinstruction, which points to an ACB, to formally present itself to VTAM. After this macroinstruction completes successfully, the application program can request VTAM services.

More than one ACB can be opened by an application program. This means that an application program that performs more than one function (for example, communicating with logical units and acting as a program operator application program) can be defined so that it is viewed by VTAM as being more than one application program. However, most VTAM users will find it satisfactory to open only one ACB for each application program.

Closing an ACB

To stop using VTAM services, an application program uses the CLOSE macroinstruction. The CLOSE macroinstruction terminates all sessions that VTAM started for the application program and halts communication between VTAM and the application program. As far as VTAM is concerned, the application program becomes totally inactive. The application program can continue to perform other processing not related to VTAM, however.

For more details on ACBs and the OPEN and CLOSE processes, refer to z/OS Communications Server: SNA Programming.

RPL control block and APPC extension

Application programs request LU 6.2 services from VTAM with the APPCCMD macroinstruction. The APPCCMD macroinstruction uses a request parameter list (RPL) to describe its requests to VTAM and to provide necessary processing information. The RPL is a control block that LU 6.2 application programs use frequently. It must be specified on each APPCCMD or other macroinstruction issued by the application to request VTAM services.

Application programs can explicitly set RPL fields themselves and specify only the address of the RPL on the VTAM macroinstruction, or they can use operands on the macroinstruction that cause VTAM to set the relevant RPL fields for a particular operation. As with any RPL-based request, the application program can specify an exit routine to receive control when the operation completes.

For APPCCMD requests, an RPL extension is used with the RPL. An RPL APPC extension supports only LU 6.2 functions. This extension is not appended to the RPL but is pointed to by the AAREA field in the RPL. In effect, the RPL for APPCCMD macroinstructions consists of a linked data structure. VTAM supplies the macroinstruction ISTRPL6 to create this extension and the DSECT ISTRPL6X to enable the application program to refer to its fields symbolically.

The following keyword operands for the RPL are valid for APPCCMD macroinstructions:

- ACB
- AAREA
- AREA
- AREALEN
- BRANCH
- ECB
- EXIT
- OPTCD
- RECLEN

All other information in the RPL that pertains to LU 6.2 functions is in the extension. The length of the extension is found in AAREALN. The length is determined and the length field properly initialized by the APPCCMD macroinstruction. The application program does not need to modify this field.

BIND request unit (RU) mapping

The BIND includes numerous session parameters relevant to LU 6.2 support. For details, see “BIND image and response” on page 124.

LU 6.2 control blocks and mappings

Two of the control blocks deal with session limits. The CNOS session limits control block is used as an interface for the APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS macroinstruction. The DEFINE/DISPLAY session limits control block is used as an interface to the LU-mode table for the APPCCMD CONTROL=OPRCNTL, QUALIFY=DEFINE and APPCCMD CONTROL=OPRCNTL, QUALIFY=DISPLAY macroinstructions.

A mapping of the function management header type 5 (FMH-5) is used in conversation allocation. It is used on the APPCCMD CONTROL=ALLOC, APPCCMD CONTROL=RCVFMH5, and the APPCCMD CONTROL=SENDFMH5 macroinstructions.

The TESTSTAT control block contains information passed to the application about the status of one or more active conversations.

The RESTORE control block contains information that describes the LUs, modes, and sessions that are being restored for a VTAM application program that is recovering after a previous failure. For information on the:

- Recovery process, see “Restoring modes and any associated persistent LU-LU sessions” on page 49.

- RESTORE control block, see “Retrieving information for a mode and sessions to be restored” on page 147.
- Retrieval of information from the RESTORE control block, see Appendix D, “Example of retrieving information for a mode and any restored sessions,” on page 329.

Common macroinstructions

Every VTAM application program must use some non-APPCCMD macroinstructions, even if it uses LU 6.2 services exclusively. In addition to these required macroinstructions, an LU 6.2 application program might find a number of other non-APPCCMD macroinstructions useful.

Macroinstructions required for requesting LU 6.2 services

The following non-APPCCMD macroinstructions establish the environment for using the APPCCMD macroinstructions:

- OPEN
- CLOSE
- SETLOGON

OPEN and CLOSE were discussed earlier. (See “Opening an ACB” on page 13 and “Closing an ACB” on page 13.) The SETLOGON macroinstruction must be issued so VTAM can begin accepting session-initiation requests on behalf of the application program.

These three macroinstructions are all that an application program must use in addition to the APPCCMD macroinstruction to request LU 6.2 services from VTAM. A complete description of these macroinstructions, including return code information, is found in z/OS Communications Server: SNA Programming.

Macroinstructions for building non-LU 6.2 control blocks

Application programs can use VTAM-supplied macroinstructions to build non-LU 6.2 control blocks such as the ACB or EXLST. Application programs have many options for using macroinstructions; therefore, no macroinstructions are listed as required. VTAM supplies macroinstructions that build control blocks at assembly time and those that allow control blocks to be built dynamically. In addition, VTAM supplies DSECT macroinstructions that enable the application program to obtain storage for a control block and use the DSECT to map the fields in the control block.

The macroinstructions that an application program can use to build and manipulate non-LU 6.2 control blocks and their fields are:

- ACB
- EXLST
- GENCB
- MODCB
- RPL
- SHOWCB
- TESTCB

The DSECT macroinstructions that an application program can use to map control block storage for non-LU 6.2 control blocks are:

- IFGACB
- IFGEXLST
- IFGRPL
- ISTNRIPL

- ISTBLENT
- ISTDBIND
- ISTUSFBC

IBM also supplies other DSECT macroinstructions, but these map control blocks are not used for LU 6.2 functions. Four of the above macroinstructions (the manipulative macrosinstructions GENCB, MODCB, SHOWCB and TESTCB) are used to create control blocks dynamically and test and set values in the control block fields. These can be used only for control blocks that are not exclusively for LU 6.2 use, such as the ACB, RPL, and EXLST. The manipulative macroinstructions do not support LU 6.2-unique control blocks such as the RPL extension for LU 6.2 or the session limits control blocks.

The declarative macroinstructions (ACB, RPL, and EXLST) build control blocks at assembly time. VTAM supplies an LU 6.2-unique declarative macroinstruction, ISTRPL6, to build the RPL extension. The other LU 6.2-unique control blocks, such as session limits control blocks, can be built from storage supplied by the application program. VTAM supplies DSECTs that enable the application program to map the storage. In addition, the APPCCMD macroinstruction includes operands that can set many of the RPL and RPL extension fields that must be initialized.

Non-APPCCMD VTAM macroinstructions

The following list summarizes the non-APPCCMD VTAM macroinstructions and indicates whether an application program can use them for VTAM-supported LU 6.2 sessions.

Note: These comments and restrictions apply only to those application programs that use VTAM LU 6.2 support by including APPC=YES on their APPL definition statements. They do not apply to application programs that provide their own LU 6.2 implementations.

ACB
This macroinstruction can be used to build the ACB. LU 6.2 application programs must specify MACRF=LOGON on the ACB macroinstruction, even if they do not contain a LOGON or SCIP exit. This allows VTAM to activate sessions for the application program's conversations. MACRF=NLOGON prevents VTAM from activating any sessions or from responding to session-initiation requests from partner LUs.

CHANGE
This macroinstruction can be used by an LU 6.2 application program acting as a generic resource to terminate an association with a partner LU.

CHECK
This macroinstruction cannot be issued for an RPL that specified an APPCCMD macroinstruction. APPCCMD CONTROL=CHECK must be used to check the completion status of an APPCCMD request.

CLOSE
This macroinstruction is unaffected by LU 6.2 support. LU 6.2 application programs should use this macroinstruction to terminate use of VTAM services.

CLSDST
The function of this macroinstruction depends on whether the application program specifies that a particular session or a group of sessions is to be terminated. If the application program specifies the secondary logical unit (SLU) name by means of the NIBSYM field and does not specify the communication identifier (CID) of a session (NIBCID=0), only non-LU 6.2 sessions are terminated. If the application program specifies the CID of an LU 6.2 session, the CLSDST is rejected.

The APPCCMD CONTROL=DEALLOC macroinstruction must be used in conjunction with APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS to free and then terminate active LU 6.2 sessions in an orderly manner. The application program can use the APPCCMD CONTROL=REJECT macroinstruction to abnormally terminate sessions and their conversations.

If the application has specified a SCIP exit when this exit is scheduled, the application may choose to terminate the pending LU 6.2 sessions by issuing the APPCCMD CONTROL=OPRCNTL, QUALIFY=DACTSESS macroinstruction.

EXECRPL
LU 6.2 application programs can use this macroinstruction to reissue an APPCCMD that is rejected because of a temporary condition. The temporary condition is marked by return codes in RCPRI and RCSEC that indicate a condition that can be retried.

EXLST
Application programs can use this macroinstruction to build the EXLST exit list. If they are providing an exit list, LU 6.2 application programs need to include the ATTN exit in this list.

GENCB
This macroinstruction is unaffected by LU 6.2 support. This macroinstruction cannot be used to create LU 6.2-unique control blocks, such as the RPL extension.

INQUIRE
This macroinstruction is unaffected by LU 6.2 support.

INTRPRET
This macroinstruction is unaffected by LU 6.2 support.

MODCB
This macroinstruction is unaffected by LU 6.2 support. This macroinstruction cannot be used to modify LU 6.2-unique control blocks, such as the RPL extension.

NIB
This macroinstruction is not used by application programs for LU 6.2 session establishment. However, NIB can be used by LU 6.2 application programs to specify a generic resource name with which it will identify itself to the network.

OPEN
LU 6.2 application programs must use this macroinstruction to obtain the use of VTAM services.

OPNDST
The application program cannot use this macroinstruction to establish an LU 6.2 session. Any attempt to do so results in the rejection of the OPNDST. In addition, this macroinstruction cannot be used to accept a pending CINIT that is initiated by VTAM. The following list describes the preceding two points in more detail:

- An OPNDST that specifies a BIND image (using the BNDAREA field of a NIB) is rejected if the BIND image contains an LU type of 6.2. For OPNDST ACCEPT, the CINIT is not rejected; only the macroinstruction is rejected. The application program must issue APPCCMD CONTROL=OPRCNTL, QUALIFY=ACTSESS or APPCCMD CONTROL=OPRCNTL, QUALIFY=DACTSESS to accept or reject the CINIT.
- An OPNDST that specifies a set of default session parameters or a logon mode name (using the LOGMODE field of a NIB) that resolves to a set of session parameters containing an LU type of 6.2 is rejected. For OPNDST ACCEPT, the CINIT is also rejected.
- An OPNDST ACCEPT SPEC by CID that would result in a LU 6.2 BIND is rejected. The CINIT is not rejected. The application program must respond to the CINIT by using APPCCMD CONTROL=OPRCNTL, QUALIFY=ACTSESS or APPCCMD CONTROL=OPRCNTL, QUALIFY=DACTSESS.
- An OPNDST RESTORE by CID that specifies an LU 6.2 session is rejected. The application program must restore the LU 6.2 session using APPCCMD CONTROL=OPRCNTL, QUALIFY=RESTORE.

If the application has specified a LOGON exit, then when this exit is scheduled, the application may choose to accept the pending session by issuing the APPCCMD CONTROL=OPRCNTL, QUALIFY=ACTSESS macroinstruction.

OPNSEC
The application program cannot use this macroinstruction to establish an LU 6.2 session. If it attempts to do so, both the OPNSEC and the specified BIND request are rejected. The APPCCMD CONTROL=OPRCNTL, QUALIFY=ACTSESS macroinstruction must be issued to accept BIND requests from partner LUs for LU 6.2 sessions under VTAM's control.

RCVCMD
This macroinstruction is unaffected by LU 6.2 support.

RECEIVE
OPTCD=SPEC cannot be issued for an LU 6.2 session. If this occurs, the request is rejected. OPTCD=ANY is unaffected by LU 6.2 support. It can receive data for non-LU 6.2 sessions, but it cannot be used to receive data for an LU 6.2 session.

REQSESS
This macroinstruction is unaffected by LU 6.2 support.

RESETSR
This request cannot be issued for an LU 6.2 session. If this occurs, the request is rejected.

RPL
Application programs can use this macroinstruction to build the RPL used by the APPCCMD macroinstructions. The AAREA field must be set to point to an extension for LU 6.2 services. The extension can be built with the LU 6.2-unique ISTRPL6 macroinstruction.

SEND
This request cannot be issued for an LU 6.2 session. If this occurs, the request is rejected.

SENDCMD
This macroinstruction is unaffected by LU 6.2 support.

SESSIONC
CONTROL=CLEAR, CONTROL=RQR, CONTROL=SDT, and CONTROL=STSN cannot be issued for an LU 6.2 session. If this occurs, the request is rejected. CONTROL=BIND still rejects a bind for an LU 6.2 session.

SETLOGON
This macroinstruction is used to control the scheduling of the LOGON and SCIP exits. It is not affected by LU 6.2 support. An application program must issue SETLOGON OPTCD=START before VTAM can begin to establish sessions for the application program, even if the application program does not provide a LOGON or SCIP exit. This macroinstruction can also be used to enable or disable an application's support for persistent sessions, or to create or delete an association between an application's network name and a generic resource name specified on the NIB macroinstruction.

SHOWCB
This macroinstruction is unaffected by LU 6.2 support. This macroinstruction cannot be used to access information in LU 6.2-unique control blocks, such as the RPL extension.

SIMLOGON
This macroinstruction cannot be used to initiate an LU 6.2 session by an application that uses LU 6.2 support.

TERMSESS
The function of this macroinstruction depends on whether the application program specifies that a particular session or group of sessions is to be terminated. If the application program specifies the PLU name by means of the NIBSYM field and does not specify the CID of a session (NIBCID=0), only non-LU 6.2 sessions are terminated. If the application program specifies the CID of an LU 6.2 session, the TERMSESS is rejected.

TESTCB
This macroinstruction is unaffected by LU 6.2 support. This macroinstruction cannot be used to test LU 6.2-unique control blocks, such as the RPL extension.

Session limits and CNOS commands

Although application programs cannot explicitly establish and terminate LU 6.2 sessions, they do take actions that result in sessions being established or terminated. Most of these actions involve controlling the session limits between an application program and a partner LU for a given mode name.

An application program uses the APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS macroinstruction to set new session limits for sessions between itself and another LU using a specified mode name. These new session limits can cause VTAM to activate or deactivate sessions for the application program. However, the application program has no control over which particular session is activated or deactivated.

The session limits apply to a given mode name. An application program can set new session limits with a partner LU for a mode name of EXAMPLE and can leave session limits unchanged for other mode names for the same LU. Session limits for a mode must be negotiated before VTAM can activate a session on that mode.

See Chapter 6, “Managing sessions,” on page 83 for a discussion of the interaction of session limits, the CNOS macroinstruction, and session activation and deactivation.

VTAM conversations

Although application programs do not control LU 6.2 sessions, they do explicitly start and stop conversations. The macroinstructions used to do this are APPCCMD CONTROL=ALLOC and APPCCMD CONTROL=DEALLOC. Allocation and deallocation can cause VTAM to activate or deactivate sessions between the application program and its partner LU, depending on the session limits and number of sessions in use. (See Chapter 6, “Managing sessions,” on page 83 for a detailed description.)

The way application programs use conversations is analogous to the way they use non-LU 6.2 sessions. When conversations are allocated, VTAM assigns an identifier that application programs can find in the CONVID field in the RPL extension. (When the APPCCMD CONTROL=ALLOC or CONTROL=RCVFMH5 macroinstructions complete successfully, CONVID is returned.) The application programs can use the CONVID value to specify particular conversations on APPCCMD macroinstructions.

The application program uses the APPCCMD macroinstruction to control a particular conversation. For example, macroinstructions used to send data or deallocate a conversation all specify a particular conversation. Application programs can request that VTAM do the following for a particular conversation:

- Send or receive data and control information over the conversation.
- Deallocate a conversation.
- Change the role of one of the conversation partners on a half-duplex conversation from sender to receiver.
- Change how data is received from a partner.
- Return status information about an active conversation to the application.

All of these options are discussed in detail in Chapter 6, “Managing sessions,” on page 83 through Chapter 11, “Sending and receiving data using high performance data transfer,” on page 217. For basic information about conversations, see “Conversations” on page 5.

Conversation states

In supporting LU 6.2, VTAM uses the concept of finite state machines to describe the condition of one side of a conversation at any given time. The LU 6.2 architecture defines what transaction programs can do in any given state. Consequently, application programs are restricted to using a subset of the APPCCMD macroinstructions at any given time. For example, an application program that has been asked to respond to a confirmation request must use one of the APPCCMDs appropriate to a confirmation response. VTAM rejects any other macroinstructions. For more information on conversation states, see “Maintaining conversation states” on page 55.

LU 6.2 global variables

Application programs can use a set of global variables to verify the LU 6.2 options that VTAM supports. The application program requests VTAM to initialize the variables by using one of the following macroinstructions:

- The VTAM macroinstruction ISTGAPPC
- The macroinstruction IFGRPL AM=VTAM
- The macroinstruction IFGACB AM=VTAM

For a list of the global variables and their meanings, refer to z/OS Communications Server: SNA Programmer's LU 6.2 Reference.

The initialization of variables is done during program assembly, not during program execution. To have the indicators show the support for new functions that are in VTAM, you must reassemble the application program. Each global variable is an arithmetic symbol that can be set to 0, 1, or 2. The meanings of the numbers are:

X'00'
Option is not supported.

X'01'
Option is supported.

X'02'
Option is not provided by VTAM but can be implemented by the application program as a pass-through function of VTAM.

Vector lists

Vector lists provide an open-ended interface for application programs to exchange information with VTAM. Each vector list contains a 2-byte length field followed by a set of contiguous vectors in random order, as shown in Figure 7 on page 20.

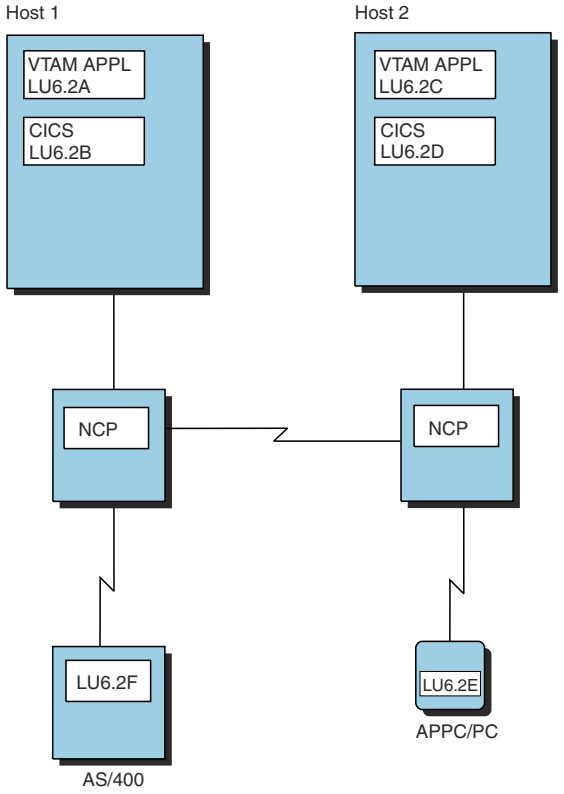


Figure 7. Format of a vector list

Each vector in the vector list is a set of contiguous data containing a length field, an identifier, and a value field containing vector data. The fields containing vector data can have trailing blanks. Some vector lists are built before or during OPEN processing, while others are built during processing of the APPCCMD macroinstruction. Table 1 on page 21 shows the vector lists available to VTAM applications, the macroinstruction with which they are associated, and their function.

Table 1. Vector lists

Name	Pointer	Macroinstruction	DSECT	Purpose	For More Information
Access-Method-Support	ACBAMSVL field in the ACB	OPEN ACB	ISTAMSVL	Supplies information to the application about the VTAM that opened the ACB	"Access-method-support vector list" on page 23
Resource-Information	ACBRIVL field in the ACB	OPEN ACB	ISTRIVL	Supplies information to the application about the resources available to the application	"Resource-information vector list" on page 24
Application-ACB	ACBAVPTR field in the ACB	OPEN ACB	ISTVACBV	Supplies information to VTAM about the application's capabilities	"Vector lists supplying information to VTAM" on page 21
VTAM-APPCCMD	VTRINA field in the RPL extension	APPCCMD	ISTAPCVL	Supplies information to the application about a particular conversation with a partner LU	"Vector lists used during APPCCMD processing" on page 25
Application-APPCCMD	VTROUTA field in the RPL extension	APPCCMD	ISTAPCVL	Supplies information to VTAM about a particular request on a conversation with a partner LU	"Vector lists used during APPCCMD processing" on page 25

Vector lists used during OPEN processing

There are two types of vector lists associated with OPEN processing:

- Those used by the application program to supply information to VTAM
- Those used by VTAM to supply information to the application program

Vector lists supplying information to VTAM

The application program can pass information to VTAM for OPEN processing through the application-ACB vector list (pointed to by the ACBAVPTR field in the ACB). The IBM-supplied DSECT ISTVACBV enables you to refer to the fields in the application-ACB vector list symbolically.

OPEN processing vector lists built by the application have a 2–byte length field. The format of the application-capabilities vector is shown in Figure 8 on page 21.

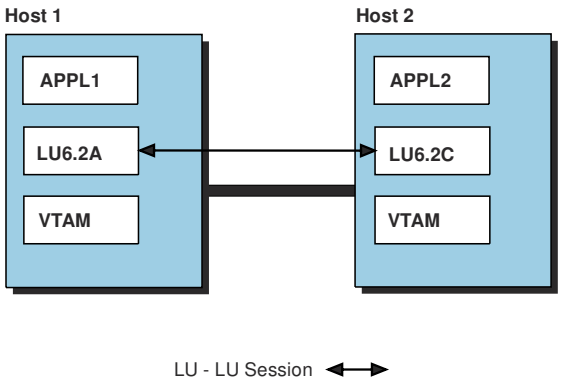


Figure 8. Format of Application-Capabilities Vector

The application-ACB vector list consists of the following vectors:

- **Application-Capabilities vector:**

  This vector is used by VTAM LU 6.2 applications to provide the following information about the application's capabilities:

  Field
  Capability

  VAC81MLE (X'80')
  Application supports having its logon exit driven multiple times per session request. Applications with LOGON exits must set this indicator to benefit from verification reduction. For more information about this function, refer to z/OS Communications Server: SNA Network Implementation Guide.

  VAC81FPR (X'40')
  Application can receive data directly into CSM buffers by specifying OPTCD=XBUFLST on the APPCCMD macroinstruction.

  VAC81PWS (X'20')
  Application can use password substitution.

  VAC81ESS (X'10')
  Application is capable of handling extended security sense codes.

  VAC81FPS (X'08')
  Application can send data directly from CSM storage by specifying OPTCD=XBUFLST on the APPCCMD macroinstruction.

  The vector ID is X'81'.

- **Local-Application's-DCE-Capability Vector:** This vector is used by LU 6.2 applications to inform VTAM about the application's DCE security capabilities. During session establishment, VTAM passes this information to the partner LU in byte 22 of the BIND.

  The vector ID is X'82'.

To supply the application-ACB vector list, an application specifies PARMS=(APPLVCTR=*address*) on the OPEN ACB macroinstruction, where *address* is the location of the application-built vector list. To build the application-ACB vector list, the application program:

1. Obtains storage for the vector list.
2. Initializes the fields using the ISTVACBV DSECT and the DSECTs contained within the ISTVACBV DSECT.

For the complete layout of the ISTVACBV DSECT, refer to z/OS Communications Server: SNA Programmer's LU 6.2 Reference.

Vector lists supplying information to the application

All VTAM application programs have access to vector lists that VTAM builds during its processing of the OPEN macroinstruction. These vector lists describe the features of VTAM to the application program and provide lists of resource identifiers that might be unknown to the application program.

Two address fields in the ACB point to the vector lists. These areas are located in storage that is read-only for the application program. The storage is addressable from the MVS address space or the VM virtual machine in which the OPEN is issued. Each address field contains the address of a vector list. The two vector lists are:

- The access-method-support vector list (pointed to by the ACBAMSVL field in the ACB DSECT). This list describes the VTAM program that processed the OPEN macroinstruction.
- The resource-information vector list (pointed to by the ACBRIVL field in the ACB DSECT). This list specifies resource identifiers and definition values that might be unknown to the application program.

OPEN processing vector lists built by VTAM have a field that is 1-byte in length. The format for vectors contained in the access-method-support vector list and the resource-information vector list is shown in Figure 9 on page 23.

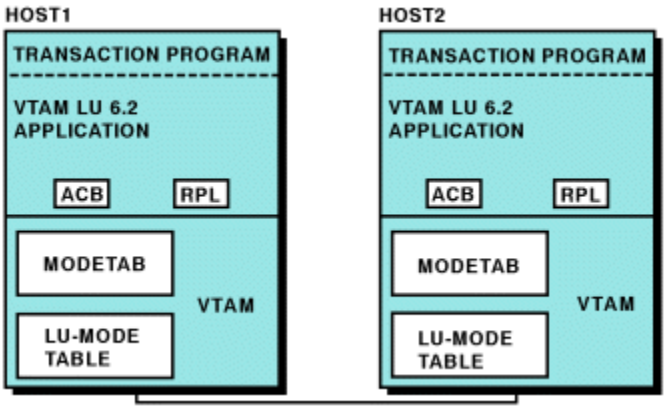


Figure 9. Format of vectors built by VTAM during OPEN processing

The vector lists can be examined at any time after the OPEN macroinstruction completes and until the CLOSE macroinstruction or equivalent terminations, such as an abend, occurs.

Access-method-support vector list

The access-method-support vector list is pointed to by the ACB's ACBAMSVL field in the ACB DSECT. This list describes the global variables for the VTAM program that processed the OPEN macroinstruction.

For the complete layout of the ISTAMSVL DSECT, refer to z/OS Communications Server: SNA Programmer's LU 6.2 Reference.

The access-method-support vector list consists of the following vectors:

- **Component-Identification Vector:** This vector contains product identification information about a major component or feature of the VTAM licensed program. This information is used by IBM for VTAM program maintenance. When a vector list contains multiple component-identification vectors, the first vector designates the base VTAM product; subsequent vectors designate features or other major components of VTAM. The vector ID is X'04'.
- **Function-List Vector:** This vector contains a variable-length bit string in which each bit corresponds to a particular VTAM function. If a bit is on, the corresponding function is present in the particular release of VTAM. If a bit is off, the function is not available. If the vector is not present, or if it is shorter than expected, you can assume the value of the missing bits to be 0. Refer to z/OS Communications Server: SNA Programmer's LU 6.2 Reference for more information about the fields contained in this vector. The vector ID is X'05'.

  The information contained in the function-list vector is also available at assembly time in global variables created with the ISTGLBAL macroinstruction. z/OS Communications Server: SNA Programming describes the use of global variables.
- **LU 6.2-Support-Function-List Vector:** This vector is provided to indicate which LU 6.2 options are supported by this release of VTAM. This vector is not present for applications that do not use VTAM's APPC API. The vector ID is X'06'.

  Each subvector in this vector list can have one of the following values that corresponds one-to-one with LU 6.2 global variables:

  X'00'
  Option is not supported.

  X'01'
  Option is supported.

  X'02'
  Pass-through. (VTAM offers support for the function, but the application program must implement the function.)

  Refer to z/OS Communications Server: SNA Programmer's LU 6.2 Reference for more information about the fields contained in this vector.

The vector list information is not available until execution time. Refer to z/OS Communications Server: SNA Programmer's LU 6.2 Reference for the list of global variables.

- **Release-Level Vector:** This vector contains the access method version and release number. The vector ID is X'01'.

Resource-information vector list

The resource-information vector list provides information about the application program that opened an ACB at execution time. The ACB's ACBRIVL field points to the list. The application program must search the vector list to find a particular vector.

For the complete layout of the ISTRIVL DSECT, refer to z/OS Communications Server: SNA Programmer's LU 6.2 Reference.

The resource-information vector list provides the following vectors:

- **APPCCMD-Vector-Area-Length Vector:** This vector contains the absolute minimum length and the recommended minimum length for full use of the APPCCMD vector area. The vector ID is X'11'.
- **Application-ACB-Name Vector:** This vector contains the ACBNAME of the application program. This is the name specified by the APPLID operand of the ACB statement; if the ACBNAME operand is not present, the network name of the application program LU is used. The vector ID is X'03'.
- **Application-Network-Name Vector:** This vector contains the network name of the application program LU. This name is specified by the name field of the APPL definition statement. The vector ID is X'02'.
- **Application-To-VTAM-Vector-Keys Vector:** This vector contains a list of all ACB vector keys presented by the application program on the VTAM-ACB-information vector. The vector ID is X'12'.
- **Host-Subarea-PU-Network-Address Vector:** This vector contains the network address of the host subarea PU. The vector ID is X'09'.
- **Host-Subarea-PU-Network-Name Vector:** This vector contains the network name of the host subarea physical unit (PU) contained in this host. This name is specified by the HOSTPU start option; if the HOSTPU start option is not specified, this vector contains the default host subarea PU name, ISTPUS. This name is part of the session awareness data provided to the NetView program or session monitor when a resource is contained within the subarea of the host PU. The vector ID is X'08'.
- **LU 6.2-Application-Definition Vector:** This vector is present in this list for LU 6.2 application programs. The LU 6.2 application program may use this vector to determine the values coded on the APPL definition statement.

  For more information about the APPL definition statement, refer to z/OS Communications Server: SNA Resource Definition Reference. Refer to z/OS Communications Server: SNA Programmer's LU 6.2 Reference for more information about the fields contained in this vector. The vector ID is X'0B'.
- **Maximum-Subarea Vector:** This vector contains the maximum subarea number (in binary) that is valid for this host's domain. This is obtained from the MAXSUBA start option. The vector ID is X'0A'.
- **Network-Name Vector:** This vector contains the name of the network in which the host resides. This name is specified by the NETID start option; if the NETID start option is not specified, this vector contains 8 bytes of blanks. The vector ID is X'06'.
- **Performance-Monitor Vector:** This vector identifies any retired fields in the performance data parameter list of the installation-wide performance monitor exit routine (ISTEXCPM). The vector points to a table of retired field entries, each of which contains information needed to locate the position of the retired field within the affected vector. Refer to z/OS Communications Server: SNA Customization for more information. Refer to z/OS Communications Server: SNA Programmer's LU 6.2 Reference for more information about the fields contained in this vector. The vector ID is X'13'.
- **SSCP-Name Vector:** This vector contains the name of the SSCP. This name is specified by the SSCPNAME start option. The name contained in this vector is used to identify the SSCP to the NetView* program. The vector ID is X'07'.

Vector lists used during APPCCMD processing

LU 6.2 applications can obtain information from VTAM by specifying the address and length of a VTAM-APPCCMD vector list on the VTRINA and VTRINL parameters of the APPCCMD macroinstruction. The application can pass vector information to VTAM by specifying the address and length of an application-APPCCMD vector list on the VTROUTA and VTROUTL parameters of the APPCCMD macroinstruction. The format of an APPCCMD vector is shown in Figure 10 on page 25.

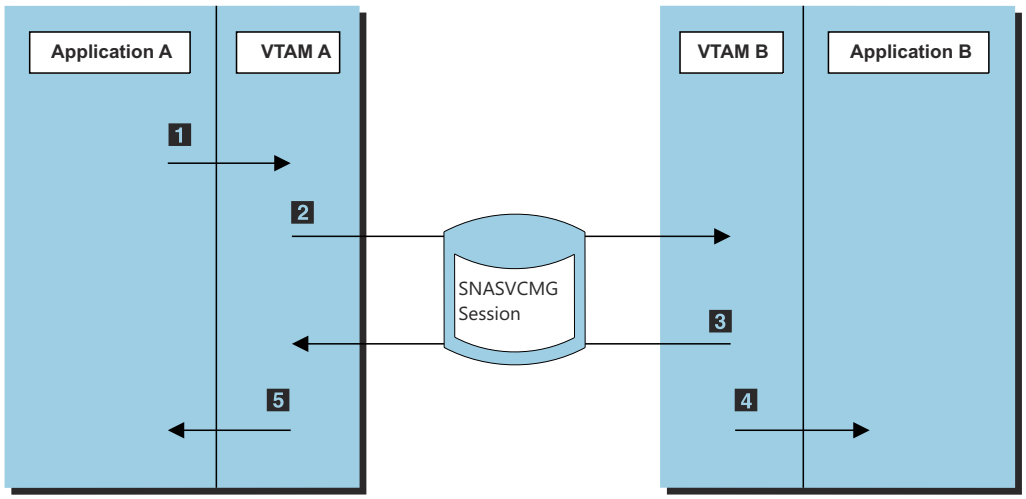


Figure 10. Format of APPCCMD vectors

Vector lists used for APPCCMD processing are mapped by the ISTAPCVL DSECT. Refer to z/OS Communications Server: SNA Programmer's LU 6.2 Reference for more information.

Vectors supplied by the application

When an application specifies the VTROUTA and VTROUTL parameters on an APPCCMD CONTROL=RECEIVE macroinstruction, the application can supply the **XBUFLST-receive** vector (vector ID=X'82'). This vector provides information to VTAM about the area in which the application will receive data in CSM buffers. See “Passing HPDT receive requirements to VTAM” on page 229 for a complete description of the fields in this vector.

Vectors returned to the application

When an application specifies the VTRINA and VTRINL parameters on an APPCCMD macroinstruction, VTAM can return the following vectors:

- **Local-Nonce Vector:** This vector contains the nonce (random data) that the local application will use to generate an encrypted password for the session on which the vector was returned. The vector can be returned on an APPCCMD CONTROL=PREALLOC macroinstruction. The vector ID is X'13'.
- **Name-Change Vector:** This vector indicates whether the partner LU is known by a name other than the one that was provided on the APPCCMD macroinstruction. This vector can be returned on the following macroinstructions:
  - APPCCMD CONTROL=ALLOC
  - APPCCMD CONTROL=OPRCNTL,QUALIFY=CNOS
  - APPCCMD CONTROL=PREALLOC

  This vector can also be returned on an ATTN CNOS exit. The vector ID is X'18'.
- **Partner-Application-Capabilities Vector:** This vector provides information about functions supported by the partner LU. This vector can be returned on the following macroinstructions:
  - APPCCMD CONTROL=ALLOC

- APPCCMD CONTROL=PREALLOC
- APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS
- APPCCMD CONTROL=OPRCNTL, QUALIFY=DISPLAY
- APPCCMD CONTROL=RCVFMH5

This vector can also be returned on the ATTN CNOS exit. The vector ID is X'1A'

- **Partner's-DCE-Capability Vector** This vector indicates whether the partner LU supports third party authentication (DCE security). If the partner does support DCE security, it indicates which type of authentication mechanism it uses. Valid authentication mechanisms include:
  - DCE authentication
  - Kryptoknight
  - Kerberos V5
  - DCE Performance Mechanism

  This vector can be returned on the following macroinstructions:
  - APPCCMD CONTROL=PREALLOC
  - APPCCMD CONTROL=RCVFMH5
  - APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS
  - APPCCMD CONTROL=OPRCNTL, QUALIFY=DISPLAY

  This vector can also be returned on an ATTN CNOS exit. The vector ID is X'12'.
- **Partner's-Nonce Vector:** This vector contains the nonce (random data) that the local application will use to verify the partner's encrypted password for the session on which the vector was returned. The vector can be returned on an APPCCMD CONTROL=PREALLOC macroinstruction. The vector ID is X'14'.
- **PCID Vector:** This vector provides the procedure correlation identifier of the session that is being used by a conversation. This vector can be returned on the following macroinstructions:
  - APPCCMD CONTROL=ALLOC
  - APPCCMD CONTROL=PREALLOC
  - APPCCMD CONTROL=RCVFMH5

  The vector ID is X'17'.
- **Receive-FMH_5-Sequence-Number Vector:** This vector contains the sequence number that the local application will use to verify the partner's encrypted password for the session on which the vector was returned. The vector can be returned on an APPCCMD CONTROL=RCVFMH5 macroinstruction. The vector ID is X'16'.
- **Send-FMH_5-Sequence-Number Vector:** This vector contains the sequence number that the local application will use to generate an encrypted password for the session on which the vector was returned. The vector can be returned on an APPCCMD CONTROL=PREALLOC macroinstruction. The vector ID is X'15'.
- **Session-Information Vector:** This vector provides information relevant to applications using CSM storage for sending and receiving data. This vector can be returned on the following macroinstructions:
  - APPCCMD CONTROL=ALLOC
  - APPCCMD CONTROL=PREALLOC
  - APPCCMD CONTROL=RCVFMH5

  The vector ID is X'19'.
- **VTAM-to-APPL-Required-Information Vector:** This vector indicates whether VTAM was able to return vector information successfully and the minimum length required to contain the vectors. The application can obtain the length required to contain the VTAM-APPCCMD vector list from the APPCCMD-vector-area-length vector, which can be found in the resource-information vector list.

  This vector is returned only when other VTAM-APPCCMD vectors are returned. The vector ID is X'10'.

Refer to z/OS Communications Server: SNA Programmer's LU 6.2 Reference for more information about the fields contained in these vectors.

Note: The vector length requested by the application on the VTRINL parameter must be large enough to accept the 2-byte length field of the VTAM-APPCCMD vector list *plus* the VTAM-to-APPL-required-information vector. Otherwise, VTAM returns an RCPRI, RCSEC combination of X'002C', X'002F', PARAMETER_ERROR— VECTOR_AREA_LENGTH_INSUFFICIENT.

Application exit routines

In addition to using an ACB, all VTAM application programs can provide VTAM with a list of exit routines to use when external events occur that affect the application program. They can, for example, provide an error-handling exit routine for VTAM to schedule when a communication link between the application program and an LU is lost. When the event occurs, VTAM gives the exit routine control as soon as possible.

The following information shows two kinds of application program exit routines and a description of each.

RPL-specified exit routines

These exit routines contain instructions to be executed when asynchronous RPL-based operations are completed. In any individual conversation-establishment, communication, or other RPL-based macroinstruction, if an RPL exit-routine address is specified, the exit routine is scheduled as an alternative to VTAM's posting an event control block (ECB) when the requested action completes. Using the ECB provides programs with greater control over the order in which events are to be handled. A program can use a mixture of ECB-posting and RPL exit routines, or it can use all one or the other.

Exit-list (EXLST) exit routines

These are special-purpose exit routines that VTAM schedules when appropriate, such as on receipt of a CINIT request as a result of a logon or initiate request. The exit-routine addresses (entry points) are specified in an exit list created with the EXLST macroinstruction.

A program can define more than one exit list by using multiple EXLST macroinstructions. However, an ACB can point to only one exit list at a time.

An exit list can be specified in an ACB and thus can be used by VTAM when an exit-routine event occurs for any session with the program, including those sessions that VTAM sets up to support LU 6.2 conversations.

The ATTN exit, which is an EXLST exit, handles LU 6.2-specific event notifications. VTAM schedules the ATTN exit when one of the following conditions occurs:

- The application program receives an FMH-5.
- VTAM processes a CNOS request from a partner LU or an operator command, and new session limits are established.
- The last session or each session of a mode name group is terminated.

For details on this exit routine, see “Using the ATTN exit” on page 239.

Not all the exit routines that application programs can supply have meaning for LU 6.2 functions. The names of the already-existing special-purpose exit routines applicable to LU 6.2 and the events that cause them to be entered are summarized in Table 2 on page 27.

Table 2. Special-purpose exit routines applicable to all session types

Exit Routine Name	Event
LERAD	A logic error has been detected for an RPL-based request.
LOGON	An application program is being requested to establish a session as a primary logical unit.

Table 2. Special-purpose exit routines applicable to all session types (continued)

Exit Routine Name	Event
LOSTERM	A session with an application program has been terminated or disrupted, or a conditional terminate request for a session has been received.
RELREQ	Another application program has requested a session with a logical unit that is presently in session with this program, and the logical unit is at its session limit.
SCIP	A session-control request has been received. This exit is only scheduled for LU 6.2 sessions when a BIND request is received for the application program. The other types of session-control requests that this exit deals with are handled by VTAM for an LU 6.2 session.
SYNAD	An error other than a logic error has been detected for an RPL-based request.
TPEND	The VTAM operator shuts down the network or this application program, or VTAM halts or abends, or a VTAM application program is being taken over.

The existing exit routines *not* applicable to LU 6.2 sessions are:

- DFASY
- NSEXIT
- RESP

For more details on the special-purpose exit routines, see Chapter 12, “Using exit routines,” on page 239.

Register usage

VTAM uses the general-purpose registers in the same fashion regardless of the type of session being used, and it uses them for a variety of purposes. VTAM frequently passes back return codes and parameter-list addresses in the registers. Refer to z/OS Communications Server: SNA Programmer's LU 6.2 Reference for a summary of how VTAM uses the registers.

The APPCCMD macroinstruction follows register usage of other RPL-based macroinstructions, including the requirement that register 13 point to an 18-word save area. VTAM does define some additional codes for LU 6.2 services. See the following sections for further information:

- For information on evaluating register feedback, see “Registers” on page 47.
- For information on register contents when using the ATTN exit, see “Using the ATTN exit” on page 239.

(For more details on handling VTAM errors and special conditions associated with the APPCCMD macroinstructions, see Chapter 14, “Handling errors,” on page 263.)

Operating system environment

LU 6.2 support, with one exception, does not affect the operating system environment for VTAM application programs. Refer to z/OS Communications Server: SNA Programming for a description of this environment.

Application programs can still use multitasking, service request blocks (SRBs), and authorized path in their application programs. The only LU 6.2 restriction is that VTAM does not support the full use of the MVS multiple-address-space facility for LU 6.2 application programs. Application programs can issue APPCCMD macroinstructions *only* from the address space where the ACB was opened. Refer to the section on operating system considerations in z/OS Communications Server: SNA Programming for details and restrictions on using cross-memory mode.

Sessions established by applications defined as APPC=YES in the application major node are associated with the task that opens the ACB.

Overview of LU 6.2 transaction processing

Although each LU 6.2 application program is unique, part of the initial sequence of information is similar from one application program to another. The following examples show a typical sequence for an inquiry transaction for both the initiator and the responder.

Initiator's view of the inquiry transaction:

- At Application Startup
  - APPC=YES on the APPL statement.
  - Open ACB.
  - Enable sessions—SETLOGON OPTCD=START.
  - Initialize session limits with CNOS—APPCCMD.
- Ongoing:
  - Allocate a conversation—APPCCMD.
  - Send data—APPCCMD.
  - Receive and wait for a reply—APPCCMD.
  - Deallocate the conversation—APPCCMD.

Responder's view of the inquiry transaction:

- At Application Startup:
  - APPC=YES on the APPL statement.
  - Open ACB.
  - Enable sessions—SETLOGON OPTCD=START.
  - Be informed of session limits—ATTN (CNOS) exit.
- Ongoing:
  - Be informed of the allocate—ATTN (FMH-5) exit or RPL.
  - Receive the allocate—APPCCMD.
  - Receive the data—APPCCMD.
  - Send the reply—APPCCMD.
  - Receive deallocate—APPCCMD.
  - Wait for the next request.

Chapter 3. How VTAM implements LU 6.2 architecture

About this chapter

Although the VTAM program implements a large portion of the LU 6.2 function, the application program must implement some of it. The application program also can implement some optional LU 6.2 functions. This chapter describes the following items:

- Functions that VTAM implements
- Functions that the application program implements (pass-through verbs and mapped conversation verbs)
- Optional LU 6.2 functions

Refer to *SNA Transaction Programmer's Reference Manual for LU Type 6.2* for more information on the architecture.

LU 6.2 verbs

LU 6.2 architecture defines functions in terms of verbs. The process of establishing a conversation, for example, is defined with the ALLOCATE verb.

Verbs that VTAM implements

The VTAM APPCCMD macroinstruction implements the functions of the following LU 6.2 verbs:

Conversation verbs

The following information shows LU 6.2 conversation verbs.

ALLOCATE
Start a conversation.

CONFIRM
Transmit any data in the send buffer and request confirmation that the data has been received and meets criteria specified by the transaction programs.

CONFIRMED
Give confirmation that all data to this point has been received successfully and meets criteria specified by the transaction programs.

DEALLOCATE (except TYPE=LOCAL)
End a conversation. (TYPE=LOCAL is explained under “Verbs not supported by VTAM” on page 34.)

FLUSH
Transmit any data remaining in the send buffer.

PREPARE_TO_RECEIVE
Transmit any data remaining in the send buffer along with an indication that the remote transaction program can now transmit. This can include the function of the CONFIRM verb.

RECEIVE_AND_WAIT
If information sent by the partner LU has arrived, receive it; if no information has been sent yet, wait for it.

RECEIVE_EXPEDITED_DATA
Receive expedited data from a partner LU.

RECEIVE_IMMEDIATE
If information sent by the partner has arrived, receive it; if no information has been sent, do not wait for it.

REQUEST_TO_SEND
Ask the partner LU for permission to send data.

SEND_DATA
Place data in the send buffer and, if enough data has accumulated, transmit it.

SEND_ERROR
Place information describing an error in the send buffer and, if enough data has accumulated, transmit it. (Negative responses to confirmation requests are sent immediately, without being buffered.)

SEND_EXPEDITED_DATA
Send expedited data to a partner LU on a full-duplex-capable session.

Control operator verbs

The following information shows control operator verbs:

CHANGE_SESSION_LIMIT
Change the LU-mode session limit and number of contention-winner sessions for each LU for parallel-session connections. The new LU-mode session limit and number of contention-winner sessions for each LU are enforced until changed by another CNOS verb. As a result of this change, LU-LU session with the specified mode name might be activated or deactivated to match the new session limits.

DEACTIVATE_CONVERSATION_GROUP
Deactivate the session associated with a specified conversation group ID.

INITIALIZE_SESSION_LIMIT
Establish the initial LU-mode session limit for single-session or parallel-session connections and the contention-winner polarities for parallel-session connections. Initialization of the session limit might cause one or more LU-LU sessions with the specified mode name to be activated.

RESET_SESSION_LIMIT
Reset to 0 the LU-mode session limit for both single-session or parallel-session connections and the contention-winner polarities for the parallel-session connections. This function causes all active sessions with the specified mode name, or all mode names, to be deactivated.

Pass-through verbs (application program implements)

Although VTAM does not implement the following verbs, VTAM provides services that enable application programs to implement them. In some cases, the application program handles everything, and the data passes through VTAM. An example of this would be the LUW_IDENTIFIER. In other cases, the application program handles most things, but requires assistance from VTAM. For example, the application program handles the use of SECURITY_USER_ID and SECURITY_PROFILE in conversation security, but VTAM negotiates conversation security at the time of the BIND. Also, the application program handles SYNCPT and BACKOUT, but VTAM provides a function for that with APPCCMD CONTROL=REJECT and APPCCMD CONTROL=SETSESS and the handling of the PS headers.

GET_ATTRIBUTES
Returns information pertaining to the specified conversation.

Much of the information defined for GET_ATTRIBUTES can be found in the FMH-5 that is used to start a conversation. To obtain this function, the application program should save the pertinent information from the FMH-5 when it is received or built.

The types and location of information returned on GET_ATTRIBUTES are:

- CONVERSATION_CORRELATOR

  A unique value used to correlate conversations. It is available in the FMH-5.
- CONVERSATION_GROUP_ID

  The variable for returning the conversation group identifier. This value is returned to the application program on the RPL extension for the conversation winner on the APPCCMD CONTROL=ALLOC and APPCCMD CONTROL=RCVFMH5 macroinstructions.
- CONVERSATION_STATE

The variable for returning the current conversation state of the specified conversation. The current conversation state is returned in the RPL extension for every conversation-based (APPCCMD) macroinstruction.

- MODE_NAME

  The mode name group used for a session supporting a conversation. It is a returned parameter on an RCVFMH5 macroinstruction. The initiating LU must save the mode name it has requested.

- PARTNER_LU_NAME

  The variable for returning the name of the LU at which the remote transaction program is located. This is a name by which a local LU knows the partner-LU for the purpose of allocating a conversation. For more details, refer to the description of the LUNAME operand of the APPCCMD CONTROL=ALLOC macroinstructions in the z/OS Communications Server: SNA Programmer's LU 6.2 Reference. This name is contained in the BIND and in the read-only copy of the RPL provided by the ATTN(FMH5) exit.

- PARTNER_NETWORK_QUALIFIED_LU_NAME

  The variable for returning the network-qualified name of the LU at which the remote transaction program is located. If the partner's network-qualified LU name is not known, a null value is returned. It can be obtained by using APPCCMD CONTROL=OPRCNTL, QUALIFY=DISPLAY. This name is also contained in the BIND.

- SYNC_LEVEL

  Indicates the synchronization level of the conversation, as defined by LU 6.2 architecture. It is available in the FMH-5.

GET_TP_PROPERTIES

Returns information pertaining to the transaction program that issues the verb.

The types and location of information returned on GET_TP_PROPERTIES are:

- LUW_IDENTIFIER

  A unique conversation identifier used for accounting and sync point purposes. It is available in the FMH-5.

- OWN_FULLY_QUALIFIED_LU_NAME

  The application program's unique network name, consisting of the network identifier, if applicable, and the application LU name. The application program can find the network identifier (if present) in the X'06' vector and the application LU name in the X'02' vector in the resource-information vector list. This list is available to the application program when the OPEN for the ACB of the application program completes. (See “Vector lists” on page 20 for the details of the resource-information vector list.)

- OWN_TP_INSTANCE

  The variable for returning the system-generated identifier for this instance of the transaction program.

- OWN_TP_NAME

  The variable for returning the local application program's own transaction program name. This information is received in the ATTN(FMH5) and in data of the RCVFMH5 macroinstruction.

- PROTECTED_LUW_IDENTIFIER

  The variable for returning the logical unit of work (LUW) identifier used by the transaction program when it accesses protected resources. For VTAM LU 6.2, this is the same as the LUW identifier, which is in the data returned for the RCVFMH5 macroinstruction.

- SECURITY_PROFILE

  A profile used for conversation security purposes. It is available in the FMH-5, in an access security subfield.

- SECURITY_USER_ID

A user identifier used for conversation-level security. It is available in a security access subfield in the FMH-5.

GET_TYPE
Returns the type of conversation (full-duplex basic or full-duplex mapped). This information is available in the FMH-5.

Note: GET_ATTRIBUTES, GET_TP_PROPERTIES, and GET_TYPE are not technically pass-through verbs because the application program does not issue a macroinstruction that is processed by VTAM.
The information requested by the GET_ATTRIBUTES, GET_TP_PROPERTIES, and GET_TYPE verbs is conversation-level information (for example, the partner network-qualified LU name or whether the conversation is basic or mapped). To manage its transaction programs, the application program must maintain this information. VTAM does not maintain this type of information. The GET_ATTRIBUTES, GET_TP_PROPERTIES, and GET_TYPE verbs are listed here to show that the function of these verbs, although not supported by VTAM, can be achieved by the application program.

BACKOUT
Restore all protected resources to their status as of the previous synchronization point.

POST_ON_RECEIPT
Requests that the conversation be posted when data or other information is available.

PREPARE_FOR_SYNCPT
Causes protected resources associated with the conversation to be prepared to advance to the next synchronization point. Protected resources are those currently allocated to the transaction with a synchronization level of SYNCPT. As part of processing the PREPARE_FOR_SYNCPT verb, the LU flushes its send buffer for the conversation.

SET_SYNCPT_OPTIONS
Changes the options that affect the processing of the SYNCPT, BACKOUT, and PREPARE_FOR_SYNCPT verbs. The default options are in effect when the transaction program starts. The options set by this verb remain in use until the verb is issued again or the transaction program ends. The program is not required to specify any parameter having a value that it does not need to change.

SYNCPT
Advance all protected resources to the next synchronization point.

Mapped conversation verbs (application program implements)

These verbs provide an interface that is at a higher level than that provided by the basic conversation verbs and that allows data mapping.

Implementing mapped conversations means that the application program is offering its own API, consistent with LU 6.2 architecture for mapped conversations, to its transaction programs. The application program translates input from this API into VTAM macroinstructions.

For information on mapped conversations, refer to *SNA Transaction Programmer's Reference Manual for LU Type 6.2*.

Verbs not supported by VTAM

VTAM does not support the following conversation and control operator verbs.

Conversation verbs

The following conversation verbs are not supported:

DEALLOCATE (TYPE=LOCAL)
VTAM does not define a local type of deallocation corresponding to the DEALLOCATE (TYPE=LOCAL) described in the LU 6.2 architecture. The purpose of the local type of deallocation is to enable a transaction program to inform the LU of whether it wants a resource identifier to be discarded or retained for a future reconnection. Because VTAM does not support the reconnect function, it always discards resource identifiers at the end of a conversation. Therefore, no local type of deallocation is necessary across the VTAM API.

RECONNECT
Reconnects a conversation that previously existed between the local transaction program and a specified remote transaction program.

TEST
Determines if the conversation has been posted or if a request-to-send indication has been received from the remote transaction program. The request-to-send portion may be accomplished by APPCCMD CONTROL=TESTSTAT.

WAIT
Waits for any conversation in a list of conversations to be posted.

Control operator verbs

The following control operator verbs are not supported:

DELETE
Deletes parameter values, established by the DEFINE verbs, that control the operation of the local LU.

PROCESS_SIGNOFF
Permits a privileged service transaction program to instruct the LU to receive and process one or more Sign-Off GDS variables.

SIGNOFF
Permits a privileged program to remove list entries from the LU's signed-on lists. The control operator uses this verb to maintain the signed-on list.

LU 6.2 option sets

This section describes the support that VTAM provides for the option sets listed in the LU 6.2 architecture. Option sets are LU 6.2 functions that are not required for the minimum implementation of a type 6.2 LU. The option set number is in parentheses following the option set name.

Option sets that VTAM implements

VTAM implements the following conversation option sets:

Flush the LU's send buffer (101)
Enables a transaction program to explicitly cause the LU to transmit any data in its SEND buffer, regardless of the amount of data in the buffer. Any of the APPCCMD macroinstructions that send data accept a QUALIFY value that forces data transmission through the network.

PREPARE_TO_RECEIVE (105)
Enables an application program to change a half-duplex conversation state from SEND to RECEIVE and, at the same time, include the function of the FLUSH or CONFIRM verb. The APPCCMD CONTROL=PREPRCV macroinstructions implement this option set.

Receive immediate (106)
Enables a transaction program to receive whatever information is available on a conversation without having to request posting of the conversation. The functions available through VTAM (including the VTAM receive-any function and asynchronous return after issuing an APPCCMD macroinstruction) largely negate any need for this option set.

Full-duplex and expedited data support (112)
Enables a transaction program to allocate a full-duplex conversation. This option set also enables the transaction program to send and receive expedited data. Option sets 101, 110, 113, and 247 are required in addition to this option set for full-duplex support.

Nonblocking architecture (113)
Enables support for nonblocking architecture. This option set provides enhancement for half-duplex conversations and provides the basic functions required to support full-duplex conversation architecture.

Queued allocation of a contention-winner session (201)
Enables a local program to allocate a conversation to a remote program on a session for which the local LU must be the contention winner. It also enables a local program to queue the conversation request if no contention-winner session is available immediately.

Immediate allocation of a session (203)
Enables a program to allocate a contention-winner session only if one is immediately available; otherwise, the allocation is unsuccessful. The application program issues the APPCCMD CONTROL=ALLOC, QUALIFY=IMMED macroinstruction.

Queued allocation for when session free (205)
Enables a local program to allocate a conversation to a remote program on a session for which the local program waits for the duration of a session activation, if necessary. If session activation cannot be accomplished by the LU 6.2 application program, control is returned to the local program if activation fails. The application program issues the APPCCMD CONTROL=ALLOC, QUALIFY=WHENFREE macroinstruction.

Long locks (244)
Enables a transaction program on a half-duplex conversation to issue a PREPARE_TO_RECEIVE verb, which includes the function of the CONFIRM verb, and to resume processing when data is subsequently received from the remote transaction program. The LOCKS=LONG keyword on the APPCCMD CONTROL=PREPRCV macroinstructions implement this option set.

Test for request to send received (245)
Enables a transaction program on a half-duplex conversation to test whether a request-to-send notification has been received on a conversation (for example, following sync point processing).

VTAM and security option sets

The LU 6.2 architecture defines a number of conversation-level security option sets that include passwords, user identifiers, and profiles in allocation requests. The LU 6.2 architecture also defines a session-level security option set. The architecture requires that session-level LU-LU verification be allowed when conversation-level security option sets are enabled and when the LUs that make up the network are not physically secure (as determined by installation management).

VTAM supports session-level security and offers pass-through support for conversation-level security. It is the responsibility of the application programs to implement conversation-level security.

Because VTAM's security support is compatible with the architecture's conversation-level security features, application programs can have conversations with LUs that implement the architecture's security options. These conversations can make use of passwords, user identifiers, and profiles in implementing security procedures.

Session-level security option sets

VTAM supports the following session-level security option:

Session-level LU-LU verification (211)
Enables two LUs to verify each other's identity before a session is established between them.

Conversation-level security option sets

VTAM supports the following conversation-level security option sets:

User ID verification (212)
Enables an application program to specify a user identifier and password that the local LU can use to verify a user identifier on a request for a conversation. The application program that receives the request can use this information to determine whether to accept the conversation request.

Program supplied user identifier and password (213)
Enables a program that is allocating a conversation to supply the user identifier and password to be sent on the allocation request. VTAM offers pass-through support for this option set by allowing security access subfields on the allocation requests. The application program can implement the option set.

User ID authorization (214)
Enables a program or operator to designate the user identifiers that are authorized to access specific resources of the LU, such as transaction programs. VTAM offers pass-through support for this option set by allowing security access subfields on the allocation requests. The application program can implement the option set.

Profile verification and authorization (215)
Enables a program or operator to designate the profiles that the local LU uses to verify a profile carried on allocation requests it receives. The program can also designate the profiles that are authorized to specific resources of the LU, such as transaction programs. VTAM offers pass-through support for this option set by allowing security access subfields on the allocation requests. The application program can implement the option set.

Profile pass-through (217)
Enables the transaction program that is allocating a conversation to specify that the allocation request carry the profile received on the request that started the program. The application program can implement this option set by maintaining a table of profiles used when allocation requests are received. VTAM offers pass-through support for this option set by allowing security access subfields on the allocation requests. The application program can implement the option set.

Program supplied profile (218)
Enables the application program that is allocating a conversation to supply the profile to be sent on the allocation request. VTAM offers pass-through support for this option set by allowing security access subfields on the allocation requests. The application program can implement the option set.

Control operator option sets

VTAM supports the following control operator option sets:

CHANGE_SESSION_LIMIT verb (501)
Enables a program at the source LU to request a change in the mode name group session limit from one nonzero value to another or to request a change in the minimum number of contention-winner sessions for the source LU or target LU. This is implemented through APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS.

LU-definition verbs (505)
Enables a program or an operator to specify the operating parameters of its LU. These parameters include the LU's network-qualified name, session limit information, and security information such as passwords and profiles. See Chapter 6, “Managing sessions,” on page 83 for information on controlling session limits. Security information can be maintained by the application program independently of VTAM.

MIN_CONTENTION_WINNERS_TARGET parameter (601)
Enables a program at the source LU to request a nonzero value for the target LU's minimum number of contention-winner sessions. This is implemented through APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS.

RESPONSIBLE(TARGET) parameter (602)
Enables a program at the source LU to request that the target LU deactivate sessions when a CNOS verb requires a decrease in the number of active sessions. This is implemented through the RESP field in the CNOS session limits control block.

DRAIN_TARGET(NO) parameter (603)
Enables a transaction program to prevent its partner from honoring queued conversation requests for a mode name group when the group's session limit is being set to 0. This is implemented with the DRAINR field in the CNOS session limits data structure.

Locally known LU names (606)
Enables a program or an operator to specify the locally known names of remote LUs. The 1- through 8-byte LU name that is used on all of VTAM's APIs is considered by architecture to be the locally known name for the network-qualified LU name.

Uninterpreted LU names (inbound only) (607)
Enables a program to specify the uninterpreted names of partner LUs. VTAM provides this function as part of its services to all of its application programs. It is not unique to LU 6.2 support.

Single-session reinitiation (608)
Enables a program to specify the responsibility for reinitiating single sessions to partner LUs. VTAM does this by ensuring that the BIND image used to establish a single session specifies that either side of the session can reinitiate.

Maximum RU size bounds (610)
Enables a program to specify the upper bounds for the maximum RU sizes on sessions within a mode name group. The application program can control maximum RU sizes by setting fields in the BIND when issuing APPCCMD CONTROL=OPRCNTL, QUALIFY=ACTSESS.

Session-level mandatory cryptography (611)
Enables a program or an operator to specify that session-level mandatory cryptography is to be used on sessions within an LU-mode group.

Contention winner automatic activation limit (612)
Enables a program to specify the limit for automatically activating contention-winner sessions within a mode name group. This option set can be implemented through the use of the AUTOSES parameter on the APPL definition statement.

Session-level selective cryptography (617)
Enables data that is both encrypted and not encrypted to be sent on the same session.

Option sets that the application program implements

Although VTAM does not offer the option sets in this section, the application program can implement them. An application should verify VTAM's support for these option sets at assembly time with the ISTGAPPC macroinstruction, or at OPEN ACB time by examining the LU-6.2-function-list vector in the access-method-support vector list:

Get attributes (102)
Enables a program to obtain attributes of a mapped conversation.

Post on receipt with wait (104)
Enables a transaction program to request posting of multiple conversations and to wait (suspend its processing) until information is available on any one of the conversations. The functions available through VTAM (including the VTAM receive-any function and asynchronous return after issuing an APPCCMD macroinstruction) largely negate any need for this option set.

Sync point services (108)
Enables a program to request synchronization point processing of all protected resources (any resource to which access is controlled) throughout the transaction. Transaction programs can synchronize their resources when partners in protected conversations fail or encounter errors. To keep databases consistent, a network of protected resources is built into a synchronization tree. Sync point services include the SYNCPT and BACKOUT verbs.

Get conversation type (110)
Enables a program that supports both basic and mapped conversations to determine which category of verbs it should use in conjunction with a resource identifier.

Conversations between transaction programs located at the same LU (204)
Enables a local transaction program to allocate a conversation to a remote transaction program located at the same LU as the local program. This option set is not available for applications that are either single-session capable or serving as a generic resource.

Send persistent verification (219)
Enables a program to allocate consecutive conversations during which the conversation-level security information, once verified, remains verified for specific combinations of user identifiers, profiles, and partner LUs.

Receive persistent verification (220)
Enables a program or an operator to designate the remote LUs from which the local LU can accept consecutive allocation requests that, once verified, remain verified for specific combinations of user identifiers, profiles, and partner LUs.

Password substitution (223)
Enables an LU to send or receive tokens that represent an end user's password for an ALLOCATE.

A VTAM application indicates support for this option on the application capabilities vector. For more information, see "Vector lists supplying information to VTAM" on page 21.

Extended Security Sense Codes (225)
Allows an LU to generate and return a more granular set of return codes for security errors during ALLOCATE. The more granular security return codes provide a greater level of detail in regards to the security failures on ALLOCATE. The use of the more granular return codes is indicated to the partner on the BIND. Option set 211 is a prerequisite.

A VTAM application indicates support for this option on the application capabilities vector. For more information, see "Vector lists supplying information to VTAM" on page 21. Extended security sense codes are listed in z/OS Communications Server: IP and SNA Codes.

Authentication using GSS-API Mechanisms (230)
Allows a program or an operator to designate the remote LUs that are permitted to send to the local LU allocation requests carrying a DCE ticket based authentication information. This option set also allows the program allocating a conversation to specify that the allocation request carry DCE ticket based access security information determined by the local LU.

A VTAM application indicates support for this option on the application capabilities vector. For more information, see "Vector lists supplying information to VTAM" on page 21.

Optimization using GSS_Continue_Deferred (231)
Allows an LU to optimize performance when using a GSS-API Authentication Mechanism that returns GSS_Continue_Deferred.

Option sets 230 and 243 are prerequisites.

Send PIP data (241)
Enables the local program allocating a conversation to provide the remote program with program initialization parameter (PIP) data. VTAM offers pass-through support for this option by allowing PIP data on the FMH-5.

Receive PIP data (242)
Enables the local program to receive some program initialization parameters from the remote program allocating a conversation. VTAM offers pass-through support for this option by allowing PIP data on the FMH-5.

Accounting (243)
Enables an LU implementation to generate and send both a logical-unit-of-work (LUW) identifier and a conversation correlator (CC) to the remote LU.

Data mapping (246)
Enables a program to request mapping of the data by the local and partner LUs. This is done through the use of mapped conversations.

FMH data (247)
Enables programs to send and receive data records containing FM header data.

Mapped conversation LU services component (291)
Enables implementation of a mapped conversation LU services component program, which processes mapped conversation verbs.

Logging of data in a system log (296)
Enables a transaction program to record error information in the system's error log.

Option sets that VTAM does not offer

VTAM does not offer the following conversation and control operator option sets:

Unsupported conversation option sets

The application program should not attempt to implement these option sets. They are part of the conversation verb option sets related to either verbs or verb parameters.

Post on receipt with test for posting (103)
Enables a transaction program to request posting of a conversation and to test the conversation to determine whether information is available.

Unsupported control operator option sets

The application program should not attempt to implement these option sets. They are part of the control operator verb option sets related to verb parameters and are found in the *SNA Transaction Programmer's Reference Manual for LU Type 6.2*.

ACTIVATE_SESSION verb (502)
Enables a program to activate LU-LU sessions.

DEACTIVATE_SESSION verb (504)
Enables a program to deactivate LU-LU sessions. However, you can still use a VTAM VARY command, such as VARY TERM, to deactivate a session. Refer to z/OS Communications Server: SNA Operation for details on the VARY commands. The application program can use CNOS to decrease session limits to cause VTAM to deactivate sessions (if necessary) to adhere to the new limit. The application program can also use the APPCCMD CONTROL=REJECT macroinstruction to deactivate sessions and conversations due to protocol violation or for cleanup purposes.

FORCE parameter (604)
Enables a program or an operator to specify that the mode name group session limit be reset to 0 even if the CNOS exchange between the source LU and target LU is unsuccessful.

LU-LU session limit (605)
Enables a program or an operator to specify the LU-LU session limit.

LU 6.2 verb cross reference

Table 3 on page 40 indicates the correspondence between the LU 6.2 architected verbs and VTAM's APPCCMD macroinstruction. VTAM does not implement some LU 6.2 verbs, and the LU 6.2 architecture does not have some APPCCMD macroinstructions.

Note: Table 3 on page 40 is used to represent related verbs and macroinstructions. The correlation does not imply an exact duplication of function.

Table 3. Verb-to-macro cross-reference

LU 6.2 Verb	APPCCMD Macroinstruction
ACTIVATE_SESSION	(none)
ALLOCATE (LUNAME=OWN)	CONTROL=ALLOC, LUNAME=*target*, where target is the same as the source
ALLOCATE (RETURN_CONTROL=IMMEDIATE)	CONTROL=ALLOC, QUALIFY=IMMED
ALLOCATE (RETURN_CONTROL= WHEN_CONVERSATION GROUP_ALLOCATED)	CONTROL=ALLOC, QUALIFY=CONVGRP
ALLOCATE (RETURN_ CONTROL= WHEN_CONWINNER_ALLOCATED)	CONTROL=ALLOC, QUALIFY=CONWIN

Table 3. Verb-to-macro cross-reference (continued)

LU 6.2 Verb	APPCCMD Macroinstruction
ALLOCATE (RETURN_CONTROL= WHEN_SESSION_ALLOCATED)	CONTROL=ALLOC, QUALIFY=ALLOCD
ALLOCATE (RETURN_CONTROL= WHEN_SESSION_FREE)	CONTROL=ALLOC, QUALIFY=WHENFREE
BACKOUT	(none)
CHANGE_SESSION_LIMIT	CONTROL=OPRCNTL, QUALIFY=CNOS
CONFIRM	CONTROL=SEND, QUALIFY=CONFIRM
CONFIRMED	CONTROL=SEND, QUALIFY=CONFRMD
DEACTIVATE_CONVERSATION_GROUP	CONTROL=REJECT, QUALIFY=CONVGRP
DEACTIVATE_SESSION	(none)
DEALLOCATE (TYPE=FLUSH)	CONTROL=DEALLOC, QUALIFY=FLUSH
DEALLOCATE (TYPE=SYNC_LEVEL(CONFIRM))	CONTROL=DEALLOC, QUALIFY=CONFIRM
DEALLOCATE (TYPE=ABEND_PROG)	CONTROL=DEALLOC|DEALLOCQ, QUALIFY=ABNDPROG
DEALLOCATE (TYPE=ABEND_SVC)	CONTROL=DEALLOC|DEALLOCQ, QUALIFY=ABNDSERV
DEALLOCATE (TYPE=LOCAL)	(none)
DEALLOCATE (TYPE=TIMER)	CONTROL=DEALLOC|DEALLOCQ, QUALIFY=ABNDTIME
DELETE	(none)
FLUSH	CONTROL=SEND, QUALIFY=FLUSH
GET_ATTRIBUTES	(none)
GET_TP_PROPERTIES	(none)
GET_TYPE	(none)
INITIALIZE_SESSION_LIMIT	CONTROL=OPRCNTL, QUALIFY=CNOS
POST_ON_RECEIPT	CONTROL=TESTSTAT
PREPARE_FOR_SYNCPT	(none)
PREPARE_TO_RECEIVE (LOCKS=LONG, TYPE=SYNC_LEVEL(CONFIRM))	CONTROL=PREPRCV, QUALIFY=CONFIRM, LOCKS=LONG
PREPARE_TO_RECEIVE (LOCKS=SHORT, TYPE=SYNC_LEVEL(CONFIRM))	CONTROL=PREPRCV, QUALIFY=CONFIRM, LOCKS=SHORT
PREPARE_TO_RECEIVE (TYPE=FLUSH)	CONTROL=PREPRCV, QUALIFY=FLUSH
PROCESS_SESSION_LIMIT	(none)
PROCESS_SIGNOFF	(none)
RECEIVE_AND_WAIT (FILL=BUFFER)	CONTROL=RECEIVE, QUALIFY=SPEC, FILL=BUFF
RECEIVE_AND_WAIT (FILL=LL)	CONTROL=RECEIVE, QUALIFY=SPEC, FILL=LL
RECEIVE_EXPEDITED_DATA	CONTROL=RCVEXPD, QUALIFY=ANY|IANY

Table 3. Verb-to-macro cross-reference (continued)

LU 6.2 Verb	APPCCMD Macroinstruction
RECEIVE_EXPEDITED_DATA	CONTROL=RCVEXPD, QUALIFY=SPEC|ISPEC
RECEIVE_IMMEDIATE (FILL=BUFFER)	CONTROL=RECEIVE, QUALIFY=ISPEC, FILL=BUFF
RECEIVE_IMMEDIATE (FILL=LL)	CONTROL=RECEIVE, QUALIFY=ISPEC, FILL=LL
RECONNECT	(none)
RESET_SESSION_LIMIT	CONTROL=OPRCNTL, QUALIFY=CNOS
REQUEST_TO_SEND	CONTROL=SEND, QUALIFY=RQSEND
SEND_DATA	CONTROL=SEND, QUALIFY=DATA
SEND_DATA and CONFIRM	CONTROL=SEND, QUALIFY=DATACON
SEND_DATA and DEALLOCATE (TYPE=CONFIRM)	CONTROL=DEALLOC, QUALIFY=DATACON
SEND_DATA and DEALLOCATE (TYPE=FLUSH)	CONTROL=DEALLOC, QUALIFY=DATAFLU
SEND_DATA and FLUSH	CONTROL=SEND, QUALIFY=DATAFLU
SEND_DATA and PREPARE_TO RECEIVE (TYPE=CONFIRM, LOCKS=LONG)	CONTROL=PREPRCV, QUALIFY=DATACON, LOCKS=LONG
SEND_DATA and PREPARE_TO RECEIVE (TYPE=CONFIRM, LOCKS=SHORT)	CONTROL=PREPRCV, QUALIFY=DATACON, LOCKS=SHORT
SEND_DATA and PREPARE_TO RECEIVE (TYPE=FLUSH)	CONTROL=PREPRCV, QUALIFY=DATAFLU
SEND_ERROR (TYPE=PROG)	CONTROL=SEND, QUALIFY=ERROR, TYPE=PROGRAM
SEND_ERROR (TYPE=SVC)	CONTROL=SEND, QUALIFY=ERROR, TYPE=SERVICE
SEND_EXPEDITED_DATA	CONTROL=SENDEXPD
SET_SYNCPT_OPTIONS	(none)
SIGNOFF	(none)
SYNCPT	(none)
TEST	(none)
TEST_POSTED	(none)
TEST_REQUEST_TO_SEND_RECEIVED	CONTROL=RCVEXPD, QUALIFY=ANY|IANY|ISPEC|SPEC
WAIT	(none)
(none)	CONTROL=CHECK
(none)	CONTROL=DEALLOC|DEALLOCQ, QUALIFY=ABNDUSER
(none)	CONTROL=OPRCNTL, QUALIFY=ACTSESS
(none)	CONTROL=OPRCNTL, QUALIFY=DACTSESS
(no direct correlation between verb and macroinstruction)	CONTROL=OPRCNTL, QUALIFY=DEFINE
(no direct correlation between verb and macroinstruction)	CONTROL=OPRCNTL, QUALIFY=DISPLAY

Table 3. Verb-to-macro cross-reference (continued)

LU 6.2 Verb	APPCCMD Macroinstruction
(none)	CONTROL=OPRCNTL, QUALIFY=RESTORE
(none)	CONTROL=PREALLOC
(none)	CONTROL=RCVFMH5
(none)	CONTROL=RECEIVE, QUALIFY=ANY
(none)	CONTROL=RECEIVE, QUALIFY=IANY
(none)	CONTROL=REJECT, QUALIFY=CONV
(none)	CONTROL=REJECT, QUALIFY=SESSION
(none)	CONTROL=RESETRCV
(none)	CONTROL=SEND, QUALIFY=ERROR, TYPE=USER
(none)	CONTROL=SENDFMH5
(none)	CONTROL=SENDRCV
(none)	CONTROL=SETSESS, QUALIFY=RESUME
(none)	CONTROL=SETSESS, QUALIFY=SUSPEND
(none)	CONTROL=SETSESS, QUALIFY=SYNCBEG
(none)	CONTROL=SETSESS, QUALIFY=SYNCEND

VTAM LU-mode table

The LU 6.2 architecture describes system definition data structures that represent the state and configuration of the LU's resources. These data structures include information about the partner LUs with which an LU can communicate and the mode names characterizing possible sessions with particular LUs.

Data structures

The architectural data structures and VTAM's representation of those structures are provided in this section.

Architectural base

The data structures defined by the architecture are:

- The **PARTNER_LU** structure describes a partner LU. This information includes the partner LU's names, local LU name, network-qualified LU name, and uninterpreted LU name. It also includes the set of the LU's optional capabilities such as parallel sessions. The PARTNER_LU structure also contains a list of mode descriptions. The list of mode descriptions has one entry for each mode name that is defined for the particular partner LU name.
- The **MODE** structure describes a logon mode that can be used to establish sessions with a partner LU. This structure contains the session parameters that characterize this mode, such as maximum RU size. It also includes the set of optional functions that are supported by the partner LU on a mode basis.

  The MODE structure contains several session limit fields. These fields limit the total number of sessions and the number of contention-winner and contention-loser sessions that this LU can have with the partner LU and mode name represented by the MODE structure.

VTAM's representation of data structures

VTAM's representation of the data structures is the LU-mode table. One LU-mode table exists for each opened ACB of an application program capable of using the VTAM LU 6.2 interface. It contains two kinds of information:

- Defined

  Information provided by system definition (on the APPL statement) or the application program (on APPCCMD CONTROL=OPRCNTL, QUALIFY=DEFINE) to define values used when negotiating a CNOS reply.

- Dynamically accumulated

  Information dynamically accumulated by VTAM to record data pertinent to the partner LU, or mode, or both. This information is used by VTAM when controlling the assignment of conversations to sessions and when controlling the activation and deactivation of sessions. Some of it can be displayed to the application program with APPCCMD CONTROL=OPRCNTL, QUALIFY=DISPLAY.

Blank mode names

VTAM's support for LU 6.2 does not allow a blank name to be specified as a value on the LOGMODE keyword. For example, if the application issues an APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS and the LOGMODE value is blanks, the macroinstruction will fail with an RCPRI,RCSEC of X'002C',X'0001' PARAMETER_ERROR_INVALID_MODE.

After a successful CNOS negotiation, the application might attempt to allocate a conversation with the partner using the name and mode just negotiated. If this occurs, VTAM will internally issue the equivalent on the SIMLOGON macroinstruction to start the session for the conversation to use. If the logmode name specified on the APPCCMD CONTROL=ALLOC, QUALIFY=ALLOCD and subsequently used on the SIMLOGON is not defined[1] with the VTAM that owns the secondary logical unit (SLU), the SIMLOGON fails with a SNA sense code of X'08210002'. However, when VTAM receives this sense code, it reissues the internal version of the SIMLOGON with a logmode of blanks. When this macroinstruction completes with the default logmode values in the session initiation information (for example, the BIND image in the CINIT) then VTAM's support for LU 6.2 alters these values as specified in Table 26 on page 125.

To summarize, after a successful CNOS, VTAM successfully activates the session even if the VTAM that owns the SLU does not contain a definition for the logmode name specified.

Table entries

An LU-mode table has two kinds of entries:

- LU entries

  Each LU entry holds information about a partner LU. The entries for all partner LUs for a specific LU are associated with each other. LU entries, once placed in the table, cannot be changed by APPCCMD QUALIFY=OPRCNTL, QUALIFY=DEFINE, or CNOS.

- Mode entries

  Each mode entry holds information about one mode associated with one partner LU. All the mode entries for the same partner LU are associated with each other, and the collection of mode entries is associated with the LU entry.

Figure 11 on page 45 shows a conceptual view of the LU-mode table entry. Table 4 on page 45 and Table 5 on page 45 show how the session parameters are used for session activation.

[1] Defined in this sense means that either the logon mode name is defined by the MODEENT macroinstruction of a logmode table or that other system parameters from the IBM-supplied table, ISTINCLM, have overridden the blank mode name. See “Logon mode table” on page 93 for more information.

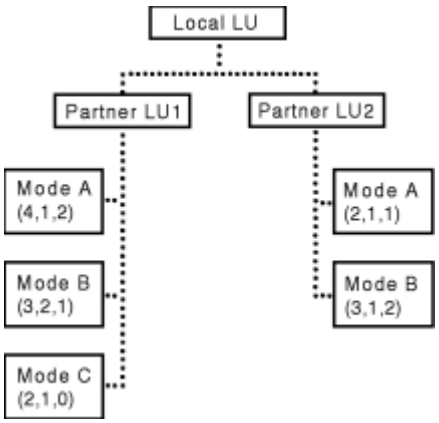


Figure 11. Conceptual view of LU-mode table structure

The local LU has the possibility of establishing sessions with partner LU1 or partner LU2.

The session establishment with partner LU1 can use the session parameters specified in the logon mode table for Mode A, Mode B, or Mode C.

Table 4. Session establishment with partner LU1

Mode Group	Pool Size	Results (from the local LU's point of view)
A	4	One session is guaranteed to be a contention winner; two sessions are guaranteed to be contention losers. The unspecified session can be activated as a contention winner by either LU.
B	3	Two sessions are guaranteed to be contention winners; one session is guaranteed to be a contention loser.
C	2	One session is guaranteed to be a contention winner. The unspecified session can be activated as a contention winner by either LU.

The session establishment with partner LU2 can use the session parameters specified in the logon mode table for Mode A or Mode B.

Table 5. Session establishment with partner LU2

Mode Group	Pool Size	Results (from the local LU's point of view)
A	2	One session is guaranteed to be a contention winner; one session is guaranteed to be a contention loser.
B	3	One session is guaranteed to be a contention winner; two sessions are guaranteed to be contention losers.

Mode A can be defined in both partner LUs and have the same session limits or different session limits. Also, the mode names defined can be different for the partner LUs.

Initializing the LU-mode table

LU 6.2 architecture specifies that the partner LU and mode name pairs be explicitly defined during system definition. By dynamically defining LU and mode entries for the application program, VTAM removes the burden of additional system definition and coordination that is required by the architectural

implementation of the LU-mode table. See “VTAM's LU-mode table” on page 52 for more discussion about updating information in the LU-mode table.

When an LU-mode pair is first added to the LU-mode table, VTAM associates it with the following LU 6.2-specific operands that are supplied on the application program's APPL definition statement:

- ATNLOSS
- AUTOSES
- DDRAINL
- DMINWNL
- DMINWNR
- DRESPL
- DSESLIM
- LIMQSINT
- LMDENT
- OPERCNOS
- SECACPT
- SYNCLVL
- VERIFY

Refer to z/OS Communications Server: SNA Resource Definition Reference for a description of these operands.

After VTAM places the LU-mode pair in the table, the values determined by the APPL definition statement can be changed for only the following operands:

- AUTOSES
- DDRAINL
- DMINWNL
- DMINWNR
- DRESPL
- DSESLIM

The MODIFY DEFINE operator command and the APPCCMD CONTROL=OPRCNTL, QUALIFY=DEFINE macroinstruction explain how to change these values. Refer to z/OS Communications Server: SNA Operation for a description of the MODIFY DEFINE command.

Chapter 4. Designing programs to use LU 6.2 services

About this chapter

The organization of a VTAM application program affects how much storage it uses, how well it performs, how easy it is to write, and how easy it is to migrate to a later release of VTAM, if required.

z/OS Communications Server: SNA Programming contains VTAM programming considerations for organizing an application program. Refer to that book for the following issues that affect LU 6.2:

- Processing synchronously versus asynchronously
- Using ECBs versus RPL exit routines
- Interfacing with the operating system
- Managing control block storage
- Easing migration to later VTAM releases

This chapter covers the following additional considerations that are unique to LU 6.2:

- "Request of LU 6.2 services" on page 47
- "Startup processing for LU 6.2 application programs" on page 48
- "LU 6.2 transaction processing" on page 54
- "Implementing LU 6.2 option sets" on page 63

Note: The first three items in the list of unique LU 6.2 considerations must be addressed. The application program may also implement optional LU 6.2 functions. These depend on the specific application in which the LU 6.2 architecture is being used.

Request of LU 6.2 services

To request LU 6.2 services, the application program uses the RPL-based APPCCMD macroinstruction (in conjunction with at least the VTAM OPEN, CLOSE, and SETLOGON macroinstructions). The RPL extension provides feedback and return code information to the application program.

RPL extension user field

The USERFLD field of the RPL extension enables the application program to uniquely identify an RPL. This field could contain the address of a data structure that represents a conversation. With the RPL extension, the application program can handle a series of input and output actions for a particular conversation. For example, the application program might associate a conversation request with the address of a user-defined data structure that contains information about the conversation that the application program needs.

The USERFLD field contains space for 4 bytes of information. VTAM saves the information that is placed in the field at the time a conversation is established. This information is available to the application program in the RPL extension specified for a particular macroinstruction.

Evaluating feedback information

Feedback information is found in the registers and in the return codes. For more information, see "General sequence of error checking" on page 263.

Registers

The starting points for checking feedback from an APPCCMD macroinstruction are registers 15 and 0. Register 15 contains the general return code from VTAM. Only a few values are defined for the general

return code, and it gives a quick indication of whether the macroinstruction was successful. Register 0 contains either a conditional completion return code or recovery action return code. In the case of errors, it provides more detail on the cause of the error.

If both register 15 and register 0 are 0, the APPCCMD has completed or been accepted without error. If register 15 is 0 but register 0 contains X'000000B', the request completed conditionally.

Return codes

APPCCMD macroinstructions can pass back any of a large number of return codes in the RPL extension. Each individual macroinstruction description in the z/OS Communications Server: SNA Programmer's LU 6.2 Reference includes a list of applicable return codes. In addition to describing error situations, many of the return codes also indicate whether the error is likely to recur. For example, an RCPRI, RCSEC return code combination of X'0004', X'0001' on an ALLOC macroinstruction indicates that the macroinstruction failed because of a temporary condition. The application program can attempt to allocate the conversation again. Return codes also indicate whether data purging or truncation occurred in an error situation and whether attempts to send or receive data should be repeated.

The extent to which the application program screens return codes depends upon its needs and the extent of the error handling it must provide. Some of the return codes indicate temporary conditions. Therefore, you should analyze the possible return codes to determine which ones the application program should handle.

The mechanism for handling return codes depends on the application program. The application program can include inline code for dealing with selected return codes, or it can include error-handling subroutines. The primary return codes passed back in the RPL extension field RCPRI are multiples of four and can be used to index into a branch table.

VTAM also uses feedback from an APPCCMD macroinstruction to pass other important information to the program. VTAM uses fields in the RPL extension to indicate to the application program that a conversation request has been received from a partner LU and that a request-to-send has been received from a partner LU. When an error is reported, a field in the RPL extension indicates whether error log data follows. The application program should check these fields and respond appropriately.

Startup processing for LU 6.2 application programs

Some of the functions associated with an LU 6.2 application program are used only during startup. They are:

- Opening an ACB
- Issuing a SETLOGON instruction
- Restoring modes and any associated persistent LU-LU sessions
- Negotiating session limits

Opening an ACB

An LU 6.2 application program must issue an OPEN macroinstruction, which points to an ACB, to formally present itself to VTAM. LU 6.2 application programs must also be defined to VTAM using APPC=YES on the APPL definition statement. The OPEN ACB also plays a role in the recovery of a persistent LU-LU session.

LU 6.2 applications can inform VTAM about application capabilities during OPEN ACB by building an application-ACB vector list. The address of the application-ACB vector list is indicated on the PARMS=APPLVCTR parameter of the ACB macroinstruction. For more information about the application-ACB vector list, see “Vector lists used during OPEN processing” on page 21.

Refer to z/OS Communications Server: SNA Programming for detailed information about opening an ACB.

Issuing a SETLOGON macroinstruction

An LU 6.2 program must issue the SETLOGON OPTCD=START macroinstruction before VTAM can activate any sessions managed by VTAM. Session activation can be initiated, but not completed, until the SETLOGON is issued. Therefore, session activations that are initiated as a result of APPCCMD CONTROL=CNOS, CONTROL=ALLOC, or CONTROL=PREALLOC macroinstructions are queued until the SETLOGON is issued. The APPCCMD does not complete while the activation is queued.

Table 6 on page 49 summarizes how SETLOGON OPTCD=START affects session initiation.

Table 6. Impact of SETLOGON on session establishment

Application's Capability	Before SETLOGON OPTCD=START	After SETLOGON OPTCD=START
For the establishment of sessions initiated by VTAM to satisfy conversation requests.	All CINITs are queued.	CINITs either: • Schedule the LOGON exit. • If a LOGON exit has not been provided, then they are accepted by VTAM for the application program if the application has specified APPC=YES and this is an LU 6.2 session.

The SETLOGON macroinstruction is also used to control other characteristics of an application program. Two types of the SETLOGON macroinstruction, OPTCD=PERSIST and OPTCD=NPERSIST, permit the persistent LU-LU session function to be dynamically enabled and disabled. The PERSIST option enables VTAM to retain modes and any associated sessions when an application program fails. The NPERSIST option disables that capability. In addition, a timer can be specified with the PERSIST option to control the maximum amount of time that can elapse between a failure and a recovery.

The SETLOGON macroinstruction with OPTCD=GNAMEADD and OPTCD=GNAMEDEL may be used in a sysplex environment to allow an application to support a generic resource name. For more information about the generic resources function, refer to z/OS Communications Server: SNA Network Implementation Guide.

Refer to z/OS Communications Server: SNA Programming for detailed information on the SETLOGON macroinstruction.

Restoring modes and any associated persistent LU-LU sessions

Persistent LU-LU session support can be used by a VTAM application program to facilitate recovery after a failure, or to manage a planned takeover. There are two types of persistent session support.

Single-Node persistent session support is used to help recover sessions that are disrupted by an application failure. When a VTAM application program fails after it has enabled persistence, VTAM retains the LU-LU sessions. In order for sessions to be retained as single-node persistent sessions, VTAM and the operating system must remain active.

Multinode persistent session support is used to help recover sessions that are disrupted by a node failure, such as a failure in the hardware, operating system or VTAM. Multinode persistent session support is available in a sysplex environment. Information used to reestablish the LU-LU session environment with their associated modes is maintained in the coupling facility.

After the recovery of the application program, these retained sessions and modes must be restored to permit continued use. (Refer to the information about using persistent LU-LU session support in z/OS Communications Server: SNA Programming for more information about how the two types of session persistence can be used to recover from unplanned or planned outages.)

Establishing persistence

An application program indicates that it is capable of persistence by specifying PARMS=(PERSIST=YES) on the ACB macroinstruction. In addition, if the application is to be capable of Multinode persistent sessions as well as single-node persistence, PERSIST=MULTI must be specified on the APPL definition statement for the application program and it must be operating in a sysplex environment. The application program enables persistence by specifying OPTCD=PERSIST on the SETLOGON macroinstruction. At any time, the application program can disable persistence by issuing SETLOGON OPTCD=NPERSIST. (Refer to z/OS Communications Server: SNA Programming for more information.)

Note: An application program must be capable of persistence before it can enable persistence. Throughout this book, the term "enabled" (relative to persistence) implies that the application program is also capable of persistence.

Managing the recovery

Persistent LU-LU session support can be used to manage the recovery or the planned takeover of an application program that has enabled persistence.

When a failure occurs, the application program has several ways to manage the recovery:

- Another instance of the failing application program can restart, recover the connection to the active VTAM, and restore sessions.
- An alternate application program that has been started already can recover the connection to the active VTAM and restore sessions.
- Using multinode persistence, another instance of the application program can restart on the same node or a different node in the sysplex, connect to the VTAM on that node, and restore sessions.

Opening the ACB during recovery

During a failure, VTAM closes the ACB on behalf of the application program but retains the LU-LU sessions and modes. The recovering application program opens its ACB specifying PARMS=(PERSIST=YES), and VTAM reconnects the failed application program.

When a single-node persistent session planned takeover occurs, the takeover application program connects to VTAM and restores sessions, even though the original application program has not failed. The takeover application program opens its ACB specifying PARMS=(PERSIST=YES). VTAM schedules the TPEND exit of the original application program to notify that application program to issue a CLOSE and then closes the original application program. The original application program issues a CLOSE to handle cleanup. If the original application program does not have a TPEND exit, VTAM closes it without any notification. For a multinode persistent session planned takeover, the application must close its ACB before it can be reopened on the same node or a different node in the sysplex.

The failure of an application program starts the safety timer, if one was specified on the SETLOGON OPTCD=PERSIST request. If the timer expires before a recovering application program issues an OPEN ACB, VTAM terminates the retained sessions as if persistence had not been enabled and releases the held resources. For more details, refer to z/OS Communications Server: SNA Programming.

States of recovery

After a failure, a VTAM application program that has enabled persistent LU-LU session support is in one of three recovery states:

Recovery Pending
The application program fails, and the recovering application program has not opened its ACB.

Recovery-in-Progress
The recovering application program has opened its ACB, and sessions are pending recovery. The application program remains in this state until the last session is either restored or terminated.

Recovery Complete
The application program has restored all modes.

Note: VTAM retains a mode even if it has no associated sessions at the time of the failure. The mode must be restored before it can be used.

Activity during a failure

During a failure, VTAM places any existing modes and active LU-LU sessions in a recovery pending state and deallocates all active conversations. If VTAM cannot communicate the deallocation to the partner LU, the deallocation is delayed until the conversation state permits VTAM to send the deallocation.

Any data associated with a deallocated conversation is discarded. Newly arriving FMH-5s and any associated data received during the outage are queued for delivery following recovery.

Pending and queued sessions are terminated. If an application program has enabled persistence and fails (or is taken over) during a critical exchange of information on a session, such as a synchronization exchange, VTAM unbinds that session. The unbind permits the LUs to make consistent decisions and ensures continued synchronization between the two LUs.

Restoring resources

After the recovering application opens its ACB, the application program can use the APPCCMD CONTROL=OPRCNTL, QUALIFY=RESTORE macroinstruction for any modes that need to be restored. The SETLOGON START macroinstruction must be issued before the RESTORE macroinstruction. When all modes are restored, the recovery is complete because VTAM restores all sessions that are pending recovery for a given LU and mode as it restores the mode.

Restoring a mode

A recovering (or takeover) application program can manage the restore process by specifying an LU and mode, an LU only, or neither an LU nor a mode. If neither an LU nor a mode is specified, VTAM restores each LU and its related modes.

To ensure that all LUs and modes are restored, at the end of recovery processing, the application program can issue the RESTORE macroinstruction without specifying an LU and mode. If X'0000' RCPRI and X'0006' RCSEC are received, all LUs and modes have been processed. (The meaning associated with this RCPRI,RCSEC combination is that the RESTORE was unnecessary.)

During the restore process, VTAM, if requested, provides information that describes the LUs, modes, and sessions that are being restored. The application program specifies the level of information that is to be returned in the RESTORE control block by using the LIST keyword in the APPCCMD CONTROL=OPRCNTL, QUALIFY=RESTORE macroinstruction. For more details on the control block, see “Retrieving information for a mode and sessions to be restored” on page 147.

In addition to using the APPCCMD CONTROL=OPRCNTL, QUALIFY=RESTORE macroinstruction, you can take other actions for modes that are pending recovery.

- The following APPCCMDs can be issued:
  - APPCCMD CONTROL=OPRCNTL, QUALIFY=DISPLAY
  - APPCCMD CONTROL=REJECT, QUALIFY=SESSION
  - APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS for an LU that was not a partner at the time of the failure
  - APPCCMD CONTROL=RECEIVE, QUALIFY=ANY, even if modes are pending recovery, because this macroinstruction is not specific to a mode
- New modes can be added after the SNASVCMG mode for an LU has been restored, but any mode that exists when the failure (or takeover) occurs cannot be used until that mode has been restored.
- When the application program's LOGON or SCIP exit is driven, the APPCCMD CONTROL=OPRCNTL, QUALIFY=ACTSESS and APPCCMD CONTROL=OPRCNTL, QUALIFY=DACTSESS macroinstructions can be issued.

Negotiating session limits

The factors in the negotiation of session limits are discussed in this section. For details about the negotiation, see Chapter 6, “Managing sessions,” on page 83.

VTAM's LU-mode table

VTAM maintains the LU-mode table to record partner-LU names and mode-name groups associated with each LU. (See “Mode name groups” on page 7 and “VTAM LU-mode table” on page 43, respectively, for more information on mode name groups and LU-mode pairs.) Application programs can use APPCCMD CONTROL=OPRCNTL, QUALIFY=DISPLAY to display values in the table and APPCCMD CONTROL=OPRCNTL, QUALIFY=DEFINE to change negotiation values in the table. VTAM can determine when entries are added and deleted from the table, but the application program also can control entries and deletions with the DEFINE macroinstruction. (See “Adding to the LU-mode table” on page 94 for more discussion about updating information in the LU-mode table.)

Defining logon mode names

Application programs define logon mode names to establish characteristics for sessions with various partner LUs. The existing VTAM logon mode table holds logon mode name definitions. The application program refers to these definitions by using the LOGMODE operand on the APPCCMD macroinstruction. VTAM constructs the session parameters for a session, using the information in the logon mode table. Session parameters are carried in the BIND image used to set up a session.

IBM supplies a definition for the SNASVCMG mode name, which can be used as a model for defining other logon mode names. Refer to z/OS Communications Server: SNA Resource Definition Reference for information on creating entries in the logon mode table. IBM also supplies a definition for the CPSVCMG mode name, which is reserved.

Note: Do not confuse the logon mode table with the LU-mode table maintained by VTAM for the application program. The logon mode table defines a set of session characteristics for each logon mode name. The LU-mode table associates logon mode name groups with partner LUs.

Keep a SNASVCMG mode name definition in the logon mode table. Even if parallel sessions are not used, VTAM can still attempt to establish a conversation on this logon mode when establishing the initial session with a single-session partner, if the partner is not specified as single-session capable.

Single-session partners

Although most application programs that use LU 6.2 protocols are capable of parallel sessions, some are not. (Application can refer to code that implements LU 6.2 in devices in the network.) A number of limitations apply to single-session partners.

Session limits designating local and remote contention-winner sessions cannot exceed 1 for any mode name group for a single-session partner. Only one mode name group at any given time can have nonzero session limits because only one session can be active at a time. Before it issues an ALLOC or PREALLOC using a second mode name, the application program must issue a CNOS request setting the session limits for the first mode name to 0, which causes the session to be deactivated when the conversation ends. It then issues a CNOS request for the second mode name.

When an application program issues a CNOS request, session limits are negotiated if the partner is capable of parallel sessions. No negotiation takes place for single-session partners. Instead, all processing is done by VTAM in the local host system. For single-session devices, the session limits are always 0 or 1.

If the application program knows that the partner LU is a single-session LU, it can specify this information in the CNOS request. This prevents the initiation of a SNASVCMG session for initial CNOS requests involving the partner LU. A session partner that does not support parallel sessions can initiate an LU 6.2 session as an SLU. The mode entry used must specify that parallel sessions and CNOS are not supported. Reference the byte 'BINFLG2' at hex offset 17 in the BIND DSECT. (Refer to z/OS Communications Server: SNA Programmer's LU 6.2 Reference for this byte.)

Note: When VTAM receives a BIND request for a single-session-capable LU and no LU-mode entry exists for the LU, VTAM creates an LU-mode entry with session limits of (1,0,0) for the LU. Because of these limits, the application program receiving the BIND cannot successfully issue an APPCCMD CONTROL=ALLOC, QUALIFY=CONWIN macroinstruction once the session is available.

SNASVCMG mode name group

As with single-session partners, special considerations apply to the SNASVCMG mode name group used by VTAM for control functions. The application program should not use SNASVCMG sessions for conversations. SNASVCMG should be specified by the application program *only* for control operator functions or other transaction programs required to use this mode name. The only valid session limits for SNASVCMG are two sessions (with each partner assured one contention-winner session) or limits of 0. SNASVCMG cannot be specified as a mode name on a CNOS request for a single-session partner or on an APPCCMD CONTROL=OPRCNTL, QUALIFY=DEFINE macroinstruction.

Initializing session limits

An application program can attempt to initialize session limits with APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS before VTAM has determined the partner LU's session capability. If the partner LU is single-session capable, the session request sent by VTAM returns that information. (For information on how a partner LU can return this indication, see “Session limits for single-session-capable partners” on page 119.) VTAM determines the session-capability of the partner from the response, and the CNOS request completes with an RCPRI, RCSEC combination of X'0000', X'0004', indicating that the partner LU is single-session capable. The session limits are, however, initialized. If session limits greater than 1 are specified on the macroinstruction or if no session limits are specified, the default values of (1,0,0) are used.

An installation should always include the IBM-supplied entry for the SNASVCMG mode in its logon mode tables. Even if no parallel sessions are planned for an application program, VTAM can still use this mode as part of its processing for session limits with partner LUs whose session capability is unknown.

Note: The SNASVCMG logon mode is included in the logon mode table provided by IBM with the VTAM system.

As with CNOS processing for a single-session partner, no CNOS negotiation is done for the SNASVCMG mode. The local LU handles all CNOS processing. The SNASVCMG session limits cannot be reset to 0 until all other mode name groups with the partner LU have session limits of 0. VTAM deactivates the SNASVCMG session only after all other sessions to the partner LU have been deactivated. No draining of pending conversation allocation requests is to take place on any of these SNASVCMG sessions for the local LU.

Impact on conversation initiation

The requests issued to start a conversation differ depending on whether an application program requires explicit control of the SNASVCMG session. The application program can control the activation and deactivation of the SNASVCMG session through the APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS request, or it can let VTAM automatically activate the SNASVCMG session. VTAM activates the SNASVCMG session if *all* of the following conditions are met.

- The application program issues APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS for a mode name other than SNASVCMG.
- The SNASVCMG session is not active.
- The application program has not specified on the CNOS request that the partner LU is single-session capable.

When the application program controls the activation of the SNASVCMG session, the first APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS (where LOGMODE=SNASVCMG) for a partner LU indicates that the partner LU should support parallel sessions. VTAM does not negotiate the activation with the partner LU. If a subsequent BIND is received from the partner LU indicating single-session capability, VTAM rejects the BIND.

Application programs cannot issue APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS for the SNASVCMG mode name if the partner LU is not parallel-session capable. Partner-LU parallel-session capability cannot be changed once it is established by VTAM.

Session initiation and termination

Application programs have little explicit control over whether sessions are initiated. They use CNOS requests to control session limits. These requests cause VTAM to activate or deactivate sessions.

Application programs can explicitly accept or reject session establishment requests in the LOGON and SCIP exit routines by using CONTROL=OPRCNTL, QUALIFY=ACTSESS or CONTROL=OPRCNTL, QUALIFY=DACTSESS. These exits must be present in the application program for this capability to exist. The LOGON and SCIP exits also enable the application program to change session parameters by specifying an alternate BIND or BIND response.

A session partner that does not support parallel sessions can initiate a 6.2 session as an SLU. The mode entry used must specify that parallel sessions and CNOS are not supported. Reference the byte 'BINFLG2' at hex offset 17 in the BIND DSECT. (Refer to z/OS Communications Server: SNA Programmer's LU 6.2 Reference for this byte.)

After VTAM establishes the session, the application program can request rejection termination of the session by issuing the APPCCMD CONTROL=REJECT, QUALIFY=SESSION macroinstruction. See Chapter 6, “Managing sessions,” on page 83 for more details on controlling session limits.

Application termination

There is a field in the RPL extension called RPL6LAST. This field is completed by APPC/VTAM when it schedules the ATTN(LOSS) exit. RPL6LAST is a 2–bit field that indicates whether the session being terminated is:

- Not the last session to this LU for this particular modename. (Other sessions remain active for this modename.)
- The last session to this LU for this particular modename.
- The last session to this LU for all non-control mode modenames. (SNASVCMG and CPSVCMG are control modes).
- The last session to this LU for all modenames.

The safest way to terminate an application is to perform the following tasks:

1. Issue an APPCCMD (DEFINE) to set session limits to 0 for all modes. This stops a CNOS from the partner from starting sessions successfully.
2. Issue an APPCCMD (CNOS, ALL) to set session limits to 0 for all non-control mode modenames.
3. When RPL6LAST indicates all non-control mode sessions are ended, issue an APPCCMD (CNOS) to set SNASVCMG session limits to 0.
4. Repeat these steps for all partner LUs.
5. When RPL6LAST indicates all sessions are ended for all LUs, CLOSE the ACB.

LU 6.2 transaction processing

Transaction processing for LU 6.2 is achieved by means of conversations on established sessions. Transaction processing for LU 6.2 includes the following steps:.

- Allocating a conversation and receiving the allocate
- Sending and receiving data
- Deallocating the conversation

Understanding conversations

A conversation is the communication mechanism between two transaction programs. The conversation may appear different to each transaction program. While the appearance of the conversation differs, the views of each conversation partner are complementary. The appearance of a conversation is described by the *state* of the conversation. Whenever the term *conversation state* is used in this book, it refers to the local view of the conversation. Just as the term "half-session" is used to describe the local state of the session, "conversation state" is used to describe the local view of the conversation. See Appendix A, “Conversation states,” on page 281 for more information.

Synchronous nature of conversations

Conversations are single-thread processes. Most APPCCMD macroinstructions cannot be issued for a conversation until the previous one completes. This is independent of whether the APPCCMD macroinstruction is issued synchronously or asynchronously.

Each conversation is considered to be one thread in a multithread program.

Maintaining conversation states

LU 6.2 architecture defines the status of a conversation at any given time in terms of a finite state machine. The states define conditions such as whether the application program is sending or receiving data, or ending a conversation. VTAM maintains a finite state machine internally to track changes in a conversation's status. As the name implies, only a limited number of states are possible. While in a given state, application programs are restricted to a subset of APPCCMD macroinstructions. VTAM always tracks the state of the conversation and prevents application programs from issuing macroinstructions not allowed in a specific state.

The state of a conversation is somewhat predictable. For half-duplex conversations, the states for each side of the conversation are complementary. When one side of the conversation is in SEND state, the other side of the conversation is in RECEIVE state. The SEND state implies that the LU on the side of the conversation in this state can send data. The RECEIVE state implies that the LU on the side of the conversation in RECEIVE state can receive data. When the application program on the SEND side of a half-duplex conversation issues a RECEIVE macroinstruction, the application program's side of the conversation enters RECEIVE state and the other side of the conversation enters SEND state.

For full-duplex conversations, both sides of the conversation maintain a SEND/RECEIVE state, because both conversation partners can send and receive information concurrently. When an APPCCMD CONTROL=DEALLOC|DEALLOCQ macroinstruction is issued, the conversation initiating the deallocation enters RECEIVE_ONLY state. The conversation partner's side of the conversation is placed in SEND_ONLY state as soon as the deallocation request is received.

Application programs do not have to implement their own finite state machine to track a conversation. Many of the states are of interest only to VTAM. When designing the application program, however, be aware of which state the conversation is in and which APPCCMD macroinstructions the application program can issue from that conversation state. (For example, normal deallocation of a half-duplex conversation could not be initiated by the application program while the conversation is in RECEIVE state.) You can code the application program to include flags to prevent the application program from attempting to issue invalid macroinstructions, or you can implement your own version of a finite state machine.

Information about the current conversation state can be obtained several ways. VTAM returns the current conversation state in the RPL6CCST (CONSTATE) field of the ISTRPL6X on many of the APPCCMD macroinstructions. The APPCCMD CONTROL=TESTSTAT, QUALIFY=SPEC|ISPEC macroinstruction can also return conversation state information. The following sections provide further information about conversation states.

- For a list of macroinstructions that set the CONSTATE field, see “Keywords and returned parameters” on page 72.
- Refer to z/OS Communications Server: SNA Programmer's LU 6.2 Reference for the ISTRPL6X DSECT.

- See “Querying the current status of a conversation” on page 56 for more information on TESTSTAT.
- For more information on conversation states and the APPCCMD macroinstructions that cause changes in these states, see Appendix A, “Conversation states,” on page 281.
- The z/OS Communications Server: SNA Programmer's LU 6.2 Reference lists state changes as a result of nonzero return codes on APPCCMD macroinstructions.

Querying the current status of a conversation

The APPCCMD CONTROL=TESTSTAT macroinstruction provides a variety of status information about a conversation to the application. The application is able to query the information on a specific conversation, as well as for all conversations for a particular ACB.

Types of information returned by this macroinstruction include the following items:

- CONVID of the conversation being reported.
- Amount of the normal data waiting to be received, including the LL field and the LL remainder.
- Amount of expedited data waiting to be received.
- Whether a REQUEST_TO_SEND indication is waiting to be received.
- Whether the conversation is in CA mode for receiving normal information.
- Whether the conversation is in CA mode for receiving expedited information.
- State of the conversation reported. (The conversation state is reported only when QUALIFY=SPEC|ISPEC is specified.)

The APPCCMD CONTROL=TESTSTAT, QUALIFY=ALL|IALL macroinstructions return status information from any active conversation for the specified ACB. The APPCCMD CONTROL=TESTSTAT, QUALIFY=ALL macroinstruction returns any status that is immediately available. If no information is currently available, the LU waits until status information is received before completing. The APPCCMD CONTROL=TESTSTAT, QUALIFY=IALL macroinstruction also returns status on any immediately available information, but if no information is currently available, the macroinstruction completes with an RCPRI, RCSEC combination of X'0000', X'0008' NO_IMMEDIATELY_AVAILABLE_INFORMATION.

The APPCCMD CONTROL=TESTSTAT, QUALIFY=SPEC|ISPEC macroinstructions provide status information for a specific conversation. The APPCCMD CONTROL=TESTSTAT, QUALIFY=SPEC macroinstruction returns any information immediately available. If no information is immediately available, the LU waits for this information before the macroinstruction completes. The APPCCMD CONTROL=TESTSTAT, QUALIFY=ISPEC macroinstruction returns status information that is immediately available. If no information is available, the macroinstruction completes with an RCPRI, RCSEC combination of X'0000', X'0008' NO_IMMEDIATELY_AVAILABLE_INFORMATION.

For more information on the APPCCMD CONTROL=TESTSTAT, refer to the macroinstructions in z/OS Communications Server: SNA Programmer's LU 6.2 Reference.

Conversation queues for macroinstruction processing

Conversation queues are used to store macroinstructions until their processing is complete. A conversation queue, in general, is used to store macroinstructions that perform a specific function. For example, a macroinstruction that sends expedited information is stored on the EXPEDITED SEND conversation queue until processing of the macroinstruction is complete.

VTAM limits the conversation queues to a maximum of one macroinstruction outstanding on a conversation queue at a time. If an APPCCMD macroinstruction is issued when that queue is at its limit (that is, a macroinstruction is already being held), it is rejected with an RCPRI, RCSEC combination of X'002C', X'0011', PARAMETER_ERROR— PREVIOUS_MACROINSTRUCTION_OUTSTANDING. The following macroinstructions are exceptions and may be allowed to process if there is a macroinstruction already outstanding on the queue:

- APPCCMD CONTROL=DEALLOC|DEALLOCQ, QUALIFY=ABNDPROG
- APPCCMD CONTROL=DEALLOC|DEALLOCQ, QUALIFY=ABNDSERV

- APPCCMD CONTROL=DEALLOC|DEALLOCQ, QUALIFY=ABNDTIME
- APPCCMD CONTROL=DEALLOC|DEALLOCQ, QUALIFY=ABNDUSER
- APPCCMD CONTROL=REJECT, QUALIFY=CONV

VTAM also allows only one APPCCMD CONTROL=TESTSTAT, QUALIFY=ALL|IALL to be outstanding at a time. If one of these macroinstructions is outstanding, the issuance of another one will be rejected with an RCPRI, RCSEC combination of X'002C', X'0011', PARAMETER_ERROR— PREVIOUS_MACROINSTRUCTION_OUTSTANDING.

The following macroinstructions are queue-independent; their acceptance is not related to a particular existing conversation:

- APPCCMD CONTROL=ALLOC
- APPCCMD CONTROL=CHECK
- APPCCMD CONTROL=OPRCNTL
- APPCCMD CONTROL=PREALLOC
- APPCCMD CONTROL=RCVFMH5, QUALIFY=NULL|QUEUE
- APPCCMD CONTROL=REJECT, QUALIFY=SESSION
- APPCCMD CONTROL=REJECT, QUALIFY=CONVGRP
- APPCCMD CONTROL=SETSESS

The APPCCMD CONTROL=RCVFMH5, QUALIFY=DATAQUE macroinstruction is initially queue-independent but moves to the SEND/RECEIVE queue for the receive function.

VTAM has no limit to the number of APPCCMD CONTROL=ALLOC|PREALLOC macroinstructions an application program can issue.

At certain points during conversation deallocation, VTAM prohibits further macroinstructions from being accepted for the conversation queues. The application program will then receive an RCPRI, RCSEC combination of X'00A0', X'0002', REQUEST_NOT_ALLOWED— REQUEST_BLOCKED, on any macroinstruction issued that is directed to one of the conversation queues. This return code indicates that the conversation is in the process of being deallocated or terminated and no other processing will be accepted for the conversation.

Queues for half-duplex conversations

There are four conversation queues associated with a half-duplex conversation:

- SEND/RECEIVE queue
- EXPEDITED SEND queue
- EXPEDITED RECEIVE queue
- TESTSTAT queue

VTAM allows macroinstructions to be outstanding on all four queues simultaneously.

The SEND/RECEIVE queue is used to store macroinstructions that either send or receive normal information. The following macroinstructions are stored on the SEND/RECEIVE queue when issued by the application program:

- APPCCMD CONTROL=SEND, QUALIFY=CONFIRM|CONFIRMD| DATA|DATACON|DATAFLU|ERROR|FLUSH
- APPCCMD CONTROL=PREPRCV
- APPCCMD CONTROL=SENDRCV
- APPCCMD CONTROL=RECEIVE
- APPCCMD CONTROL=DEALLOC
- APPCCMD CONTROL=DEALLOCQ
- APPCCMD CONTROL=REJECT, QUALIFY=CONV

- APPCCMD CONTROL=RESETRCV
- APPCCMD CONTROL=SENDFMH5

The APPCCMD CONTROL=RECEIVE, QUALIFY=ANY|IANY macroinstructions are placed on the queue when they are matched to information for the conversation.

The EXPEDITED SEND queue is used to store macroinstructions that send expedited information. The following macroinstructions are stored on the EXPEDITED SEND queue when issued by the application program:

- APPCCMD CONTROL=SEND, QUALIFY=RQSEND
- APPCCMD CONTROL=SENDEXPD, QUALIFY=DATA

The EXPEDITED RECEIVE queue is used to store macroinstructions that receive expedited information. The following macroinstructions are stored on the EXPEDITED RECEIVE queue when issued by the application program:

- APPCCMD CONTROL=RCVEXPD, QUALIFY=ANY|IANY
- APPCCMD CONTROL=RCVEXPD, QUALIFY=SPEC|ISPEC

The APPCCMD CONTROL=RCVEXPD, QUALIFY=ANY|IANY macroinstructions are placed on the queue when they are matched to information for the conversation.

The TESTSTAT queue is used to store macroinstructions that query the conversation. The following macroinstructions are stored on the TESTSTAT queue when issued by the application program:

- APPCCMD CONTROL=TESTSTAT, QUALIFY=SPEC
- APPCCMD CONTROL=TESTSTAT, QUALIFY=ISPEC

The APPCCMD CONTROL=TESTSTAT, QUALIFY=ALL|IALL macroinstructions are not placed on the queue because they pertain to all conversations for an application rather than being specific to a particular conversation.

Queues for full-duplex conversations

There are five conversation queues associated with a full-duplex conversation:

- SEND queue
- RECEIVE queue
- EXPEDITED SEND queue
- EXPEDITED RECEIVE queue
- TESTSTAT queue

VTAM allows macroinstructions to be outstanding on all five queues simultaneously. Because sending and receiving is allowed concurrently, the SEND/RECEIVE queue is replaced, on a full-duplex conversation, with a SEND and RECEIVE queue.

The SEND queue is used to store macroinstructions that send normal information. The following macroinstructions are stored on the SEND queue when issued by the application program:

- APPCCMD CONTROL=SEND, QUALIFY=DATA|DATAFLU|ERROR|FLUSH
- APPCCMD CONTROL=DEALLOC, QUALIFY=DATAFLU|FLUSH|ABNDPROG| ABNDSERV|ABNDTIME| ABNDUSER
- APPCCMD CONTROL=DEALLOCQ
- APPCCMD CONTROL=REJECT, QUALIFY=CONV
- APPCCMD CONTROL=RESETRCV
- APPCCMD CONTROL=SENDFMH5

The RECEIVE queue is used to store macroinstructions that receive normal information. The following macroinstructions are stored on the RECEIVE queue when issued by the application program:

- APPCCMD CONTROL=RECEIVE, QUALIFY=ANY|IANY
- APPCCMD CONTROL=RECEIVE, QUALIFY=SPEC|ISPEC

The EXPEDITED SEND queue is used to store macroinstructions that send expedited information. The following macroinstruction is stored on the EXPEDITED SEND queue when issued by the application program:

- APPCCMD CONTROL=SENDEXPD, QUALIFY=DATA

For a full-duplex conversation, the EXPEDITED RECEIVE and TESTSTAT queues have the same function and store the same macroinstructions as a half-duplex conversation.

Allocating a conversation and receiving the allocate

One of the partners initiates transaction processing by allocating a conversation. Its partner receives the notification of the conversation through the ATTN(FMH-5) exit and issues the RCVFMH5 to accept the conversation. (You have to receive the allocate and then deallocate the conversation if you want to reject the conversation. See Chapter 8, “Deallocating a conversation,” on page 167 for more information.) See Chapter 7, “Allocating a conversation,” on page 149 for detailed information about allocating a conversation.

Some functions used by LU 6.2 applications require use of the APPCCMD CONTROL=PREALLOC macroinstruction prior to sending the FMH-5. See “Reserving a session for a conversation” on page 162 for more information.

Types of conversation allocation

Application programs have several ways to allocate a conversation when using the APPCCMD CONTROL=ALLOC macroinstruction:

QUALIFY=ALLOCD
Requests that the conversation be assigned to any free session, regardless of whether the session is a contention-winner or contention-loser. If no sessions are available and session limits allow, VTAM activates another session and allocates the conversation to it. If session limits do not permit establishing a new session, VTAM queues the conversation request until a session becomes available.

QUALIFY=CONVGRP
Requests that the conversation be assigned to a specific session that is identified by a unique conversation group identifier. If that specific session exists but is not available, VTAM queues the request until that session becomes available. If that specific session does not exist, VTAM fails the allocation request.

QUALIFY=CONWIN
Requests that the conversation be assigned to a free contention-winner session, if one is available. If no contention-winner session is available and the session limits permit activating another contention-winner session, VTAM activates a contention-winner session and allocates the conversation to it. If session limits do not permit activating another contention-winner session, VTAM queues the allocation request until a contention-winner session becomes available.

QUALIFY=IMMED
Requests that the conversation be assigned to a free contention-winner session. If there is no contention-winner session, the allocation request fails. The conversation request can be satisfied only if a contention-winner session is available immediately.

The QUALIFY=IMMED form of the macroinstruction can complete faster than QUALIFY=ALLOCD.

QUALIFY=WHENFREE
Requests that the conversation be assigned to any free session, regardless of whether the session is a contention-winner or contention-loser. If no sessions are available and session limits allow, VTAM activates another session and allocates the conversation to that session, if no other conversation requests are queued ahead of this request. If a session is pending, VTAM queues the pending request until the request is met or until all pending session activation requests have been depleted. In other words, the allocation can be queued waiting for a session, but the application does not want

this allocation queued behind another conversation. If all pending session activation requests are depleted before the request is met, VTAM fails the request with an error code.

If optimal performance is crucial, design the application program so that contention-winner sessions are always available to it. See Chapter 6, “Managing sessions,” on page 83 for information about setting and changing session limits.

Notification of conversation requests

When a conversation request, represented as an FMH-5, is received, VTAM notifies the application program of the request. Application programs can use either of the following methods for receiving notification of the receipt of an FMH-5:

- If an ATTN exit is provided, VTAM schedules it when an FMH-5 is received.
- When an APPCCMD macroinstruction completes, VTAM sets a feedback field in the RPL extension to indicate that an FMH-5 is outstanding.

The system programmer can best determine the method to use. Application programs also can maintain an internal timer and issue a macroinstruction at regular intervals to receive FMH-5s that are outstanding.

Comparing normal information to expedited information

Systems Network Architecture (SNA) provides for the control of the flow of information through the network to prevent congestion. Session pacing is a mechanism available to the application that is used to accomplish this flow control. The implementation of flow control requires that, under certain conditions, the information be queued and not transmitted. This is the case with normal information. While this process is necessary and useful, there are times when the transaction program wants to send an indication to the partner and wants to ensure that the indication will not be queued along the session path. Expedited information may be used to meet this requirement.

Expedited information flows on the session path but is never queued if session pacing is in effect. The expedited information can bypass the normal information and is forwarded on to the conversation partner.

To summarize:

- Normal information is subject to flow control and may be queued at intermediate points in the session path.
- Expedited information is *not* subject to flow control and will *not* be queued at intermediate points in the session path.

To assist the application programmer, VTAM has provided separate macroinstructions for each type of information. These macroinstructions allow the application program to structure itself more logically by using expedited macroinstructions for expedited information and normal macroinstructions for normal information.

Normal information and expedited information both contain categories of data (information that can be transmitted by a conversation partner) and indications (information that can be sent by VTAM or intervening network applications). Each of these categories is described in the following section.

Comparing data to indications

VTAM provides intelligence or facts to the application, including data and control signals from the partner and some control signals generated locally by VTAM to report the status of the partner or the status of the connection to the partner. Information includes data transmitted to the application from the conversation partner. Information also includes indications that cause the application's APPCCMD CONTROL=RECEIVE macroinstruction to complete. Indications are initiated by the partner application by way of macroinstructions or they are initiated using the intervening network components.

This section defines the terms used to describe the categories of normal and expedited information that can be sent and received by the application.

The types of normal information differ from the types of expedited information. Normal information includes *normal data* and *normal indications*. Expedited information includes *expedited data* and *expedited indications*.

Normal information that can be provided by the application by way of an APPCCMD macroinstruction or reported to the application includes the following information:

- Normal data:
  - Normal transaction data (for example, logical records)
  - PS_Header data
  - Error data (for example, Log Data)
  - Function management data (for example, FMH-5)
- Normal indications:
  - The SEND indication
  - The DEALLOC indication
  - The CONFIRM indication
  - Error indications (for example, a conversation failure)
  - Other normal indications that do not cause a receive macroinstruction to complete:
    - Log data is waiting to be received—indicated by LOGRCV (RPL6RLOG) in the RPL extension.
    - An FMH_5 is waiting to be received—indicated by FMH5RCV (RPL6FMH5) in the RPL extension.

Expedited information that can be sent by the application or reported to the application includes:

- Expedited data:
  - Expedited transaction data
- Expedited indications:
  - The request to send received (RTSR) indication—SIGRCV (RPL6RSIG) and SIGDATA (RPL6SGNL).
  - Other expedited indications (for example, expedited data is waiting to be received—indicated by EXPDRCV (RPL6EXDR) in the RPL extension). Note that these do not cause a receive macroinstruction to complete.

Determining conversation status

The current status, or state, of a conversation can be determined by examining the RPL fields returned on the APPCCMD CONTROL=TESTSTAT, QUALIFY=ISPEC|SPEC macroinstruction.

An application program can also determine the status of a conversation by examining the RPL extension returned on the APPCCMD CONTROL=RECEIVE macroinstruction. The application program should first check the RCPRI field in the RPL extension to check that the RECEIVE completed successfully. If RCPRI is set to 0, the application program can then check the What-Received field and examine the send bit to see if the conversation has been placed in a sending state.

When the application program that is sending issues a RECEIVE, the receiver can become the sender without asking for permission to send. The receiver should check the What-Received field on every RECEIVE to determine whether the conversation has been placed in a sending state. The application program also can refer to RPL6CCST to determine the state of the conversation at the successful completion of any APPCCMD macroinstructions that specify a specific conversation.

Sending and receiving normal information

Once the conversation is allocated, the transaction programs issue SENDs and RECEIVEs to transfer the data available for the transactions. (For detailed information about sending and receiving data, see Chapter 9, “Sending information,” on page 177 and Chapter 10, “Receiving information,” on page 193.) Multiple APPCCMD macroinstructions pertaining to different conversations can be outstanding.

Receiving input without specifying a conversation

An application program might not know which conversation will have data ready to receive next. Rather than issuing an APPCCMD CONTROL=RECEIVE, QUALIFY=SPEC|ISPEC macroinstruction for each conversation, VTAM provides a way for the application program to issue a single macroinstruction to receive data on the next conversation that has data available to be received.

To receive the next available data on any conversation, the conversations are first placed in continue-any mode with the CONMODE operand of an APPCCMD macroinstruction such as APPCCMD CONTROL=PREPRCV, QUALIFY=FLUSH. Then, the application program issues APPCCMD CONTROL=RECEIVE, QUALIFY=ANY|IANY.

Receiving next available data

The APPCCMD CONTROL=RECEIVE, QUALIFY=ANY macroinstruction completes when data arrives on any of the conversations that is in continue-any mode. When APPCCMD CONTROL=RECEIVE, QUALIFY=ANY completes, the conversation identifier (CONVID) field in the RPL extension is set to indicate on which conversation the data is received.

Typically, the APPCCMD CONTROL=RECEIVE, QUALIFY=ANY macroinstruction is issued with CONMODE=CS, which places the conversation in continue-specific mode after the APPCCMD CONTROL=RECEIVE, QUALIFY=ANY macroinstruction completes. Once the conversation is in continue-specific mode, only APPCCMD CONTROL=RECEIVE, QUALIFY=SPEC macroinstructions can be used to receive data on the conversation. The application program issues APPCCMD macroinstructions to process a transaction. When the transaction is complete, the application program can return the conversation to continue-any mode.

Receiving data immediately available

The APPCCMD CONTROL=RECEIVE, QUALIFY=IANY macroinstruction can also be used to receive data that is immediately available on a conversation in continue-any mode. If data is immediately available, the data is copied into the supplied data area or control block specified by the AREA parameter. The CONVID of the conversation sending the data is also returned. If no data is immediately available, the macroinstruction completes and an RCPRI, RCSEC combination of X'0000', X'0008' NO_IMMEDIATELY_ AVAILABLE_INFORMATION is returned to the application.

Sending and receiving expedited information

Conversations on full-duplex capable sessions support the sending and receiving of expedited information. If the underlying session is not full-duplex capable, an RCPRI,RCSEC combination of X'00A0', X'0001' REQUEST_NOT_ALLOWED— LU_PAIR_DOES_NOT_SUPPORT_SENDING_ EXPEDITED_DATA is returned to the application.

Sending expedited data

The APPCCMD CONTROL=SENDEXPD, QUALIFY=DATA macroinstruction sends expedited information. This macroinstruction completes immediately without waiting for a response from the conversation partner. This macroinstruction corresponds to the SEND_EXPEDITED_DATA verb described in LU 6.2 architecture.

For more information on this macroinstruction, refer to z/OS Communications Server: SNA Programmer's LU 6.2 Reference.

Receiving expedited data

The APPCCMD CONTROL=RCVEXPD, QUALIFY=ANY|IANY| SPEC|ISPEC macroinstructions receive expedited information.

The APPCCMD CONTROL=RCVEXPD, QUALIFY=ANY|IANY macroinstructions receive expedited information from any active conversation from a partner LU whose expedited data mode is continue-any. The APPCCMD CONTROL=RCVEXPD, QUALIFY=ANY macroinstruction waits for expedited

information to arrive on a conversation in continue-any mode to complete. The APPCCMD CONTROL=RCVEXPD, QUALIFY=IANY returns any expedited information that is immediately available. If no expedited information is available, the macroinstruction completes and an RCPRI,RCSEC of X'0000', X'0008' NO_IMMEDIATELY_AVAILABLE_INFORMATION is returned to the application. Refer to z/OS Communications Server: SNA Programmer's LU 6.2 Reference for more information.

The APPCCMD CONTROL=RCVEXPD, QUALIFY=SPEC|ISPEC macroinstructions receive expedited information from the specified conversation.

The APPCCMD CONTROL=RCVEXPD, QUALIFY=SPEC macroinstruction receives expedited information from the specified conversation. If expedited information is not immediately available, the LU waits to receive expedited information before completing the request. The APPCCMD CONTROL=RCVEXPD, QUALIFY=ISPEC macroinstruction receives expedited information that is immediately available. If expedited information is not available, the macroinstruction completes and an RCPRI,RCSEC of X'0000', X'0008' NO_IMMEDIATELY_AVAILABLE_INFORMATION is returned to the application. Refer to z/OS Communications Server: SNA Programmer's LU 6.2 Reference for more information.

Deallocating the conversation

After the partners have completed the transfer of data, the conversation is deallocated. See Chapter 8, “Deallocating a conversation,” on page 167 for detailed information about deallocating a conversation.

Implementing LU 6.2 option sets

The application program implements optional LU 6.2 functions. However, it cannot implement all of the LU 6.2 options. The following list shows the options that it can implement:

- “Mapped conversations” on page 63
- “Security procedures” on page 63 (conversation–level)
- “Synchronization point services” on page 63
- “Program initialization parameters (PIP) data” on page 65

See “LU 6.2 option sets” on page 35 for details.

Mapped conversations

Mapped conversations describe conversations in which the data to be sent and exchanged is in some form other than logical records. The conversation partners cannot directly issue the VTAM APPCCMD macroinstructions to send and receive data. Instead, they must rely on some other portion of the application program to map the data to be transmitted into logical records. This processing part of the application program then issues the VTAM APPCCMD macroinstructions. Consult the *SNA Transaction Programmer's Reference Manual for LU Type 6.2* for more information on mapped conversations.

Security procedures

VTAM supports architected security options for session-level security and data encryption. VTAM also offers pass-through support for application programs that implement the architected conversation-level security options. For more information on security, see Chapter 13, “VTAM's LU 6.2 security options,” on page 249.

Synchronization point services

Recovery from errors and failures is a prime consideration in the design of transaction programs. LU 6.2 provides optional services to help transaction programs recover from errors. LU 6.2 sync point services enable transaction programs to synchronize their resources when partners in protected conversations fail or encounter errors.

VTAM offers pass-through support for sync point services, but the application program must implement these services. VTAM enables the application program to implement sync point services. VTAM supports

the flow of PS headers, which enables the application program to implement the sync point manager. The sync point manager polices the protected conversations and resources.

For detailed information on sync point services, refer to the *SNA Transaction Programmer's Reference Manual for LU Type 6.2* and the *SNA LU 6.2 Peer Protocols Reference* .

VTAM support for sync point services

Note: Sync point services cannot be used within a full-duplex conversation.

To support the application program's implementation of sync point services, VTAM includes the following functions:

- Defining the system
  - Enables the application program to indicate the synchronization level that is supported by the local LU. The application program does this by specifying the keyword SYNCLVL=CONFIRM or SYNCLVL=SYNCPT on the APPL definition statement.
  - Enables the application program to specify the frequency of occurrence of the ATTN(LOSS) exit that is being driven. The application program does this by specifying the keyword ATNLOSS=LAST or ATNLOSS=ALL on the APPL definition statement.

    Note: The use of the ATNLOSS operand on the APPL definition statement is not restricted to sync point services.
- Allocating conversations and activating sessions
  - Establishes a sync point session if both LUs support the sync point services.
  - Allocates sync point conversations on sync point sessions if the application program requests sync point in its allocation request.
  - Because an FMH-5 for a conversation requires the Logical Unit of Work (LUW) identifier, the application should include the LUW in the FMH-5 presented to VTAM with the APPCCMD CONTROL=ALLOC macroinstruction.
- Sending and receiving data
  - Enables the application program to send PS headers, which are logical records with a length field of X'0001' used to convey sync point information.
- Rejecting sessions
  - Deactivates sessions by using session instance identifiers. Terminates any conversation associated with the session.

    Note: This function is not restricted to sync point services.
- Returning the conversation state
  - Returns the current conversation state at the completion of each conversation-related command. This information is contained in the CONSTATE field in many of the APPCCMD macroinstructions.

    Note: This function is not restricted to sync point processing.
- Suspending sessions
  - Suspends a subsequent conversation's outflow of normal flow requests and suspends the normal deactivation of a session currently in use by a sync point conversation.

    Note: If an application program is using persistent LU-LU session support during sync point processing and fails after APPCCMD CONTROL=SETSESS, QUALIFY=SUSPEND has been issued, but APPCCMD CONTROL=SETSESS, QUALIFY=RESUME has not been issued, VTAM unbinds the sync point sessions. In the same situation, VTAM also unbinds sync point sessions for which APPCCMD CONTROL=SETSESS, QUALIFY=SYNCBEG has been issued but neither APPCCMD CONTROL=SETSESS, QUALIFY=SYNCEND nor APPCCMD CONTROL=SETSESS, QUALIFY=RESUME has been issued at the time of the failure.
- Resuming sessions

– Resumes a session that has been suspended.

Table 7 on page 65 shows several ways in which you might use the APPCCMD CONTROL=SETSESS macroinstructions.

Table 7. Sync point processing

Macroinstruction	Description of Circumstances
APPCCMD CONTROL=SETSESS, QUALIFY=SUSPEND APPCCMD CONTROL=SETSESS, QUALIFY=RESUME	Use these two commands when you have a period of processing time in which the session might have to be unbound before VTAM stops the outbound flow.
APPCCMD CONTROL=SETSESS, QUALIFY=SYNCBEG APPCCMD CONTROL=SETSESS, QUALIFY=SUSPEND APPCCMD CONTROL=SETSESS, QUALIFY=RESUME	Use this combination of commands if you have a period of processing time before the outbound flow is to be stopped, if you are using persistent LU-LU session support, and if you want VTAM to unbind the session during that processing period. (In this situation, APPCCMD CONTROL=SETSESS, QUALIFY=SYNCEND could be used, but it would have to be used with APPCCMD CONTROL=SETSESS, QUALIFY=RESUME.)
APPCCMD CONTROL=SETSESS, QUALIFY=SYNCBEG APPCCMD CONTROL=SETSESS, QUALIFY=SYNCEND	Use these two commands when you do not ever want the outbound flow stopped, but you do want VTAM to unbind a session during a failure when persistence is enabled.

Program initialization parameters (PIP) data

Program Initialization Parameters (PIP) data is a means of presenting data to the partner's transaction program as part of the conversation initialization. PIP data is described in the *SNA Transaction Programmer's Reference Manual for LU Type 6.2* as Send PIP Data (241) and Receive PIP Data (242).

For the application to send PIP data to the partner, it must accompany the FMH-5 when the APPCCMD CONTROL=ALLOC macroinstruction is issued. The application is responsible for:

- Constructing the PIP data properly. See the FM5PIPFM structure contained in the ISTFMH5 structure for the proper values allowed in the PIP GDS variable X'12F5'.
- Indicating in the FMH-5 that the PIP data is attached. Set FM5PIPPR=B'1'.

On the receiving side, the PIP data is not received with the FMH-5 data. After issuing an APPCCMD CONTROL=RCVFMH5, if the FMH-5 indicates that the PIP data is attached (FM5PIPPR=B'1'), then the first APPCCMD CONTROL=RECEIVE will receive the PIP GDS variable X'12F5'.

See “PIP data field” on page 153 for more information about building PIP data on a conversation allocation request.

Chapter 5. Coding the APPCCMD macroinstruction

About this chapter

Application programs request LU 6.2 services from the VTAM program with the APPCCMD macroinstruction. The APPCCMD macroinstruction has many variations, most of which are designated with the CONTROL and QUALIFY keywords. Application programs use the APPCCMD macroinstruction to request both conversation and control operator verb functions.

Application programs can use VTAM LU 6.2 support with this macroinstruction and still issue other VTAM macroinstructions on non-LU 6.2 sessions.

The CONTROL keyword is the highest level qualifier on the APPCCMD macroinstruction and is required on the APPCCMD macroinstruction. The QUALIFY keyword handles additional variations of the APPCCMD macroinstruction and can be omitted if the RPL field for the QUALIFY value is set properly. For example, a series of requests to send data might require that an APPCCMD macroinstruction be issued several times, using the same RPL. The QUALIFY keyword can be omitted after the first SEND macroinstruction if it does not need to be changed.

Use of the APPCCMD macroinstruction

The APPCCMD macroinstruction requests LU 6.2 services from VTAM. Application programs use it to request services for a specific conversation.

Although the macroinstructions can be issued asynchronously, with OPTCD=ASY specified on the macroinstruction, most macroinstruction processing for a conversation is done synchronously. A conversation cannot, for example, have two SEND macroinstructions outstanding at any one time. The asynchronous nature of the APPCCMD macroinstruction applies to the application program as a whole. The application program can have multiple APPCCMD macroinstructions outstanding against several conversations. Several macroinstructions can be outstanding on a conversation. The application program uses the APPCCMD CONTROL=CHECK macroinstruction to get completion information from an asynchronous macroinstruction that VTAM has finished processing.

Refer to z/OS Communications Server: SNA Programming for information on asynchronous processing.

In error situations, an APPCCMD macroinstruction that abnormally deallocates a conversation or terminates its session can be issued while a previous APPCCMD macroinstruction completes.

When a macroinstruction completes, return code feedback information is contained in two RPL fields and two RPL extension fields. The RPL fields are used by all VTAM application programs. The RPL extension fields support only LU 6.2 functions.

There are two RPL fields:

- RTNCD
- FDB2

There are two RPL extension fields:

- RCPRI
- RCSEC

If the RTNCD and FDB2 fields are 0, the macroinstruction has completed successfully and the RCPRI and RCSEC fields have no meaning. If the RTNCD, FDB2=X'00', X'0B', respectively, they indicate conditional completion of the macroinstruction. You should inspect the RCPRI and RCSEC fields to determine what happened.

Other RPL extension fields return:

- Security-level support information.

- An indicator that a request for permission to send data has been received from a partner LU.
- An indicator that error log data has been received.
- An indicator that an FMH-5 has been received.
- The conversation state.
- An indicator that expedited data has been received.

For RECEIVE macroinstructions, the WHATRCV field in the RPL extension also contains feedback information.

Use of the CONTROL keyword

The CONTROL keyword specifies one of the major functions of the APPCCMD macroinstruction. Table 8 on page 68 describes the CONTROL keyword operands and the functions they specify.

Table 8. CONTROL keyword operands and their meanings

Operand	Function
ALLOC	Allocate a conversation.
CHECK	Await completion of an asynchronous macroinstruction and update RPL and RPL extension fields.
DEALLOC	Deallocate a conversation.
DEALLOCQ	Deallocate a conversation.
OPRCNTL	Execute an LU 6.2 control-operator request, such as changing the number of sessions between two LUs.
PREALLOC	Reserve a session for a conversation without sending an FMH-5.
PREPRCV	Switch from SEND state to RECEIVE state (half-duplex conversations only).
RCVFMH5	Receive the function management header required to start a conversation.
RCVEXPD	Receive expedited information from a partner LU on a full-duplex capable session.
RECEIVE	Receive normal information from a partner LU.
REJECT	End a session and conversation due to error and cleanup.
RESETRCV	Reset the conversation's continuation mode
SEND	Send data to a conversation partner.
SENDEXPD	Send expedited data to a conversation partner on a full-duplex capable session.
SENDFMH5	Send the FMH-5 to a partner LU for a preallocated conversation.
SENDRCV	Send data to a conversation partner and receive normal information from the partner (half-duplex only).
SETSESS	Transfer information that pertains to a session between the LU 6.2 function in VTAM and the application program (half-duplex conversations only).
TESTSTAT	Query the current status of conversations.

Use of the QUALIFY keyword

Most of the major LU 6.2 functions designated by the CONTROL keyword can be further controlled through use of the QUALIFY keyword. For example, you can use the QUALIFY keyword with CONTROL=SEND to request a confirmation and order VTAM to flush the send buffer.

Table 9 on page 69 gives the QUALIFY variations for each CONTROL keyword for the APPCCMD macroinstruction. The z/OS Communications Server: SNA Programmer's LU 6.2 Reference describes each variation in detail.

The QUALIFY keyword is optional on the APPCCMD macroinstruction. If it is not specified, it defaults to the value contained in the RPL extension. If it is omitted, the RPL extension field must contain the proper value.

Table 9. QUALIFY keyword operands and their meanings

CONTROL Operand	QUALIFY Operand	Meaning
ALLOC	ALLOCD	Wait for a session to become available when allocating a conversation or when the session limit increases, allowing new sessions to be activated.
	CONVGRP	Allocate resources for a conversation and assign to the conversation a particular existing session with a unique specified conversation group identifier.
	CONWIN	Wait until either an existing contention-winner session becomes available or the contention-winner session limit increases, allowing for a new contention-winner session to be activated.
	IMMED	Allocate a conversation only if a contention-winner session is available immediately.
	WHENFREE	Assign an available session to the conversation. If no session is available but one can be started, queue the conversation. If no session can be started, fail the request.
DEALLOC	ABNDPROG	Deallocate a conversation when the application program detects an error.
	ABNDSERV	Deallocate a conversation when an LU services component encounters an error.
	ABNDTIME	Deallocate a conversation when information is not received within a certain amount of time.
	ABNDUSER	Deallocate a conversation and give VTAM application-specified sense information describing the error.
	CONFIRM	Flush the send buffer, send a confirmation request, and deallocate the conversation if successful (half-duplex conversations only).
	DATACON	Send data, flush the send buffer, send a confirmation request, and deallocate the conversation if successful (half-duplex conversations only).
	DATAFLU	Send data, flush the send buffer, and deallocate the conversation if successful.
	FLUSH	Flush the send buffer and deallocate the conversation if successful.

Table 9. QUALIFY keyword operands and their meanings (continued)

CONTROL Operand	QUALIFY Operand	Meaning
DEALLOCQ	ABNDPROG	Deallocate a conversation when the application program detects an error.
	ABNDSERV	Deallocate a conversation when an LU services component encounters an error.
	ABNDTIME	Deallocate a conversation when information is not received within a certain amount of time.
	ABNDUSER	Deallocate a conversation and give VTAM application-specified sense information describing the error.
OPRCNTL	ACTSESS	Respond positively to a session establishment request.
	CNOS	Change the number of sessions between LUs.
	DACTSESS	Respond negatively to a session establishment request.
	DEFINE	Change information in the LU-mode table.
	DISPLAY	Return information about an LU or mode name associated with an LU.
	RESTORE	Restore mode names and associated sessions that are pending recovery.
PREALLOC	ALLOCD	Wait for a session to become available when preallocating a conversation or when the session limit increases, allowing new sessions to be activated.
	CONVGRP	Preallocate resources for a conversation and assign to the conversation a particular existing session with a unique specified conversation group identifier.
	CONWIN	Wait until either an existing contention-winner session becomes available or the contention-winner session limit increases, allowing for a new contention-winner session to be activated.
	IMMED	Preallocate a conversation only if a contention-winner session is available immediately.
	WHENFREE	Assign an available session to the conversation. If no session is available but one can be started, queue the conversation. If no session can be started, fail the request.
PREPRCV (Half-duplex conversations only)	CONFIRM	Send confirmation request and place application program in RECEIVE state if successful.
	DATACON	Send data, send confirmation request, and place application program in RECEIVE state if successful.
	DATAFLU	Send data and place application program in RECEIVE state after flushing send buffer.
	FLUSH	Flush the send buffer and prepare to receive data by entering RECEIVE state.

Table 9. QUALIFY keyword operands and their meanings (continued)

CONTROL Operand	QUALIFY Operand	Meaning
RCVEXPD	ANY	Receive expedited information from any conversation whose expedited information mode is continue-any; the LU will wait for expedited data to arrive to satisfy the macro request.
	IANY	Receive expedited information immediately available on any conversation whose expedited information mode is continue-any; the LU will not wait for expedited data to arrive to satisfy the macro request.
	ISPEC	Receive expedited information immediately available from the specified conversation; the LU will not wait for expedited data to arrive to satisfy the macro request.
	SPEC	Receive expedited information from the specified conversation; the LU will wait for expedited data to arrive to satisfy the macro request.
RCVFMH5	DATAQUE	Queue the RCVFMH5 request prior to receipt of an FMH-5 and receive any accompanying data.
	NULL	Receive the function management header required to start a conversation. (QUALIFY=NULL is optional for this macroinstruction.)
	QUEUE	Queue the RCVFMH5 request prior to receipt of an FMH-5.
RECEIVE	ANY	Receive normal information for any conversation in continue-any mode.
	SPEC	Receive normal information on a specific conversation.
	IANY	Receive normal information that is immediately available from any conversation in continue-any mode.
	ISPEC	Receive normal information that is immediately available from a specific conversation.
REJECT	CONV	Deallocate the conversation and the underlying session.
	CONVGRP	Deactivate the session associated with a specified conversation group identifier.
	SESSION	Deactivate the session and deallocate any conversation associated with it.
RESETRCV	NULL	Reset the conversation's continuation mode. (QUALIFY=NULL is optional for this macroinstruction.)

Table 9. QUALIFY keyword operands and their meanings (continued)

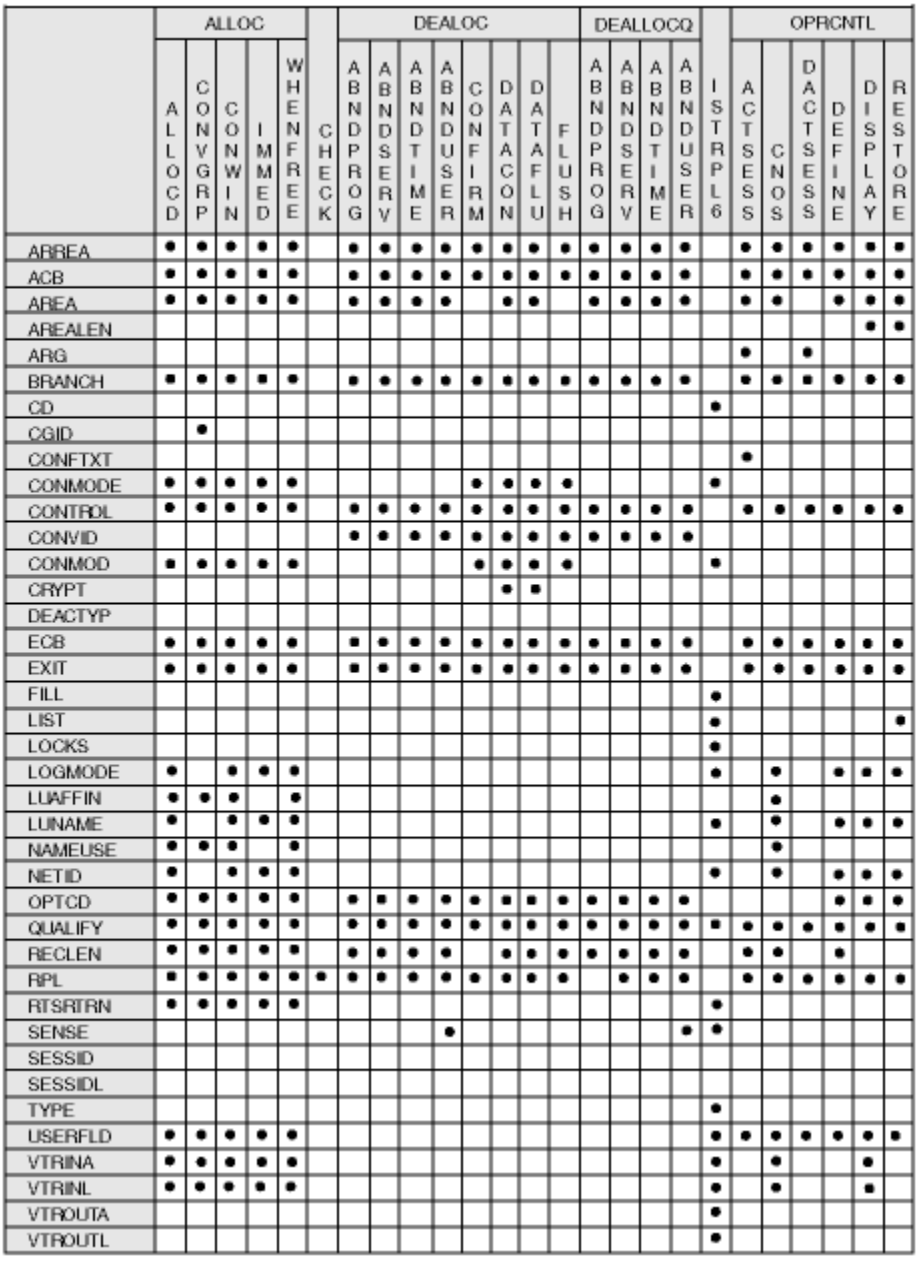
CONTROL Operand	QUALIFY Operand	Meaning
SEND	CONFIRM	Flush the send buffer and send a confirmation request (Half-duplex conversations only).
	CONFRMD	Respond positively to a confirmation request (half-duplex conversations only).
	DATA	Send normal data to a conversation partner.
	DATACON	Send normal data, flush the send buffer, and send a confirmation (half-duplex conversations only).
	DATAFLU	Send normal data and flush the send buffer.
	ERROR	Send error information or respond negatively to a confirmation.
	FLUSH	Flush the send buffer. (Special case for full-duplex conversations: If the partner LU issues a receive FILL=BUFF, the request is completed by a FLUSH-type send, regardless of the amount of data received.)
	RQSEND	Request to enter SEND state (half-duplex conversations only).
SENDEXPD (Full-duplex- capable sessions only)	DATA	Send expedited data to a conversation partner on a full-duplex capable session.
SENDFMH5	NULL	Send the FMH-5 to a partner LU for a preallocated conversation.
SENDRCV	DATAFLU	Send normal data, flush the send buffer, and receive normal information from the partner (half-duplex only).
SETSESS (Half-duplex conversations only)	RESUME	Resume the suspended session.
	SUSPEND	Suspend all outgoing normal flow requests and normal deactivation of a session.
	SYNCBEG	Indicate that a synchronization exchange is beginning.
	SYNCEND	Indicate that a synchronization exchange is ending.
TESTSTAT	ALL	Obtains status on information that is available from any active conversation associated with the specified ACB.
	IALL	Obtains status on information that is immediately available from any active conversation associated with the specified ACB.
	ISPEC	Obtains status on information that is immediately available on a specified conversation.
	SPEC	Obtains status on information that is available on a specified conversation.

Keywords and returned parameters

The following figures show the operands that can be coded for the APPCCMD macroinstruction:

- Figure 12 on page 73
- Figure 13 on page 74

- Figure 14 on page 75



	ALLOC					CHECK	DEALOC								DEALLOCQ				ISTRPL6	OPRCNTL					
	ALLOCD	CONVGRP	CONWIN	IMMED	WHENFREE		ABNDPROG	ABNDSERV	ABNDTIME	ABNDUSER	CONFIRM	DATACON	DATAFLU	FLUSH	ABNDPROG	ABNDSERV	ABNDTIME	ABNDUSER		ACTSESS	CNOS	DACTSESS	DEFINE	DISPLAY	RESTORE
ABREA	•	•	•	•	•		•	•	•	•	•	•	•	•	•	•	•	•		•	•	•	•	•	•
ACB	•	•	•	•	•		•	•	•	•	•	•	•	•	•	•	•	•		•	•	•	•	•	•
AREA	•	•	•	•	•		•	•	•	•		•	•		•	•	•	•		•	•		•	•	•
AREALEN																								•	•
ARG																				•		•			
BRANCH	•	•	•	•	•		•	•	•	•	•	•	•	•	•	•	•	•		•	•	•	•	•	•
CD																			•						
CGID		•																							
CONFTXT																				•					
CONMODE	•	•	•	•	•						•	•	•	•					•						
CONTROL	•	•	•	•	•		•	•	•	•	•	•	•	•	•	•	•	•		•	•	•	•	•	•
CONVID							•	•	•	•	•	•	•	•	•	•	•	•							
CONMOD	•	•	•	•	•						•	•	•	•					•						
CRYPT												•	•												
DEACTYP																									
ECB	•	•	•	•	•		•	•	•	•	•	•	•	•	•	•	•	•		•	•	•	•	•	•
EXIT	•	•	•	•	•		•	•	•	•	•	•	•	•	•	•	•	•		•	•	•	•	•	•
FILL																			•						
LIST																			•						•
LOCKS																			•						
LOGMODE	•		•	•	•														•		•		•	•	•
LUAFFIN	•	•	•		•																•				
LUNAME	•		•	•	•														•		•		•	•	•
NAMEUSE	•	•	•		•																•				
NETID	•		•	•	•														•		•		•	•	•
OPTCD	•	•	•	•	•		•	•	•	•	•	•	•	•	•	•	•	•					•	•	•
QUALIFY	•	•	•	•	•		•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
RECLEN	•	•	•	•	•		•	•	•	•		•	•	•	•	•	•	•		•	•		•		
RPL	•	•	•	•	•	•	•	•	•	•	•	•	•	•		•	•	•		•	•	•	•	•	•
RTSRTRN	•	•	•	•	•														•						
SENSE										•								•	•						
SESSID																									
SESSIDL																									
TYPE																			•						
USERFLD	•	•	•	•	•														•	•	•	•	•	•	•
VTRINA	•	•	•	•	•														•		•			•	
VTRINL	•	•	•	•	•														•		•			•	
VTROUTA																			•						
VTROUTL																			•						

Figure 12. Valid operands for APPCCMD macroinstructions: ALLOC—OPRCNTL

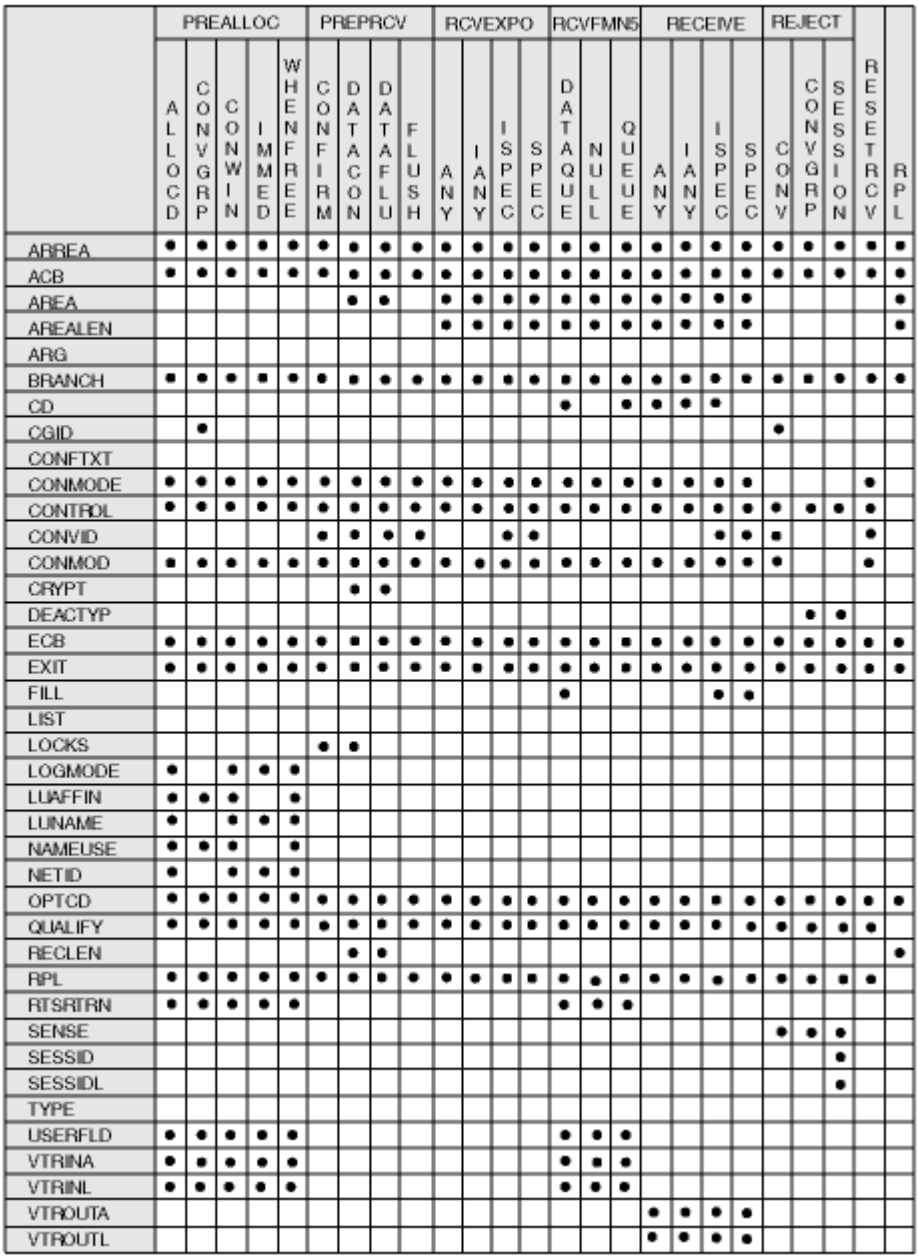


Figure 13. Valid operands for APPCCMD macroinstructions: PREALLOC—RPL

	SEND								SENDEXPO	SENDFMHS	SENDRCV	SETSESS				TESTSTAT			
	CONFIRM	CONFIRMD	DATA	DATACON	DATAFLU	ERROR	FLUSH	RQSEND	DATA	SENDFMHS	DATAFLU	RESUME	SUSPEND	SYNCBEG	SYNCEND	ALL	IALL	ISPEC	SPEC
ARBEA	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
ACB	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
AREA			•	•	•	•			•	•	•					•	•	•	•
AREALEN																•	•	•	•
ARG																			
BRANCH	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
CD											•								
CGID																			
CONFTXT																			
CONMODE	•	•	•	•	•	•	•	•	•		•								
CONTROL	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
CONVID	•	•	•	•	•	•	•	•	•	•	•		•	•				•	•
CONMOD	•	•	•	•	•	•	•	•	•		•								
CRYPT			•	•	•														
DEACTYP																			
ECB	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
EXIT	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
FILL											•								
LIST																			
LOCKS																			
LOGMODE																			
LUNAME																			
NAMEUSE																			
NETID																			
OPTCD	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
QUALIFY	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
RECLEN			•	•	•	•			•	•	•								
RPL	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
RTSRTRN																			
SENSE						•													
SESSID												•	•	•	•				
SESSIDL												•	•	•	•				
TYPE						•													
USERFLD																			
VTRINA																			
VTRINL																			
VTROUTA																			
VTROUTL																			

Figure 14. Valid operands for APPCCMD macroinstructions: SEND—TESTSTAT

The following figures show the parameters that can be returned for each APPCCMD macroinstruction.

- Figure 15 on page 76
- Figure 16 on page 77
- Figure 17 on page 78

	ALLOC					CHECK	DEALOC								DEALLOCQ				OPRCNTL					
	ALLOCD	CONVGRP	CONWIN	IMMED	WHENFREE		ABNDPROG	ABNDSERV	ABNDTIME	ABNDUSER	CONFIRM	DATACON	DATAFLU	FLUSH	ABNDPROG	ABNDSERV	ABNDTIME	ABNDUSER	ACTSESS	CNOS	DACTSESS	DEFINE	DISPLAY	RESTORE
AVEA	•	•	•	•	•															•				
CGID	•		•	•	•																			
CONSTATE	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•						
CONVID	•	•	•	•	•																			
CONVSECP	•	•	•	•	•																			
CRYPTLVL	•	•	•	•	•																			
EXPDLEN							•	•	•	•	•	•	•	•	•	•	•	•						
EXPDRCV							•	•	•	•	•	•	•	•	•	•	•	•						
FDB2	•	•	•	•	•		•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
FMNSLEN	•	•	•	•	•		•	•	•	•	•	•	•	•	•	•	•	•						
FMNSRCV	•	•	•	•	•		•	•	•	•	•	•	•	•	•	•	•	•						
LOGMODE		•																						
LOGRCV											•	•												
LUAFFIN	•	•	•		•															•				
LUNAME		•																						
NETID		•																						
PRSISTVP	•	•	•	•	•															•				
RCPRI	•	•	•	•	•		•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
RCSEC	•	•	•	•	•		•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
RECLEN																							•	•
RPLXSRV												•	•											
RTNCD	•	•	•	•	•		•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
SENSE	•	•	•		•						•	•	•	•					•	•				
SESSID	•	•	•	•	•															•				
SESSIDL	•	•	•	•	•															•				
SIGDATA												•	•											
SIGRCV												•	•											
SLS	•	•	•	•	•																			
STSHBF							•	•	•	•		•	•		•	•	•	•						
STSHDS							•	•	•	•		•	•		•	•	•	•						
USERFLD							•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
WHATRCV																								

Figure 15. Returned parameters for APPCCMD macroinstructions: ALLOC—OPRCNTL

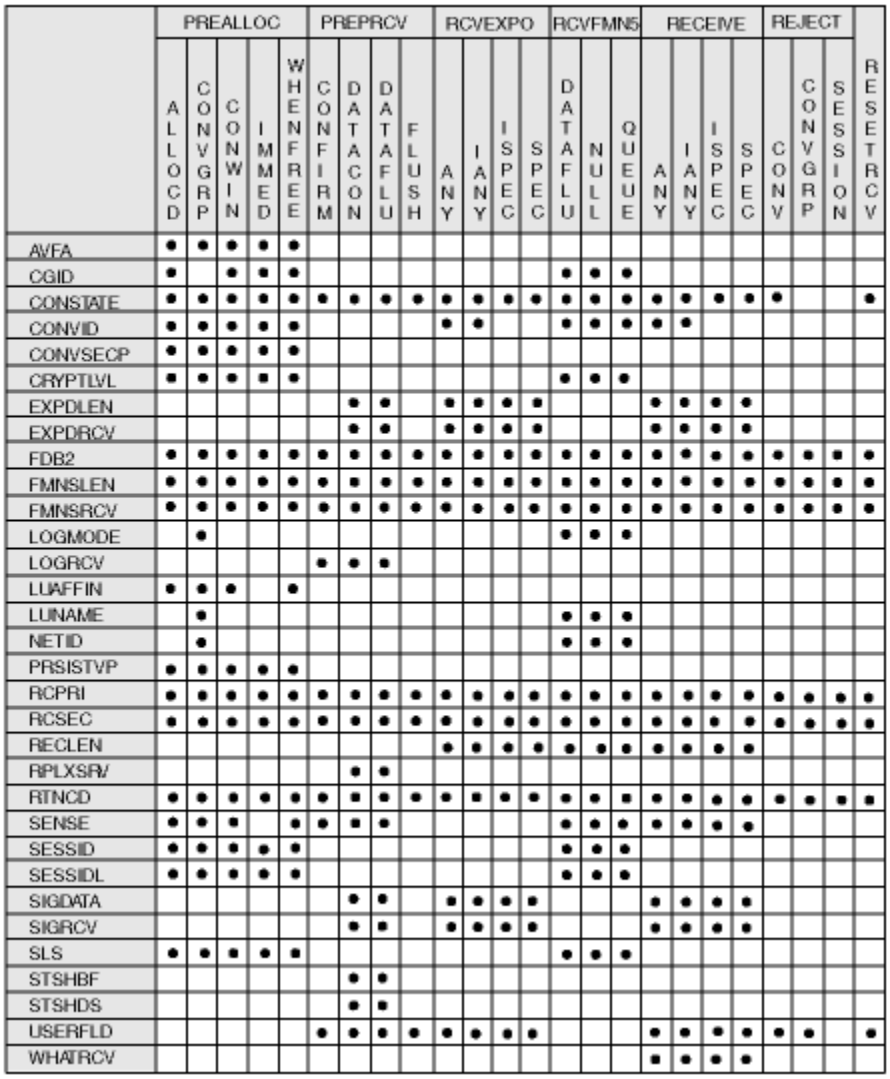


Figure 16. Returned parameters for APPCCMD macroinstructions: PREALLOC—RESETRCV

	SEND								SENDEXPO	SENDFMHS	SENDRCV	SETSESS				TESTSTAT			
	CONFIRM	CONFIRMD	DATA	DATACON	DATAFLU	ERROR	FLUSH	RQSEND	DATA		DATAFLU	RESUME	SUSPEND	SYNCBEG	SYNCEND	ALL	IALL	ISPEC	SPEC
AVFA																			
CGID																			
CONSTATE	•	•	•	•	•	•	•	•	•	•	•							•	•
CONVID																			
CONVSECP																			
CRYPTLVL																			
EXPDLEN	•		•	•	•	•			•		•								
EXPDRCV	•		•	•	•	•			•		•								
FDB2	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
FMNSLEN	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
FMNSRCV	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
LOGMODE																			
LOGRCV	•		•	•	•	•					•								
LUNAME																			
NETID																			
PRSISTVP																			
RCPRI	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
RCSEC	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
RECLEN											•					•	•	•	•
RPLXSRV			•	•	•						•								
RTNCD	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
SENSE	•		•	•	•	•					•								
SESSID																			
SESSIDL																			
SIGDATA	•		•	•	•	•			•		•								
SIGRCV	•		•	•	•	•			•		•								
SLS																			
STSHBF			•	•	•	•					•								
STSHDS			•	•	•	•					•								
USERFLD	•	•	•	•	•	•	•	•	•	•	•							•	•
WHATRCV											•								

Figure 17. Returned parameters for APPCCMD macroinstructions: SEND—TESTSTAT

Parameter-to-DSECT mapping

The following tables illustrate which fields of the RPL DSECT (IFGRPL) and the RPL extension DSECT (ISTRPL6X) correspond to the RPL and RPL extension field names used in this book. They show the RPL fields that are meaningful for an RPL used for an APPCCMD macroinstruction. Table 10 on page 78 shows the DSECT labels for the RPL fields used for LU 6.2 support. For a complete description of the RPL DSECT (IFGRPL), refer to z/OS Communications Server: SNA Programming.

This manual shows the RPL extension fields in which the application program passes information to and receives information from VTAM, along with the corresponding DSECT label in the ISTRPL6X DSECT.

For the actual storage layout of the RPL extension, refer to z/OS Communications Server: SNA Programmer's LU 6.2 Reference. The figure shows the DSECT label for each field, followed by the symbolic name of the field (if any) in parentheses. Blank fields in the figure indicate fields that should not be used or examined by application programs.

Table 40 on page 244 shows the RPL extension fields that are returned for the ATTN exit.

Table 10. RPL fields and DSECT labels in IFGRPL

RPL Field	Corresponding DSECT Label
AAREA	RPLAAREA
AAREALN	RPLAARLN

Table 10. RPL fields and DSECT labels in IFGRPL (continued)

RPL Field	Corresponding DSECT Label
ACB	RPLDACB
AREA	RPLAREA
AREALEN	RPLBUFL
ARG	RPLARG
BRANCH	RPLEXTDS
CRYPT	RPLTCRYP
ECB	RPLECB
EXIT	RPLEXTDS
FDB2	RPLFDB2
OPTCD=KEEPSRB, OPTCD=NKEEPSRB	RPLKPSRB
OPTCD=BUFFLST, OPTCD=NBUFFLST, OPTCD=XBUFLST	RPLOPT6
OPTCD=ASY, OPTCD=SYN	RPLOPT1
RECLEN	RPLRLEN
RTNCD	RPLRTNCD

Table 11. RPL extension fields and DSECT labels in ISTRPL6X

RPL Extension Field	Corresponding DSECT Label
AVFA	RPL6AVFA
CD	RPL6CD
CGID	RPL6CGID
CONFTXT	RPL6CFTX
CONMODE	RPL6CMOD
CONSTATE	RPL6CCST
CONTROL	RPL6REQ
CONVID	RPL6CNVD
CONVSECP	RPL6CLSA
CONXMOD	RPL6CXMD
CRYPTLVL	RPL6CRYP
DEACTYP	RPL6DETP
EXPDLEN	RPL6EXDL
EXPDRCV	RPL6EXDR
FILL	RPL6FILL
FMH5LEN	RPL6MH5L
FMH5RCV	RPL6RMH5

Table 11. RPL extension fields and DSECT labels in ISTRPL6X (continued)

RPL Extension Field	Corresponding DSECT Label
LAST	RPL6LAST
LIST	RPL6LIST
LOCKS	RPL6LOCK
LOGMODE	RPL6MODE
LOGRCV	RPL6RLOG
LUAFFIN	RPL6AFFN
LUNAME	RPL6LU
NAMEUSE	RPL6NAMU
NETID	RPL6NET
QUALIFY	RPL6QUAL
PRSISTVP	RPL6PV
RCPRI	RPL6RCPR
RCSEC	RPL6RCSC
SENSE (specified parameter)	RPL6SNSO
RTSRTRN	RPL6RTSX
SENSE (returned parameter)	RPL6SNSI
SESSID	RPL6SSID
SESSIDL	RPL6SIDL
SIGDATA	RPL6SGNL
SIGRCV	RPL6RSIG
SLS	RPL6SLS
STSHBF	RPL6STBF
STSHDS	RPL6STDS
TYPE	RPL6TYPE
USERFLD	RPL6USR
VTRINA	RPL6VAIA
VTRINL	RPL6VAIL
VTROUTA	RPL6VAOA
VTROUTL	RPL6VAOL
WHATRCV	RPL6WHAT

Keyword specifications

Table 12 on page 81 shows the types of values that you can specify for APPCCMD macroinstruction keywords.

Table 12. Operand specifications for the APPCCMD macroinstruction

Keyword Operand	Notation Category	Format	Example
AAREA	Address	Symbolic name	AAREA=AAREA1
		Register	AAREA=(7)
ACB	Address	Symbolic name	ACB=ACB1
		Register	ACB=(5)
AREA	Address	Symbolic name	AREA=WAREA
		Register	AREA=(7)
AREALEN	Quantity	Decimal number	AREALEN=98
		Register	AREALEN=(8)
ARG	Indirect value	Symbolic name	ARG=ARG1
		Register	ARG=(5)
BRANCH	Fixed value		BRANCH=YES
CD	Fixed value		CD=IMMED
CGID	Indirect value	Symbolic name	CGID=CONWCGID
		Register	CGID=(7)
CONFTXT	Fixed value		CONFTXT=NO
CONMODE	Fixed value		CONMODE=LLCA
CONTROL	Fixed value		CONTROL=SEND
CONVID	Indirect value	Symbolic name	CONVID=ID062
		Register	CONVID=(7)
CONXMOD	Fixed value		CONXMOD=SAME
CRYPT	Fixed value		CRYPT=NO
DEACTYP	Address	Symbolic name	DEACTYP=DEAC001
		Register	DEACTYP=(7)
ECB	Address	Symbolic name	ECB=ECB1
		Register	ECB=(7)
	Fixed value		ECB=INTERNAL
EXIT	Address	Symbolic name	EXIT=EXIT1
		Register	EXIT=(7)
FILL	Fixed value		FILL=BUFF
LIST	Fixed value		LIST=NONE
LOCKS	Fixed value		LOCKS=SHORT
LOGMODE	Name		LOGMODE=MODEX

Table 12. Operand specifications for the APPCCMD macroinstruction (continued)

Keyword Operand	Notation Category	Format	Example
LUAFFIN	Fixed Value		LUAFFIN=NOTAPPL
LUNAME	Name		LUNAME=LU6201
NAMEUSE	Fixed value		NAMEUSE=GNAME
NETID	Name		NETID=NETWORK6
OPTCD	Fixed value		OPTCD=ASY OPTCD=(ASY,BUFFLST)
QUALIFY	Fixed value		QUALIFY=DATACON
RECLEN	Quantity	Decimal number	RECLEN=98
		Register	RECLEN=(8)
RPL	Address	Symbolic name	RPL=RPL1
		Register	RPL=(1)
RTSRTRN	Fixed value		RTSRTRN=BOTH
SENSE	Quantity	Hexadecimal	SENSE=X'08890000'
		Register	SENSE=(8)
SESSID	Address	Symbolic name	SESSID=SESS001
		Register	SESSID=(7)
SESSIDL	Quantity	Decimal number	SESSIDL=8
	Indirect value	Register	SESSIDL=(7)
TYPE	Fixed value		TYPE=PROGRAM
USERFLD	Quantity	Register	USERFLD=(8)
		Character	USERFLD=C'LU01'
		Fixed-point	USERFLD=F'100'
		Hexadecimal	USERFLD=X'0043E010'
		A-type address constant	USERFLD=A(RTN1)
		V-type address constant	USERFLD=V(EXTRTN)
VTRINA	Address	Symbolic name	VTRINA=VTRINAD
		Register	VTRINA=(3)
VTRINL	Quantity	Decimal number	VTRINL=100
		Register	VTRINL=(4)
VTROUTA	Address	Symbolic name	VTROUTA=BUBBA
		Register	VTROUTA=(3)
VTROUTL	Quantity	Decimal number	VTROUTL=100
		Register	VTROUTL=(4)

Chapter 6. Managing sessions

About this chapter

This chapter describes how a VTAM LU 6.2 application program can:

- Negotiate session limits to initialize, change, and reset session limits between two LUs on a per-mode basis.
- Specify negotiation values for CNOS session limit negotiation.
- Activate and deactivate sessions in response to increased or decreased session limits resulting from changes in session limits.
- Retrieve information for a mode and sessions that are to be restored.

The information in this chapter applies to VTAM-to-VTAM communications. Other platforms might be subtly different. If you have VTAM-to-AS/400 communications, you might want to refer to:

- *SAA Common Programming Interface Communications Reference*
- *AS/400 Communications: Intersystem Communication Function Programming Guide*
- *AS/400 Programming: Data Description Specifications Reference*
- *AS/400 Communications: APPN Network User's Guide*

If you have VTAM-to-OS/2 communications, you might want to refer to:

- *Communications Server for Windows NT and Windows 2000, Version 6.1 and Personal Communications for Windows, Version 5.5.*

Negotiating session limits

Session limit negotiation is the process by which two parallel-session-capable LUs determine the session limits for a logon mode group before they establish a session on that logon mode. (Session limits are not negotiated for single-session-capable LUs.) These session limits define the type and number of sessions in use between the two LUs. The logon mode is associated with specific characteristics, such as class of service, which describe its communication capabilities, and is represented by a logon mode name.

To initiate session limit negotiation, either the local LU or the partner LU issues the APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS macroinstruction. The LU that initiates the CNOS request is the source side, and the partner LU is the target side.

VTAM performs the session limit negotiation.

During the process of negotiating session limits, the LUs also can:

- Establish draining responsibility.
- Negotiate session deactivation responsibility.
- Negotiate level of conversation-level security.

How session limits are used

The dialogue between transaction programs is called a conversation. The dialogue between logical units (LUs) is called a session. The conversation uses the session as its transmission medium. The session is a serially reusable resource of the conversation. While the session is in use for one conversation, it cannot be used for any other conversation. More than one session can be active between the same pair of LUs, which is called parallel sessions. The sessions between two LUs are grouped together by mode name. One of the application programs, acting on behalf of the transaction program, initiates the action.

To help with the allocation of storage at the LU, limits are placed on the number of sessions that an LU can have for a given logon mode name.

Before two LUs can start any conversations between them, they must negotiate the type and number of sessions, referred to as session limits. These session limits restrict the number of concurrent conversations an application program manages. Each conversation must be allocated to an active session.

Session limits do not affect the number of conversation requests that the application program can issue. VTAM manages the conversation workload for the application program by controlling when a session is assigned to a conversation request.

A session is associated with a mode, which represents a specific set of session characteristics or capabilities. LUs negotiate session limits on a per-mode basis before any conversations can be allocated on a mode. An LU can use several modes, giving it greater capability, and more than one LU can use the same mode. Each set of characteristics, or mode, is stored in the logon mode table and identified by a logon mode name.

As an example of how session limits function, suppose two LUs negotiate a limit of 10 sessions between them on a logon mode, but they need 100 conversations on that logon mode to service transaction programs. The application programs can issue macroinstructions requesting the 100 conversations. However, because a limit of 10 sessions has been negotiated, only 10 sessions can be active between the two LUs on that logon mode. Therefore, no more than 10 conversations can be active at any given time. When all 10 sessions are assigned to conversations, the remaining conversation requests can be queued until the currently active conversations end and sessions become free. As sessions are freed, they are assigned to queued conversation requests.

Types of sessions

From the perspective of two LUs that are communicating, two types of sessions exist: a contention-winner session and a contention-loser session. Contention occurs when two LUs try to assign the same session simultaneously to different conversation requests received from an application program. An allocation race results. For a specific session, one of the partner LUs is the contention winner and the other partner LU is the contention loser.

Contention-winner session

A contention-winner session is one for which the local LU is designated to win an allocation race. A contention winner does not have to request permission to use an available session.

Contention-loser session

A contention-loser session is one that the local LU surrenders to the partner LU. A contention loser must request permission from the partner LU before using the session.

Bidding for a session

Before VTAM assigns a contention-loser session to a conversation request (if allowed), VTAM bids on the session, on behalf of the contention loser, to ensure that the partner LU, which is the contention winner and controls the session, does not need the session.

Number and representation of sessions

If one LU in an LU-LU pair has only contention-loser sessions available, it might not be able to start a conversation because its partner gets first priority on using all sessions. To prevent this, the LU-LU pair negotiates a guaranteed number of contention-winner sessions for each LU, in addition to the overall session limits for the exchange.

The number of sessions, or session limit values, is negotiated between an LU-LU pair for each logon mode used. These values are described as follows:

Value
Description

SESSLIM
Overall session limit, which is the maximum number of concurrent sessions active between two LUs, using a given logon mode name

MINWINL
Number of contention-winner sessions guaranteed to the local LU, using a given logon mode name

MINWINR
Number of contention-winner sessions guaranteed to the partner LU, using a given logon mode name

In this publication, these values are shown as a triplet of numbers, (6,4,2), which represent the three types of session limits that are negotiated: SESSLIM, MINWINL, MINWINR. Use this type of notation when issuing a MODIFY CNOS command.

The same session limits are written differently for each partner LU. For example, if LU A and LU B negotiate the following session limits for a logon mode:

- The overall session limit is 6.
- LU A is guaranteed 4 contention-winner sessions.
- LU B is guaranteed 2 contention-winner sessions.

The session limits for LU A on that logon mode would be written (6,4,2), and the same session limits for LU B would be written as (6,2,4); Figure 18 on page 85 illustrates this.

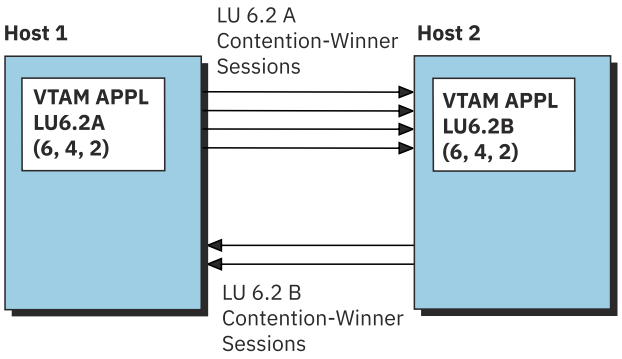


Figure 18. Results of session limit negotiation

The sum of MINWINL and MINWINR cannot be greater than SESSLIM. However, the sum can be less than or equal to SESSLIM. Either partner LU can attempt to activate more than its guaranteed number of contention-winner sessions so long as the overall session limit is maintained.

Application's role in session limit negotiation

Initiating session limit negotiation

The application program initiates session limit negotiation by issuing an APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS macroinstruction.

When the application program initiates session limit negotiation

The application program can initiate a session limit negotiation under the following circumstances:

- Before allocating conversations on a mode.

  Before VTAM can activate any sessions on a logon mode between an LU-LU pair, it negotiates with the partner LU the session limits for that logon mode. This ensures that neither application program is overburdened with conversations.

  When session limits are lowered to 0, VTAM resets session limit information. If this occurs, the application program has to negotiate session limits again before VTAM can activate more sessions.
- When application program performance needs tuning.

The application program can change session limits at any time to make better use of its resources. Raising session limits enables VTAM to activate more sessions as needed. Lowering session limits causes VTAM to deactivate sessions as they are freed by conversations.

The VTAM operator also can issue commands to raise or lower session limits.

- When closing a mode.

  During normal shutdown, the application program closes the mode for all logon modes, including the SNASVCMG mode, before it deactivates the session. This ensures that all conversations are ended in an orderly fashion. Closing the mode includes:

  – Setting session limits for the mode to 0

  – Disabling draining.

  For information on closing a mode, see "Closing a mode" on page 118.

Source-side versus target-side CNOS initiation

An application program's responsibilities during CNOS processing differ depending on whether it is the source or target side of the processing. The LU that initiates the session limit negotiation is the source side of CNOS processing. The LU that receives a session limit negotiation request from its partner is the target side of the CNOS request.

Both LUs in an LU-LU pair are equal in that either one can initiate a CNOS request. Both LUs are responsible for participating in a CNOS negotiation request. However, each logon mode between an LU-LU pair can have only one CNOS request in progress at a time, which means that session limits can be changed for only one mode name at a time with a given partner LU.

If both partners initiate a CNOS request for a logon mode at the same time, they are in a CNOS race. VTAM fails one CNOS request with return codes that indicate a CNOS race has occurred. VTAM continues CNOS negotiation for the other CNOS request. LUs must inspect the return codes.

Source-side processing

To perform a CNOS negotiation, the application program:

1. Can create an entry in the LU-mode table.

   The application program enters the negotiation values in the LU-mode table by:

   - Issuing the APPCCMD CONTROL=OPRCNTL, QUALIFY=DEFINE macroinstruction and specifying the DEFINE control block.
   - Allowing VTAM to use the default negotiation limits on the APPL definition statement. (If an application program issues a CNOS request and has not issued an APPCCMD CONTROL=OPRCNTL, QUALIFY=DEFINE macroinstruction, VTAM uses these default negotiation limits.)
   - Allowing VTAM to use the default values of (2,1,1) when no default negotiation limits have been coded on the APPL definition statement.

   The LU-mode table entry contains the defined negotiation limits that VTAM uses as session limit defaults when the application program does not specify session limits on the CNOS request. The entry in the LU-mode table represents the logon mode being negotiated between the two partner LUs and must exist before the CNOS negotiation can begin.

   The application program can change the defined negotiation limits as needed by using the APPCCMD CONTROL=OPRCNTL, QUALIFY=DEFINE macroinstruction. See "Defined negotiation limits" on page 97 for more information on setting defined negotiation limits.

   If the LU does not create an entry in the LU-mode table, VTAM builds an entry for the LU, using the default defined negotiation limits from the APPL definition statement.

2. Can build a CNOS session limits control block (ISTSLCNS structure).

   Before initiating the CNOS request, the application program builds a CNOS session limits control block to represent the session limits to be set for the logon mode group. In addition to the proposed session

limits, the CNOS session limits control block also contains information that is negotiated between the two LUs on a logon-mode basis. The application program can map storage for the CNOS session limits control block using the ISTSLCNS DSECT. See "Building a CNOS session limits control block" on page 98 for detailed information regarding the structure and meaning of the fields in the CNOS session limits control block.

If the application program does not build a CNOS session limits control block for the CNOS request, VTAM builds one internally by using the defined negotiation limits for the logon mode that are contained in the LU-mode table.

3. Issues the CNOS macroinstruction.

   The application program issues the APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS macroinstruction to start the CNOS negotiation. On the macroinstruction, the application program specifies the partner LU that is the target of the CNOS request and the logon mode that the CNOS request represents. It also specifies the address of the CNOS session limits control block, if the application program builds one.

4. Determines negotiation results.

   For a successful CNOS request, the combination for register 15 and register 0 is always X'0' and X'B', respectively. When the CNOS macroinstruction completes successfully, the application program should check the RCPRI and RCSEC fields in the RPL extension as shown in Table 13 on page 87.

   Table 13. Return codes for a successful CNOS macroinstruction

RCPRI	RCSEC	Indicates
X'0000'	X'0001'	Application program's limits were accepted as specified.
X'0000'	X'0002'	Specified limits were negotiated to different values.

   When an RCPRI code of X'0028' is returned, which indicates the partner LU has closed the mode, no sessions can be activated on that mode. This condition is not necessarily permanent. The CNOS request can be retried later.

RCPRI	RCSEC	Indicates
X'0028'	X'0000'	Partner LU has closed the mode; no sessions can be activated on the mode.

   "Example of CNOS negotiation" on page 91 shows a negotiated CNOS request and the effect that the request has on the session limits.

   If the session limits are negotiated to different values from those originally specified, the source-side LU can determine what the negotiated values are as shown in Table 14 on page 88.

Target-side processing

The application program on the target side of the CNOS negotiation does not have any required responsibilities. VTAM manages the target side of the negotiation for the application program.

VTAM creates entries in the LU-mode table as needed for target-side negotiation and initializes those entries using the defined negotiation limits found in the APPL definition statement. However, the target side can create the LU-mode table entry and set the defined negotiation limits by issuing the APPCCMD CONTROL=OPRCNTL, QUALIFY=DEFINE macroinstruction and specifying the DEFINE control block.

VTAM uses the defined negotiation limits to determine the negotiated session limits to use to process the CNOS request. See "Defined negotiation limits" on page 97 for more information.

If the session limits are negotiated to different values from those originally specified, the target-side LU can determine what the negotiated values are as shown in Table 14 on page 88.

Determining negotiated values

The application program on the source and target side of the CNOS negotiation can determine the existing session limit values that have been negotiated. The methods for doing this are shown in Table 14 on page 88.

Table 14. Methods for determining negotiated values

LU	Method	Description
Source side only	Checks the CNOS session limits control block	If the application program specified a CNOS session limits control block as input to the CNOS macroinstruction, VTAM writes the negotiated values back to the CNOS session limits control block when completing the CNOS negotiation. The application program can read the new values directly from the CNOS session limits control block.
Target side only	Processes the ATTN(CNOS) exit	When a CNOS negotiation completes successfully, VTAM schedules the LU's ATTN exit if the application program has one. The ATTN exit is scheduled with the results of the CNOS negotiation, which the application program can read. For more information on the ATTN exit, see "Using the ATTN exit" on page 239. **Note:** If a CNOS negotiation is processed as a result of the MODIFY CNOS command or an automatic internal CNOS, VTAM schedules the ATTN exit for both the source and target side.
Source or target side	Issues the DISPLAY macroinstruction	The application program can issue an APPCCMD CONTROL=OPRCNTL, QUALIFY=DISPLAY macroinstruction to retrieve from the LU-mode table current information about the partner LU and the logon mode.

VTAM's role in session negotiation

These steps describe VTAM's role in session limit negotiation when the target LU is different from the source LU, which is usually the case. See "VTAM's role in session limit negotiation when PLU=SLU" on page 90 for more information. VTAM, on both the source side and target side of the LU, manages the CNOS negotiation as follows:

1. Builds source-side LU-mode table entries as needed.

   VTAM determines whether an LU-mode table mode entry exists for the logon mode being negotiated between the LU-LU pair. If one does not exist, VTAM builds and initializes the entry using default defined negotiation limits. For more details on how VTAM gets the defined negotiation limits, see "How defined negotiation limits are set" on page 97.

2. Builds a CNOS session limits control block, if needed.

   If the application program did not specify a CNOS session limits control block when it issued the APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS macroinstruction, VTAM builds one for the application program, using the defined negotiation limits contained in the LU-mode table. This CNOS session limits control block represents the proposed session limits for the source side.

3. Starts SNASVCMG conversation.

   The source-side VTAM starts a conversation with the target-side VTAM using the SNASVCMG mode.

   Note: The application program does not have to negotiate CNOS session limits for the SNASVCMG mode before requesting a CNOS negotiation for another user logon mode. VTAM defaults the SNASVCMG mode limits to (2,1,1) automatically when needed.

4. Sends the proposed session limits to the target LU.

VTAM builds a GDS change-number-of-sessions (CNOS) variable that contains the information in the source-side CNOS session limits control block and sends it on the SNASVCMG conversation to the target side.

5. Target side receives the GDS variable and starts its CNOS processing.
6. Builds target-side LU-mode table entry as needed.

   VTAM checks for an LU-mode table mode entry for the logon mode being negotiated between the LU-LU pair. If one does not exist, VTAM builds and initializes the entry using default defined negotiation limits. See "How defined negotiation limits are set" on page 97 for more details on how VTAM gets the defined negotiation limits.
7. Negotiates the session limits.

   The target-side VTAM uses the source side's proposed limits and the target side's defined negotiation limits and determines the negotiated session limits as follows:

   a. The requested SESSLIM value (source) is compared to the DSESLIM value (target), and the smaller of the two values is used for the rest of the negotiation. This value is the overall session limits.
   b. Half of the negotiated SESSLIM value (rounded downward) is compared to the DMINWNR value. The larger of the two values is used in the next step of the negotiation.
   c. The value determined in the previous step is compared to the number of contention-winner sessions the partner application program is requesting. The smaller of the two values is used.
   d. The number of contention-winner sessions given to the partner LU is subtracted from the overall session limit determined in step "7.a" on page 89. This value is compared to the DMINWNL value, and the smaller of the two values is used to determine the number of contention-winner sessions to give to the application program whose VTAM is conducting the negotiation.
   e. The part of the CNOS request that handles responsibility for session deactivation is checked against the DRESPL value. Deactivation responsibility is performed when session limits are being lowered or set to 0.
8. Sends the negotiated limits to the source side.

   The target side updates the GDS variable with the negotiated session limits and sends them back to the source side. The SNASVCMG mode conversation is deallocated.
9. Updates LU-mode table.

   When the session limit negotiation is successful, the VTAM on each side of the negotiation updates the session limits in the LU-mode table to reflect the newly negotiated session limits.
10. Schedules ATTN exit.

    When the session limit negotiation is successful, the VTAM on only the target side of the negotiation gives control to the application program's ATTN exit routine (if it has one) to notify the application program of the new session limits.
11. Initiates session activation or deactivation as needed.

    When the session limit negotiation is successful, the VTAM on each side of the negotiation initiates session activation or deactivation as needed. If AUTOSES is set on, VTAM tries to match it within the constraints of the session limits and to satisfy all session activation requests that are waiting. For more information on session activation and deactivation, see "VTAM's role in session activation and deactivation" on page 121.
12. Completes the CNOS macroinstruction.

    VTAM sets return codes and posts them to the source application program. If the application program specified a CNOS session limits control block on the APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS macroinstruction, the negotiated session limits are copied back to the CNOS session limits control block when the macroinstruction completes.

VTAM's role in session limit negotiation when PLU=SLU

An LU 6.2 application can specify itself on the LUNAME parameter of a CNOS request. This allows transaction programs on the application to allocate LU=OWN conversations. If VTAM determines that the target LU is the same as the source LU for the CNOS request, VTAM accepts the request and manages the CNOS negotiation as follows:

1. Builds source-side LU-mode table entries as needed.

   VTAM determines whether an LU-mode table entry exists for the logon mode being negotiated between the LU-LU pair. If one does not exist, VTAM builds and initializes the entry using default defined negotiation limits.

2. Builds a CNOS session limits control block, if needed.

   If the application program did not specify a CNOS session limits control block when it issued the APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS macroinstruction, VTAM builds one for the application program, using the defined negotiation limits in the LU-mode table. This CNOS session limits control block represents the proposed session limits for the source side.

3. Starts SNASVCMG conversation.

   The source-side VTAM starts a conversation with the target-side VTAM using the SNASVCMG mode.

   Note: The application program does not have to negotiate CNOS session limits for the SNASVCMG mode before requesting a CNOS negotiation for another user logon mode. VTAM defaults the SNASVCMG mode limits to (2,1,1) automatically when needed.

4. Bypasses target side CNOS processing.

   Because the source side LU and the target side LU are the same, the proposed session limits are not sent to the target side. A separate target side LU-mode entry is not built. One LU-mode entry suffices for both sides.

5. Adjusts proposed session limits.

   The proposed session limits are adjusted as follows:

   - The requested SESSLIM value is doubled. This is because each time a session is used, internal session representations are used two at a time.
   - The number of contention winner sessions is set to the requested SESSLIM value.
   - The number of contention loser sessions is set to the requested SESSLIM value.

   For example, if SESSLIM=10, then VTAM sets the session limits to (20,10,10).

6. Updates the LU-mode table.

   If the adjusted session limits are deemed acceptable, then VTAM updates the session limits in the LU-mode table to reflect the adjusted values.

7. Does **not** schedule the ATTN exit.

   Unlike normal CNOS processing, when an LU requests a session with itself, the ATTN(CNOS) exit is not scheduled on the target side.

8. Initiates session activation or deactivation as needed.

   When the session limit negotiation is successful, VTAM initiates session activation or deactivation as needed. If AUTOSES is set on, VTAM tries to match it within the constraints of the session limits and to satisfy all session activation requests that are waiting.

9. Completes the CNOS macroinstruction.

   VTAM sets return codes and posts them to the source application program. If the application program specified a CNOS session limits control block on the APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS macroinstruction, the negotiated session limits are copied back to the CNOS session limits control block when the macroinstruction completes.

Example of a CNOS request

Figure 19 on page 91 is an example that shows an application program with an ACB of APPLA requesting new session limits for sessions using the mode EXAMPLE with a partner application program called APPLB.

The application program is requesting overall session limits of 11: 8 local contention-winner sessions, and 3 contention-winner sessions for APPLB. The CNOS requests that APPLB be responsible for deactivating sessions. (In studying this example, refer to the descriptions of the CNOS session limits control block and the ISTSLCNS DSECT. "Building a CNOS session limits control block" on page 98 describes the control block. Table 17 on page 99 shows the layout of the CNOS session limits control block. The z/OS Communications Server: SNA Programmer's LU 6.2 Reference shows the DSECT.)

```
         LA    10,CNOSCB         * LOAD ADDRESS OF CONTROL BLOCK
         USING ISTSLCNS,10       * ESTABLISH ADDRESSABILITY
         MVC   SLCSESSL,=X'000B' * SET OVERALL SESSION LIMITS FIELD
         MVC   SLCMCWL,=X'0008'  * SET LOCAL CONTENTION WINNERS FIELD
         MVC   SLCMCWP,=X'0003'  * SET PARTNER CONTENTION WINNERS FIELD
         MVI   SLCPARMS,SLCPRSPL * PARTNER LU TO DEACTIVATE SESSIONS
*
         APPCCMD CONTROL=OPRCNTL,                                      X
               QUALIFY=CNOS,                                           X
               RPL=RPLA,                                               X
               AAREA=RPLAX,                                            X
               ACB=APPLA,                                              X
               LUNAME=APPLB,                                           X
               LOGMODE=EXAMPLE,                                        X
               AREA=CNOSCB,                                            X
               RECLEN=16
                   .
                   .
                   .
CNOSCB   DS    XL16             * STORAGE FOR CONTROL BLOCK
RPLA     RPL AM=VTAM            * RPL STORAGE
RPLAX    ISTRPL6                * RPL EXTENSION STORAGE
APPLA    ACB AM=VTAM,MACRF=LOGON,APPLID=APPLNAME * ACB STORAGE
```

Figure 19. Example of a CNOS request

In the example, the AREA parameter could also have been coded as AREA=(10) because the address of the control block was in the register. See "Example of CNOS negotiation" on page 91 for an example showing the use of this CNOS request in a situation involving a partner LU.

Example of CNOS negotiation

Using the sample macroinstructions shown in Figure 19 on page 91 and in Figure 21 on page 116, see how VTAM would process these macroinstructions. Figure 20 on page 92 shows a simplified view of the flow of the CNOS request. The following description is keyed to the numbers in the figure.

Six sessions are active with Application B. Each side has three contention-winner sessions. None are presently allocated for conversations. Application A's AUTOSES value is 8.

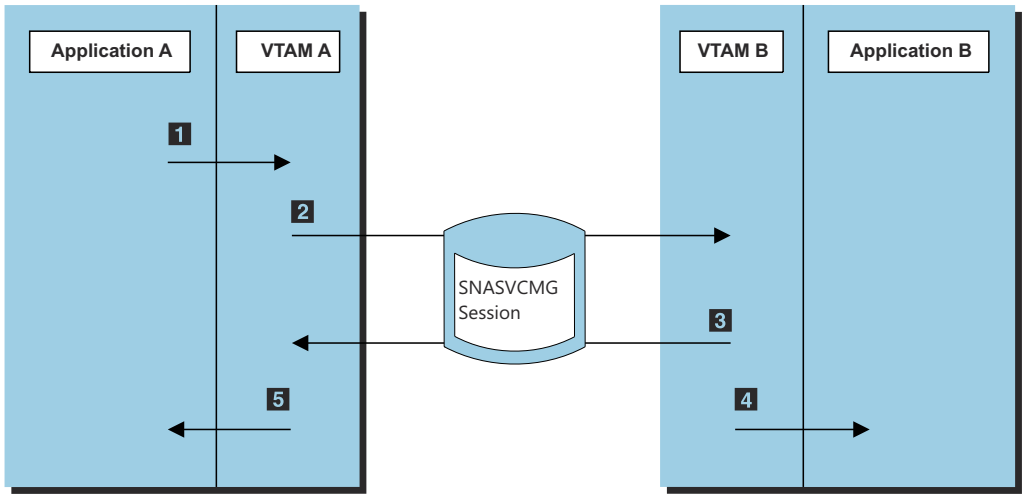


Figure 20. Sample CNOS negotiation

1 Application A issues the following APPCCMD:

```
APPCCMD CONTROL=OPRCNTL,QUALIFY=CNOS,RPL=RPLA,ACB=APPLA,      X
               AAREA=RPLAX,LUNAME=APPLB,LOGMODE=EXAMPLE,AREA=CNOSCB,    X
               RECLEN=16
```

(For more information on this sample CNOS request, see the example in Figure 19 on page 91.) The first 7 bytes of the CNOS session limits control block are set to X'000B0008000320', which indicates the requested session limit values. The values are SESSLIM=11, MINWINL=8, and MINWINR=3, and indicate that the partner application program is responsible for session deactivation.

2 VTAM A uses the SNASVCMG session to pass Application A's CNOS request to VTAM B.

3 VTAM B negotiates the CNOS request and returns the results to VTAM A over the SNASVCMG session. The negotiation values in effect in Application B's LU-mode table for the EXAMPLE mode are DSESLIM=12, DMINWNL=8, and DMINWNR=4, and deactivation responsibility will not be accepted.

The negotiation values were established earlier by Application B by issuing the following macroinstruction:

```
APPCCMD CONTROL=OPRCNTL,QUALIFY=DEFINE,RPL=RPLB,ACB=APPLAB,   X
               AAREA=RPLBX,LUNAME=APPLA,LOGMODE=EXAMPLE,AREA=CBAREA,    X
               RECLEN=68
```

(See the example in Figure 21 on page 116 for more information on the macroinstruction.)

Application B's AUTOSES value is 5.

VTAM B negotiates the CNOS request as follows:

1. The requested SESSLIM value (source) is compared to the DSESLIM value (target), and the smaller value used for the rest of the negotiation. In this case, the SESSLIM value of 11 is used because the DSESLIM value is 12.
2. Half of the negotiated SESSLIM value (rounded downward) is compared to the DMINWNR value. The larger of the two values is used in step three of the negotiation. In this case, half of SESSLIM is 5. This value is used because DMINWNR is 4.
3. The value determined in step two is compared to the number of contention-winner sessions that Application A is requesting. The smaller of the two values is used. In this case, 5 is being compared to 8; therefore, five contention-winner sessions will be given to Application A.
4. The number of contention-winner sessions given to Application B is subtracted from the overall session limit determined in step one. This value is compared to the DMINWNL value, and the smaller of the two values is used to determine the number of contention-winner sessions to give to the

application program whose VTAM is conducting the negotiation. In this case, six sessions are left over and this number is compared to the DMINWNL value of 8. Six is smaller; therefore, Application B will be given six contention-winner sessions.

5. The part of the CNOS request that handles responsibility for session deactivation is checked against the DRESPL value. In this case, VTAM B will not accept a CNOS that requires Application B to be responsible for deactivation, so the DRESPL value of the reply is set appropriately. (The DRESPL value is set in the CNOS session limits control block values returned to the application program after the CNOS negotiation is complete.)

4 VTAM B schedules Application B's ATTN exit to inform it of the new session limits. The AREA field of the read-only RPL in the exit's parameter list points to a CNOS control block. The first 7 bytes of this CNOS control block are set to hex 000B0006000520. This indicates values of SESSLIM=11, MINWINL=6, and MINWINR=5, and the partner is responsible for session deactivation.

VTAM B also activates two additional contention-winner sessions for Application B, bringing the number of contention-winner sessions to the AUTOSES limit of 5. Application B still has an inactive contention-winner session available.

5 VTAM A completes the CNOS request macroinstruction of Application A and sets return codes of X'0000' in RCPRI and X' 0002' in RCSEC. This combination of return codes indicates successful processing. (For a description of the return codes, refer to z/OS Communications Server: SNA Programmer's LU 6.2 Reference.)

The CNOS control block supplied by Application A has been updated and is now X'000B0005000600'. This indicates values of SESSLIM=11, MINWINL=5, and MINWINR=6, and the local LU (Application A) is responsible for deactivation. The AUTOSES value for Application A is 8, but VTAM A activates only two more contention-winner sessions because the contention-winner limit is 5, and three sessions are already active.

Logon mode table versus LU-mode table

Two tables are defined to VTAM for modes:

Logon mode table
: This table defines the characteristics, or capabilities, of sessions and is used by all types of LUs. Each logon mode is identified by a logon mode name.

LU-mode table
: This table contains session limits. It contains the names of LUs that are possible partners and the logon mode names and their characteristics that have been defined for each LU through the APPCCMD CONTROL=OPRCNTL, QUALIFY=DEFINE macroinstruction, or the APPL definition statement, or the VTAM default, which is dynamically built. The logon mode name in the LU-mode table should match the logon name in the logon mode table. This table is specific to LU 6.2.

Logon mode table

VTAM uses the logon mode table to construct session parameters. You define entries for the logon mode table using the MODEENT macroinstruction. This macroinstruction associates a logon mode name, defined in the logon mode table, with a set of parameters that represents session protocols. The logical unit presentation services usage field specifies the optional protocols and data streams for the LU. (This field is coded on the MODEENT macroinstruction using the PSERVIC operand.)

For more information on the MODEENT macroinstruction and on how to build a logon mode table, refer to z/OS Communications Server: SNA Resource Definition Reference. For information about the specific bits of the PSERVIC operand and setting the session parameter fields, refer to z/OS Communications Server: SNA Programming.

Define the logon mode table before using a logon mode name when allocating a conversation.

When two LUs allocate a session, they use a logon mode name, representing a logon mode table entry, to specify the session characteristics needed.

If a logon mode name is present on the LOGMODE operand, the content of the MODEENT macroinstruction is moved to the suggested BIND image that flows within the CINIT RU. (Values can be overridden as necessary.)

If you do not define the logon mode name with the MODEENT macroinstruction before attempting to allocate a conversation using the logon mode name, VTAM has no associated parameters for that logon mode name to use when establishing a session for the application program if the application program is to act as the secondary logical unit.

If a specified logon mode name is not defined to the VTAM that owns the SLU, that VTAM may use a default logon mode name, ISTCOSDF, depending on the setting of the ISTCOSDF start option. For more information on the ISTCOSDF start option, refer to z/OS Communications Server: SNA Resource Definition Reference.

VTAM's support for LU 6.2 does not allow a blank name to be specified as a value on the LOGMODE keyword in APPCCMD macroinstructions. See “Blank mode names” on page 44 for more information.

LU-mode table

VTAM dynamically builds an LU-mode table for an application program when an OPEN ACB macroinstruction is processed. Once the table is built, it contains the names of the LUs that are possible partners and the logon mode names and their session limit characteristics as defined for each LU. The table associates the logon mode names with the partner LUs.

Each LU-LU pair can have unique modes that have unique session limits. As a result, one LU-LU pair might have different capabilities than another LU-LU pair. For more information on logon mode names, refer to z/OS Communications Server: SNA Programming.

Adding to the LU-mode table

The application program and VTAM can define LU entries and mode entries for the LU-mode table as follows:

- The application program can define entries to the LU-mode table dynamically by using either of the following macroinstructions:
  - APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS
  - APPCCMD CONTROL=OPRCNTL, QUALIFY=DEFINE

  When either of these macroinstructions processes successfully, an entry is added to the LU-mode table, if one is not present.
- VTAM defines entries to the LU-mode table when it receives a session activation request (BIND or CINIT) or a CNOS request for an application program, and the partner LU or the mode name is not already in the table.

An entry also is added if the operator issues a MODIFY CNOS or MODIFY DEFINE command and the LU or the mode name is not in the table.

LU entries in the LU-mode table

VTAM allows, at most, two LU entries for a given partner LU. For example, if an application attempts to initiate a session with a partner LU using the partner's generic resource name, the partner can return its real network name on the BIND response. VTAM manages the LU entries so that the name supplied by the application can be associated with the name returned by the partner. Only one set of mode entries is maintained for the partner, eliminating conflicting information.

There can be five types of LU entries in the LU-mode table.

SUPPLIED_NAME Entry
This is an LU entry that is created using the LU name specified on an APPCCMD macroinstruction or the corresponding operator command. This type of entry holds any mode entries for the partner LU. This type of entry may or may not have an associated entry.

RCVD_NAME Entry
This is an LU entry that is created due to a session initiation request from the partner LU. This type of entry holds any mode entries for the partner LU. It may subsequently be changed to a SUPPLIED_NAME entry or a VARIANT_NAME entry. This type of entry does not have an associated entry.

VARIANT_NAME Entry
This is an LU entry that is created when the LUNAME in the BIND response is different than that specified on the APPCCMD or operator command which caused the session request. This type of entry has no mode entries and contains only a limited amount of relevant information. This type of entry is always associated with a SUPPLIED_NAME entry, which contains the complete set of LU information and all related mode entries for the partner LU.

UNUSABLE_NAME Entry
This is an LU entry that has been marked unusable due to inappropriate name translations or conflicting LU definitions in the network. Any attempt to use this LU name is rejected as an error. This type of entry has no mode entries. An UNUSABLE entry contains a limited amount of information for problem determination purposes, and remains in the LU-Mode table until the table is deleted when the application closes its ACB or an operator deletes the entry on a MODIFY DEFINE command.

DISASSOC_NAME Entry
This is an LU entry that was previously a VARIANT_NAME entry but is no longer associated with any other entry. It remains in the table but is changed to another entry type if its name is referenced in session startup. This type of entry has no mode entries.

For more information about how VTAM uses the types of LU entries in the LU-mode table, see "LU 6.2 names used for session activation" on page 140.

Retrieving information from the LU-mode table

The application program can retrieve information from the LU-mode table by using the APPCCMD CONTROL=OPRCNTL, QUALIFY=DISPLAY macroinstruction. This macroinstruction displays the fields of the DEFINE/DISPLAY control block, which contain the information from the LU-mode table.

Specifying values for session limit negotiation

The application program is responsible for specifying values used during session limit negotiation. VTAM finds values for CNOS negotiation as follows:

- If the application program sets negotiation values on the CNOS session limits control block using the APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS macroinstruction, VTAM uses those values.
- If the application program does not specify a CNOS session limits control block, VTAM builds one. VTAM uses the defined negotiation limits from the LU-mode table to build the control block. These values would have to have been set previously by specifying a DEFINE control block using the APPCCMD CONTROL=OPRCNTL, QUALIFY=DEFINE macroinstruction. (You also can issue the MODIFY DEFINE command to set defined negotiation limits.)
- If defined negotiation limits were not specified with an APPCCMD CONTROL=OPRCNTL, QUALIFY=DEFINE macroinstruction, VTAM uses the values of the parameters coded on the APPL definition statement.
- If parameters were not coded on the APPL definition statement, VTAM uses the defaults for the parameters: (2,1,1). The defaults are explained in the z/OS Communications Server: SNA Resource Definition Reference.

The values that can be negotiated during session limit negotiation are shown in Table 15 on page 96.

Table 15. Values and parameters used in session limit negotiation

Value	Source	Set by	Stored
Defined negotiation limits	DEFINE control block	Application program using the APPCCMD CONTROL=OPRCNTL, QUALIFY=DEFINE macroinstruction	LU-mode table
	APPL definition statement	Systems programmer	LU-mode table
	VTAM default values (2,1,1)	VTAM	LU-mode table
CNOS negotiation limits	CNOS control block	Application program using the APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS macroinstruction	LU-mode table
		VTAM using defined negotiation limits from the DEFINE control block or the APPL definition statement	LU-mode table
Local draining responsibility	CNOS control block	Application program using the APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS macroinstruction	CNOS control block
Remote draining responsibility	CNOS control block	Application program using the APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS macroinstruction (Remote side might negotiate to not allow draining)	CNOS control block
Local draining acceptance	DEFINE control block	Application program using the APPCCMD CONTROL=OPRCNTL, QUALIFY=DEFINE macroinstruction	LU-mode table
	APPL definition statement	System programmer	LU-mode table
	VTAM default value (DDRAINL=NALLOW)	VTAM	LU-mode table
Session deactivation responsibility	CNOS control block	Application program using the APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS macroinstruction	CNOS control block
Local session deactivation acceptance	DEFINE control block	Application program using the APPCCMD CONTROL=OPRCNTL, QUALIFY=DEFINE macroinstruction	LU-mode table
	APPL definition statement	System programmer	LU-mode table
	VTAM default value (DRESPL=NALLOW)	VTAM	LU-mode table

Table 15. Values and parameters used in session limit negotiation (continued)

Value	Source	Set by	Stored
Local security acceptance level	DEFINE control block	Application program using the APPCCMD CONTROL=OPRCNTL, QUALIFY=DEFINE macroinstruction	LU-mode table
	CNOS control block	Application program using the APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS macroinstruction	LU-mode table
	APPL definition statement	System programmer	LU-mode table
	VTAM default value (SECACPT=NONE)	VTAM	LU-mode table

Defined negotiation limits

The application program issues the APPCCMD CONTROL=OPRCNTL, QUALIFY=DEFINE macroinstruction to set the defined negotiation limits specified in the DEFINE control block. Otherwise, VTAM sets the default defined negotiation limits using values on the APPL definition statement. (The defined negotiation limits on the APPL definition statement are specified before a CNOS negotiation takes place.)

Defined negotiation limits can be predefined for use during CNOS negotiation when no other values are available. They are set for each logon mode and stored in the LU-mode table with the session limits. They are different from session limits.

When VTAM is the target of a CNOS request, VTAM uses the defined negotiation limits as the target application program's bid in the CNOS negotiation. These limits are not necessarily the final negotiated limits.

For example, an application program might have (10,5,5) as the defined negotiation limits for a logon mode, but when a CNOS negotiation completes with a partner LU, the resulting session limits might be negotiated to (4,2,2). VTAM enforces the (4,2,2) limits on that logon mode group.

The defined negotiation limits are also used when VTAM is the source side of the CNOS request and the application program does not specify a CNOS session limits control block.

How defined negotiation limits are set

When session limits are added to the LU-mode table, the defined negotiation limits are set as follows:

- When the application program sets the values

  If the application program issued the APPCCMD CONTROL=OPRCNTL, QUALIFY=DEFINE macroinstruction to add negotiated limits to the LU-mode table, VTAM reads the defined negotiation limits out of the DEFINE/DISPLAY control block (ISTSLD structure) that was specified on the macroinstruction. See “Defining negotiation limits and displaying session limits” on page 107 for more information on how to issue a DEFINE macroinstruction.

- When VTAM sets the values

  If VTAM is adding an LU-mode table entry (not modifying an entry) because a CNOS request is being processed on the logon mode (on either the target or source side), VTAM adds the LU-mode table entry and sets the defined negotiation values using the corresponding parameters on the APPL definition statement:

  Parameter
  Description

  DSESLIM
  Sets the defined overall session limit in the LU-mode table mode entry. If this parameter is not coded on the APPL definition statement, VTAM uses the default value of 2.

DMINWNL
Sets the defined number of guaranteed contention-winner sessions for the local LU. If this parameter is not coded on the APPL definition statement, VTAM uses the default value of 1.

DMINWNR
Sets the defined number of guaranteed contention-winner sessions for the remote LU. If this parameter is not coded on the APPL definition statement, VTAM uses the default value of 1.

For more information on the default values and on coding the APPL definition statement, refer to the z/OS Communications Server: SNA Resource Definition Reference.

Parameters on the APPL definition statement

In addition to the defined negotiation limits, you can define on the APPL definition statement other parameters that VTAM uses during CNOS negotiation. The application program sets these values in the CNOS session limits control block. These values are exchanged between the two LUs during CNOS negotiation at the same time the session limits are negotiated. The CNOS control block is specified on the APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS macroinstruction.

The parameters of the APPL definition statement and their corresponding fields in the CNOS session limits control block are shown in Table 16 on page 98. Refer to the z/OS Communications Server: SNA Resource Definition Reference for detailed information on coding these parameters. For the meaning and usage of these parameters, see "Building a CNOS session limits control block" on page 98.

Table 16. APPL statement parameters and CNOS session control block fields

APPL Parameter	Symbolic Name	DSECT Name
DSESLIM	SESSLIM	SLCSESSL
DMINWNL	MINWINL	SLCMCWL
DMINWNR	MINWINR	SLCMCWP
DRESPL	RESP	SLCPRSPL
DDRAINL	DRAINL	SLCDRAL

Building a CNOS session limits control block

The CNOS session limits control block (ISTSLCNS) is used to:

- Provide the CNOS session limits that VTAM is to use to process a CNOS request from an application program.
- Return the negotiated values—both to an application program that issues a CNOS request and to an application program that is the target of a CNOS request—provided the target application program has an ATTN exit.

Building a CNOS session limits control block is an optional action. If you request a CNOS session limits control block by issuing the APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS macroinstruction, set the AREA field of the RPL to point to the control block and set the AREALEN field of the RPL to decimal 16, the length of the control block.

If you provide the CNOS session limits control block, when the APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS macroinstruction completes, VTAM sets in the control block the updated values that were negotiated. For details on setting the address of the control block and its length in the RPL, refer to the z/OS Communications Server: SNA Programmer's LU 6.2 Reference.

Note: When a CNOS session limits control block is specified on the APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS macroinstruction, initialize all fields in the control block, even when only one field is to be changed. If the application program does not specify a CNOS session limits control block, VTAM uses previously defined negotiation limits found in the LU-mode table. For more information on defined values,

see “How defined negotiation limits are set” on page 97. Table 17 on page 99 shows the values for each field.

Layout of the CNOS session limits control block

Table 17 on page 99 shows the layout of the control block for CNOS session limits.

Table 17. Layout of the CNOS session limits control block

Byte (Hex)	Bit	Symbolic Name	DSECT Name	Usage
0–1	—	SESSLIM[1]	SLCSESSL	Session limit of mode name group.
2–3	—	MINWINL[1]	SLCMCWL	Number of parallel sessions of which the source side is guaranteed to be the contention winner.
4–5	—	MINWINR[1]	SLCMCWP	Number of parallel sessions of which the partner LU is guaranteed to be the contention winner.
6	0	DRAINL[1]	SLCDRAL	Whether the source side can drain its allocation requests.
6	1	DRAINR[1]	SLCDRAP	Whether the target side can drain its allocation requests.
6	2	RESP[1]	SLCPRSPL	The LU responsible for deactivating the sessions as a result of resetting the session limit or changing the contention-winner values for parallel-session connections.
6	3	NBRMODE[3]	SLCALL	Whether the resetting of the session limits to 0 is to apply for only the specified mode name or for all mode names that apply to the partner LU.
6	4	SNGSESLU[3]	SLCSSLU	The partner LU is a single-session LU and does not support parallel sessions.
6	5	CONVSECL[1]	SLCLCONV	Whether the local LU accepts conversation requests that include security information from a partner LU.
6	6	ALRDYVL[1]	SLCLAVFA	Whether the already-verified indicator can be accepted by the local LU on conversation requests.
6	7	PRSISTVL[1]	SLCLPV	Whether persistent-verification indicators can be accepted by the local LU on conversation requests.

Table 17. Layout of the CNOS session limits control block (continued)

Byte (Hex)	Bit	Symbolic Name	DSECT Name	Usage
7	—	(reserved)		
8–9	—	DSESLIM[2]	SLCDSESL	The value for the maximum session limit for the mode name group. This value is used when the application program is the target side of a CNOS request.
A–B	—	DMINWNL[2]	SLCDMCWL	The number of parallel sessions of which the application program is the contention winner. This value is used when the application program is the target side of a CNOS request.
C–D	—	DMINWNR[2]	SLCDMCWP	The number of parallel sessions of which the partner LU is the contention winner. This value is used when the application program is the target side of a CNOS request.
E	0	DDRAINL[2]	SLCDDRAL	Whether the local LU accepts permission to drain its allocation requests. This value is used when the application program is the target side of a CNOS request.
E	1	DRESPL[2]	SLCDRSPL	Whether the local LU is willing to be responsible for session deactivations. This value is used when the application program is the target side of a CNOS request.
E	2	DEFINE[2]	SLCDEFND	Whether ATTN CNOS is scheduled because of a CNOS request processed on the target side or because the MODIFY DEFINE command is issued.
F	—	(reserved)		

The following notes apply to the entries in the Symbolic Name field.

[1] These fields specify the input and return output. The output value might not be the same as the input value.

[2] These values are returned from CNOS and are set either with the values specified on the APPL definition statement or with the APPCCMD CONTROL=OPRCNTL, QUALIFY=DEFINE macroinstruction.

[3] These fields only specify input.

The following information shows a description of the fields in the control block. The DSECT name of the field is given in parentheses after the symbolic name used in the text.

- **ALRDYVL (SLCLAVFA)** indicates whether the already-verified indicator can be accepted by the local LU on conversation requests. If YES is specified, it also must be specified on the CONVSECL parameter. See the "CONVSECL (SLCLCONV)" description in this section for more information.

  - **B'0' (NO)** specifies that the local LU does not accept already-verified requests.
  - **B'1' (YES)** specifies that the local LU does accept already-verified requests.
- **CONVSECL (SLCLCONV)** specifies whether the local LU accepts conversation requests that include security information from a partner LU. Security information can include a password, an identifier, an optional profile, and other indicators. If no session limits structure is provided on the CNOS request, VTAM uses as the default the value provided by the SECACPT operand on the APPL definition statement. This field is either YES or NO. Because this value must be consistent across all sessions between the LU_mode pair, this value is significant only when the session limits are being changed from 0 to a nonzero value. Therefore, this field is used only for a CNOS request that is initializing session limits from 0 to a nonzero value. It is ignored for a CNOS request that simply changes session limits from one nonzero value to another.
  - **B'0' (NO)** indicates that the LU will not accept FMH-5s that include security fields.
  - **B'1' (YES)** indicates that the LU accepts FMH-5s that include security fields.
- **DDRAINL (SLCDDRAL)** specifies whether the local LU accepts permission on a CNOS request to drain its allocation requests. This field applies only when the session limit is reset to 0. The application program can indicate either the value ALLOW or NALLOW.
  - **B'0' (NALLOW)** specifies that VTAM does not accept a CNOS request that specifies that the LU can drain its allocation requests. If DDRAINL=NALLOW is specified, VTAM negotiates the drain target value to indicate that no draining is to be performed.
  - **B'1' (ALLOW)** specifies that VTAM accepts a CNOS request that specifies that the LU can drain its allocation requests. If DDRAINL=ALLOW is specified, VTAM does not negotiate the drain target value that is carried in the received CNOS request.
- **DEFINE (SLCDEFND)** specifies whether ATTN CNOS is scheduled because a MODIFY DEFINE command is issued or because a CNOS request is processed.
  - **B'0' (NO)** specifies that ATTN CNOS is scheduled because a CNOS request is processed. This CNOS request may be the result of the following conditions:
    - MODIFY CNOS issued locally.
    - Automatic internal MODIFY CNOS issued locally.
  - **B'1' (YES)** specifies that ATTN CNOS is scheduled because the MODIFY DEFINE command is issued.
- **DMINWNL (SLCDMCWL)** contains the number of parallel sessions for which the application program is guaranteed to be the contention winner. This value is used to negotiate limits only when VTAM receives a CNOS initiated by the partner. The specified number can range from 0–32767.

  This field does not apply to single-session connections. For an initial CNOS request, the partner might not be identified as single-session.
- **DMINWNR (SLCDMCWP)** contains the number of parallel sessions for which the partner LU is guaranteed to be the contention winner. This value is used to negotiate limits only when VTAM receives a CNOS initiated by the partner. The specified number can range from 0–32767.

  This field does not apply to single-session connections.
- **DRAINL (SLCDRAL)** specifies whether the source side can drain its allocation requests. For parallel-session connections, the target side cannot negotiate this value. This field applies only if the session limit is being reset to 0. The application program can indicate either YES or NO for all mode names except the SNASVCMG mode name. If the SNASVCMG mode name is specified, VTAM sets DRAINL to NO.
  - **B'0' (NO)** specifies that the source side cannot drain its allocation requests. All requests currently awaiting allocation or subsequently issued at the source side are rejected with a return code indicating the session limit is 0.
  - **B'1' (YES)** specifies that the source side can drain its allocation requests. The source side continues to allocate conversations to the sessions until no requests are awaiting allocation, at which time its draining is ended. Allocation requests that are issued at the source side subsequent to this issuance of the APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS macroinstruction while draining is

taking place are added to the source side's queue of allocation requests to be serviced. All allocation requests issued at the source side after draining is ended are rejected with a return code indicating the session limit is 0.

If the session limits structure is not specified, DRAINL defaults to NO.

- **DRAINR (SLCDRAP)** specifies whether the target side can drain its allocation requests. This field applies only if the limit is being reset to 0. The application program can indicate either YES or NO for all mode names except the SNASVCMG mode name. If the SNASVCMG mode name is specified, VTAM sets DRAINR to NO.
  - **B'0' (NO)** specifies that the target side cannot drain its allocation requests. All requests currently awaiting allocation or subsequently issued at the target side are rejected with a return code indicating the session limit is 0. For parallel-session connections, the target side cannot negotiate this field.

    If DRAINR=NO is specified, this field is accepted for all mode names, regardless of the session limit currently in effect.
  - **B'1' (YES)** specifies that the target side can drain its allocation requests. The target side continues to allocate conversations to the sessions until no requests are awaiting allocation, at which time its draining is ended. All allocation requests issued at the target side after draining is ended are rejected with a return code indicating the session limit is 0. For parallel-session connections, the target side can negotiate this value to NO, in which case the target side cannot drain its allocation requests.

  If the session limits structure is not specified, DRAINR defaults to NO.
- **DRESPL (SLCDRSPL)** specifies whether the local LU is willing to assume responsibility for deactivating sessions if a CNOS request is received that specifies it as the responsible LU. This value is for CNOS negotiation. The application program can indicate a value of either ALLOW or NALLOW.
  - **B'1' (ALLOW)** specifies that VTAM accepts a CNOS request that specifies that the LU is responsible for deactivating sessions. If DRESPL=ALLOW is specified, VTAM does not negotiate the responsibility value carried in the received CNOS request.
  - **B'0' (NALLOW)** specifies that VTAM does not accept a CNOS request that specifies that the LU is responsible for deactivating sessions. If DRESPL=NALLOW is specified, VTAM negotiates the responsibility value for the CNOS reply to be the sender of the CNOS request.
- **DSESLIM (SLCDSESL)** contains the value for the maximum mode name group session limit that VTAM is to accept when it receives a CNOS request. The specified number can range from 0–32767.

  This field is only for the target side of the CNOS negotiation. It does not apply to single-session connections.
- **MINWINL (SLCMCWL)** specifies the number of parallel sessions of which the source side is guaranteed to be the contention winner.

  For parallel-session capable LUs, the specified session limit value can range from 0–32767, except for the SNASVCMG mode, which must have a value of 1 (when SESSLIM=2) or 0 (when SESSLIM=0). For single-session capable LUs, the specified session limit must be 1 or 0.

  The value of MINWINL is determined as follows:
  - If you specify a CNOS session limits control block on the APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS macroinstruction, specify a MINWINL value to use for the CNOS negotiation.
  - If you do not specify a CNOS session limits control block, the current value of the DMINWNL operand of the last APPCCMD CONTROL=OPRCNTL, QUALIFY=DEFINE macroinstruction or the DMINWNL operand on the LU's APPL definition statement is used. The value of the APPL definition statement is used until an APPCCMD (CNOS or DEFINE) provides a new value. For APPL definition statement default values, refer to the z/OS Communications Server: SNA Resource Definition Reference.
- **MINWINR (SLCMCWP)** specifies the number of parallel sessions of which the partner LU is guaranteed to be the contention winner.

  For parallel-session capable LUs, the specified session limit value can range from 0–32767, except for the SNASVCMG mode, which must have a value of 1 (when SESSLIM=2) or 0 (when SESSLIM=0). For single-session capable LUs, the specified session limit must be 1 or 0.

The value of MINWINR is determined as follows:

- If you specify a CNOS session limits control block on the APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS macroinstruction, specify a MINWINR value to use for the CNOS negotiation.
- If you do not specify a CNOS session limits control block, the value of the current DMINWNR operand of the last APPCCMD CONTROL=OPRCNTL, QUALIFY=DEFINE macroinstruction or the DMINWNR operand on the LU's APPL definition statement is used. The value of the APPL definition statement is used until an APPCCMD (CNOS or DEFINE) provides a new value. For APPL definition statement default values, refer to the z/OS Communications Server: SNA Resource Definition Reference.

- **NBRMODE (SLCALL)** specifies whether the resetting of the session limit and contention-winner polarity values to 0 is to apply for only the specified mode name or for all mode names that apply to the partner LU. This field is specified when the application program is changing the session limit and contention-winner polarity values to 0. The application program can indicate either ONE or ALL.
  - **B'0' (ONE)** specifies that the session limit and contention-winner polarity values for only the mode name specified in the LOGMODE operand on the APPCCMD are to be reset to 0.
  - **B'1' (ALL)** specifies that the session limit and contention-winner polarity values for all mode names that apply to the partner LU are to be reset to 0 (the request is rejected if ALL is specified with a nonzero session limit), except for SNASVCMG, which remains unchanged. ALL is applicable only to parallel-session connections.

    If the session limits structure is not specified, the value of the NBRMODE parameter defaults to ONE.
- **PRSISTVL (SLCLPV)** specifies whether persistent-verification indicators can be accepted by the local LU on conversation requests. If YES is specified, the value of the CONVSECL parameter must also be YES. See the "CONVSECL (SLCLCONV)" description in this section for more information.
  - **B'0' (NO)** specifies that the local LU does not accept persistent-verification indicators.
  - **B'1' (YES)** specifies that the local LU does accept persistent-verification indicators.
- **RESP (SLCPRSPL)** specifies which LU is responsible for deactivating the sessions as a result of decreasing the session limits. The application program can indicate a value of either LOCAL or REMOTE if the partner LU is parallel-session capable and the mode name is not the SNASVCMG mode name. Otherwise, VTAM sets the RESP parameter to LOCAL.
  - **B'0' (LOCAL)** specifies that the source side is responsible for deactivating sessions. The target side cannot negotiate this value.
  - **B'1' (REMOTE)** specifies that the target side is responsible for deactivating sessions. The target side can negotiate this value to LOCAL, in which case the local LU becomes responsible.

  The DRAINL and DRAINR fields determine when the responsible LU can deactivate the sessions. (For DRAINL, if the local LU is responsible for deactivating the session, VTAM does this on behalf of the LU.)
  - If an LU is to drain its allocation requests, it continues to allocate conversations to active sessions. The responsible LU deactivates a session only when the conversation allocated to the session is deallocated and no request is awaiting allocation to any session with the specified LU name and mode name. The allocation of a waiting request takes precedence over the deactivation of a session.
  - If an LU is not to drain its allocation requests, the responsible LU deactivates a session as soon as the conversation allocated to the session is deallocated. If no conversation is allocated to the session, the responsible LU deactivates the session immediately.

  Active conversations are not deallocated abnormally. If the session limits structure is not specified, the value of the RESP parameter defaults to LOCAL. Although both VTAM and the application program can act as the LU, the application program never deactivates a session.
- **SESSLIM (SLCSESSL)** specifies the maximum number of sessions that can be activated between the source side LU and the target side LU on a particular logon mode.

  For parallel-session capable LUs, the specified session limit value can range from 0–32767, except for the SNASVCMG mode, which must have a value of 0 or 2. For single-session capable LUs, the specified session limit must be 0 or 1.

  The value of SESSLIM is determined as follows:

- If you specify a CNOS session limits control block on the APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS macroinstruction, specify a SESSLIM value to use for the CNOS negotiation.
- If you do not specify a CNOS session limits control block, the current value of the SESSLIM operand of the last APPCCMD CONTROL=OPRCNTL, QUALIFY=DEFINE macroinstruction or the DSESLIM operand on the LU's APPL definition statement is used. The value of the APPL definition statement is used until an APPCCMD (CNOS or DEFINE) provides a new value. For APPL definition statement default values, refer to the z/OS Communications Server: SNA Resource Definition Reference.

- **SNGSESLU (SLCSSLU)** specifies that the partner LU is a single-session capable LU, and does not support parallel sessions or LU=OWN conversations. VTAM uses this parameter to determine the indication of parallel-session support and CNOS support that it specifies in BIND requests and responses. VTAM examines the value of the SNGSESLU parameter only if an LU entry does not exist for the partner LU in the LU-mode table. This is the case for the initial CNOS for a mode name with a partner LU. When the supplied information indicates the partner LU is single-session-capable, the partner LU's fully qualified name (FQNAME) is not available and the name length (FQNLEN) is 0 until a successful allocation completes. The application must check the FQNLEN field and make sure it is not 0 before checking the FQNAME field.

  The application program can indicate a value of either YES or NO.

  - **B'0' (NO)** specifies that the partner LU supports parallel sessions and CNOS exchanges. If this bit is set off, VTAM activates a SNASVCMG session as the first session with the partner LU. If the partner LU does not support parallel sessions, it can either negotiate the SNASVCMG mode name BIND to single-session support, or reject the SNASVCMG mode name BIND indicating that parallel sessions are not supported. For a SNASVCMG mode name BIND negotiated to single-session support, VTAM ends the SNASVCMG mode name session and marks the partner LU as single-session. Any session activation requests or responses after the session capability has been determined specify that parallel sessions and CNOS are not supported.
  - **B'1' (YES)** specifies that the partner LU does not support parallel sessions or CNOS exchanges. If this bit is set on for the initial CNOS request for a mode name, VTAM subsequently sets off the parallel-session and CNOS support indicators in the session parameters (used in any session activation requests or responses sent to the partner LU) to indicate that the partner LU is parallel-session-capable. If the bit is set on, VTAM does not activate a SNASVCMG session while processing the CNOS request.

  If the session limits structure is not specified, the SNGSESLU parameter defaults to NO.

Draining and session deactivation responsibility

Application programs also use draining and session deactivation responsibility to manage sessions.

Draining

Draining responsibility is negotiated during processing of a CNOS request.

Draining in VTAM LU 6.2

In addition to conversations already in progress, either or both LUs can have conversation requests outstanding from previous allocation macroinstructions. When negotiating session limits to 0 with an APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS macroinstruction, the LUs can specify whether to honor these queued requests before deactivating sessions. The process of honoring queued allocation requests in these circumstances is called draining.

Each LU can set draining to YES or NO. The value YES indicates that the queued allocation requests are given a session. The value NO indicates that the queued requests are failed and that appropriate return codes are passed back on an APPCCMD CONTROL=ALLOC macroinstruction. Draining allows outstanding transaction program requests to be honored before sessions with the partner LU are deactivated.

If draining is allowed, new requests that are received before draining is complete are also honored. If allocation requests are arriving faster than they can be honored, the queue could theoretically grow

during draining. As soon as VTAM can empty the queue, draining is finished and new requests are rejected.

How an application program indicates draining

An application program indicates:

- Draining responsibility by issuing the APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS macroinstruction and specifying the CNOS session limits control block where the values are set. DRAINL (SLCDRAL) is used for the source side and DRAINR (SLCDRAP) for the target side of the CNOS request.
- Draining acceptance by issuing the APPCCMD CONTROL=OPRCNTL, QUALIFY=DEFINE macroinstruction and specifying the DEFINE control block where the values are set. The DDRAINL (SLDDDRAL) value is used when the LU is the target side of a CNOS request. This macroinstruction is used when an application program is defining its negotiation values to be used for a CNOS request.

When VTAM builds and initializes the CNOS control block and a mode entry has not been added to the LU-mode table, VTAM uses the values from the APPL definition statement or the VTAM default values.

Negotiating draining capability

The source LU can specify draining capability for itself and for its partner LU (target) when issuing a CNOS request. It would use DRAINL (SLCDRAL) and DRAINR (SLCDRAP), respectively.

The target LU cannot negotiate the draining capability of the source LU that makes the CNOS request. The target LU must accept what the source LU has specified. The target LU can negotiate the draining capability for its own side to indicate that it will not enable draining on the target side.

The source LU that issues the CNOS request can prevent draining on the target side by setting the value of the DRAINR parameter to NO in the CNOSsession limits control block.

The draining options for the target LU are shown in Table 18 on page 105.

Table 18. Draining options for the target LU

Source Side LU (DRAINR in CNOS structure (SLCNS))	Target Side LU (DDRAINL in DEFINE structure (SLD))	Target Side Options
NO	YES	Cannot support draining
NO	NO	Cannot support draining
YES	NO	Negotiates draining support to NO and does not do draining
YES	YES	Supports draining

When the CNOS request applies to all mode names, a value of YES for the DRAINR parameter in the CNOS control block is ignored for a particular mode name if draining on the target side for that mode name is currently not enabled and the current session limit for that mode name is 0.

Terminating draining

If draining is enabled and requested conversations are taking too long to complete, an application program can try to terminate draining. A conversation already in progress is not interrupted, but the draining indicator can be changed. If the indicator is changed, queued conversation requests are not honored.

Application programs can alter the draining indicator by issuing an APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS macroinstruction with the appropriate draining indicator bits set off in the CNOS session limits control block. (In addition to the SLCDRAP bit, the SLCPARMS field contains the SLCDRAL bit, the source side draining indicator bit.)

If a CNOS request is issued that specifies a value of YES for the DRAINL parameter for a mode name with a session limit of 0 and draining is not enabled on the source side for that mode name, the macroinstruction is rejected with a parameter error indication.

Session deactivation

Session deactivation responsibility determines which LU is responsible for deactivating sessions to honor new session limits for a mode name group. This value is negotiated during the processing of a CNOS request in which session limits are being lowered or set to 0. When a CNOS request is issued to decrease session limits, sessions are deactivated to reach the lowered session limits.

The application program indicates:

- Deactivation responsibility by issuing the APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS macroinstruction and specifying the CNOS control block where the values are set.
- Deactivation acceptance by issuing the APPCCMD CONTROL=OPRCNTL, QUALIFY=DEFINE macroinstruction and specifying the DEFINE control block where the values are set.

When VTAM builds and initializes the CNOS control block and a mode entry has not been added to the LU-mode table, VTAM uses the values in the APPL definition statement or the VTAM default values. When VTAM builds the CNOS control block for the source side, the source-side VTAM deactivates sessions.

The indication for responsibility is either LOCAL, which refers to the source LU, or REMOTE, which refers to the partner LU. If REMOTE is specified, the partner LU's side of the session is responsible for deactivating sessions. The partner LU does not handle deactivation if the partner is only single-session-capable or if the mode name in question is SNASVCMG.

The deactivation bits and their locations are shown in Table 19 on page 106.

Table 19. Deactivation bits and location

Bit	Control Block
SLCDRSPL	CNOS control block
SLDDRSPL	DEFINE/DISPLAY

In the case of single sessions or SNASVCMG sessions, if the local LU has permission to deactivate the session, VTAM deactivates the session on behalf of the local LU. (Single sessions and SNASVCMG sessions are the local LU.)

Application programs can divide session deactivation responsibility for different mode names.

Security acceptance information

The LUs notify one another of the level of conversation-level security that they accept on a logon mode.

Security level definition

The application program specifies the acceptance level for security using any of the following macroinstructions:

- APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS macroinstruction specifying the CNOS session limits control block
- APPCCMD CONTROL=OPRCNTL, QUALIFY=DEFINE macroinstruction specifying the DEFINE control block (target LU uses to negotiate acceptance)
- Alternate BIND using the APPCCMD CONTROL=OPRCNTL, QUALIFY=ACTSESS macroinstruction to modify the session parameters.

If the application program issues a CNOS request but does not build a CNOS control block, VTAM builds a CNOS control block and uses the values on the APPL definition statement or the VTAM default values. For

information on coding the APPL definition statement, refer to z/OS Communications Server: SNA Resource Definition Reference.

Overriding the default security acceptance level

The application program can override the default security acceptance value found in the LU-mode table. To do this, it specifies the security acceptance level (CONVSECL, ALRDYVL, PRSISTVL) on the CNOS session limits control block (ISTSLCNS) when it issues the APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS macroinstruction.

Security manager product

If you have a security manager product, it can limit the highest level of security that can be defined using the CNOS request or the APPL definition statement. (A security manager product is required for session-level security.)

If the application program specifies a security acceptance level using the APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS macroinstruction or the APPCCMD CONTROL=OPRCNTL, QUALIFY=ACTSESS macroinstruction (to modify the BIND response fields) and the security manager product also specifies a security acceptance level, VTAM ensures that the acceptance level defined by either macroinstruction is not greater than the acceptance level defined by the security manager product. (A BIND can have a security level specified in a SCIP or LOGON exit routine.)

Security indicators and subfields

The level of conversation-level security defines the fields in the FMH-5 that the application program can use to implement conversation-level security. The FMH-5 can have both security indicators and subfields. The subfields can contain user identifiers, passwords, and profiles that the application program can use to implement conversation-level security.

For more information on the fields in the FMH-5 that are related to conversation-level security, see “Verifying end users using conversation-level security” on page 255. For more information on the architected LU 6.2 verbs necessary to implement a security program, refer to *SNA Transaction Programmer's Reference Manual for LU Type 6.2*.

Defining negotiation limits and displaying session limits

The DEFINE/DISPLAY control block (ISTSLD) is used to:

- Specify defined negotiation limits that VTAM uses in CNOS negotiation.
- Display CNOS session limits that are currently in effect.

You can add defined negotiation limits to the LU-mode table using the APPCCMD CONTROL=OPRCNTL, QUALIFY=DEFINE macroinstruction and specifying the DEFINE control block where the values are set.

You can display the values currently being used for CNOS negotiations using the APPCCMD CONTROL=OPRCNTL, QUALIFY=DISPLAY macroinstruction to specify the DISPLAY control block where the values are returned.

Initializing and pointing to the control block

Depending on which APPCCMD macroinstruction the application program issues, the application program must initialize the DEFINE/DISPLAY control block and supply an area for the control block.

When to Initialize the DEFINE/DISPLAY control block fields

Before issuing an APPCCMD CONTROL=OPRCNTL, QUALIFY=DEFINE macroinstruction, the application program initializes the DEFINE/DISPLAY control block fields to the specific values to be used for setting defined negotiation limits. The application program determines what values are to be used.

When to supply a 68-byte area for the control block

When issuing either the DEFINE or the DISPLAY macroinstruction, the application program supplies a pointer to the control block in the AREA field of the RPL. The AREALEN field in the RPL should be set to decimal 68, the length of the control block.

When to supply a 40-byte area for the control block

If the APPCCMD CONTROL=OPRCNTL, QUALIFY=DISPLAY macroinstruction is issued with the LOGMODE field specified as 0, or the LOGMODE field of the RPL extension is set to 0, VTAM returns only the following data to the application program:

- LU-related fields in the control block
- Session capability of the specified partner LU
- Some security information about the partner LU
- Partner LU's network-qualified name
- Sync point level
- Negotiated conversation capability

Because these values are contained in the first 40 bytes of the control block, the application program can supply an area of only 40 bytes. For more information on setting the control block address and length in the RPL fields, refer to z/OS Communications Server: SNA Programmer's LU 6.2 Reference.

Limitations of the display function

Until VTAM receives a positive BIND response with a user data field containing subfield TYPE=X'05' (the SLU network name), the partner LU's name is not in the LU-mode table. In this situation you cannot display the partner LU's name using the APPCCMD CONTROL=OPRCNTL, QUALIFY=DISPLAY macroinstruction or using the DISPLAY, CNOS command and specifying the partner. Therefore, if an application issues a CNOS macroinstruction and the RCPRI, RCSEC is X'0000',X'0004', a subsequent DISPLAY may not contain the SLU network name until a successful ALLOC is issued and a positive BIND response is received.

Layout of the DEFINE/DISPLAY control block

Table 20 on page 108 shows the layout of the DEFINE/DISPLAY session limits control block. The field descriptions of the DEFINE/DISPLAY session limits control block follow the layout.

Table 20. Layout of the DEFINE/DISPLAY session limits control block

Byte (Hex)	Bit	Symbolic Name	DSECT Name	Usage
0	0–1	SESSCAP[2.]	SLDSCAP	Session capability of partner LU
0	2–3	SYNCLVL[2]	SLDSYNCH	Negotiated synchronization level
0	4–7	(reserved)		
1	0	CONVSECV[2]	SLDCLSV	Whether security information is valid
1	1	CONVSECP[2]	SLDPCLSA	Whether the partner LU accepts security access subfields
1	2	ALRDYVP[2]	SLDPAVFA	Whether the partner LU accepts the already-verified indicator
1	3	PRSISTVP[2]	SLDPPV	Whether the partner LU accepts the persistent-verification indicator

Table 20. Layout of the DEFINE/DISPLAY session limits control block (continued)

Byte (Hex)	Bit	Symbolic Name	DSECT Name	Usage
1	4	CONVSECL[1]	SLDLCLSA	Whether the local LU accepts security access subfields
1	5	ALRDYVL[1]	SLDLAVFA	Whether the local LU accepts the already-verified indicator
1	6	PRSISTVL[1]	SLDLPV	Whether the local LU accepts the persistent-verification indicator
7	(reserved)			
2	—	FQNLEN[2]	SLDFQNLN	Length of partner LU's fully-qualified name
3–13	—	FQNAME[2]	SLDFQNAM	Partner LU's fully-qualified name
14	0—1	CONVCAP[2]	SLDCNVCP	Negotiated conversation capability of partner LU
14	2—7	(reserved)		
15	—	NAMEUSE[2]	SLDNMUSE	Form of name being used by the local LU on sessions with the partner LU
16	—	NMTYPE[2]	SLDTYPE	Type of LU entry in the LU-mode table
17–27	—	(reserved)		
28–29	—	DSESLIM[1]	SLDDSESL	The value of the maximum session limit for the mode name. This value is used when the application program is the target side of a CNOS request.
2A–2B	—	DMINWNL[1]	SLDDMCWL	The number of parallel sessions of which the application program is the contention winner. This value is used when the application program is the target side of a CNOS request.
2C–2D	—	DMINWNR[1]	SLDDMCWP	The number of parallel sessions of which the partner LU is the contention winner. This value is used when the application program is the target side of a CNOS request.
2E	0	DRESPL[1]	SLDDRSPL	Whether the LU is willing to be responsible for session deactivations. This value is used when the application program is the target side of a CNOS request.

Table 20. Layout of the DEFINE/DISPLAY session limits control block (continued)

Byte (Hex)	Bit	Symbolic Name	DSECT Name	Usage
2E	1	DDRAINL[1]	SLDDDRAL	Whether the LU accepts permission to drain its allocation requests. This value is used when the application program is the target side of a CNOS request.
2E	2	DELETE[1]	SLDDELET	Whether a mode name entry can be deleted from the LU-mode table
2E	3	AUTOSET[3]	SLDAUTOS	Whether the AUTOSES field can override the number of contention-winner sessions started automatically
2E	4	DELUNUSE	SLDDNUNS	Whether unusable logmode entries are being deleted on a MODIFY DEFINE command
2E	5–7	(reserved)		
2F	0	DRAINL[2]	SLDDRAL	Whether the application program can drain its allocation requests
2F	1	DRAINR[2]	SLDDRAP	Whether the partner LU can drain its allocation requests
2F	2–7	(reserved)		
30–31	—	SESSLIM[2]	SLDSESSL	Returns the session limit for the current logon mode name group
32–33	—	MINWINL[2]	SLDMCWL	Returns the current number of parallel sessions of which the application program is to be the contention winner
34–35	—	MINWINR[2]	SLDMCWP	Returns the current number of parallel sessions of which the partner LU is to be the contention winner
36–37	—	AUTOSES[1]	SLDAUTO	Specifies or returns the number of contention-winner sessions that VTAM starts automatically for the local LU after the CNOS negotiation
38–39	—	SESSCNT[2]	SLDSESSC	Returns the number of active mode name group sessions
3A–3B	—	WINLCNT[2]	SLDWINLC	Returns the number of currently active sessions in which the application program is the contention winner

Table 20. Layout of the DEFINE/DISPLAY session limits control block (continued)

Byte (Hex)	Bit	Symbolic Name	DSECT Name	Usage
3C–3D	—	WINRCNT[2]	SLDWINPC	Returns the number of currently active sessions in which the partner LU is the contention winner
3E–3F	—	FREECNT[2]	SLDFREEC	Returns the number of free, active sessions
40–41	—	QALLOC[2]	SLDQALLC	Returns the number of queued ALLOCATE requests
42–43	—	(reserved)		

The following notes apply to the entries in the Symbolic Name field.

[1] This field specifies input for the DEFINE command and returns output for the DEFINE and DISPLAY commands.

[2] This field returns only output.

[3] This field specifies only input.

In the following description of the control block, the symbolic name of the field is given, followed by the actual DSECT label in parentheses.

- **ALRDYVL (SLDLAVFA)** indicates that the local LU accepts the already-verified indicator on conversation requests.
  - **B'0' (NO)** specifies that the local LU does not accept already-verified requests.
  - **B'1' (YES)** specifies that the local LU does accept already-verified requests.
- **ALRDYVP (SLDPAVFA)** indicates that the partner LU accepts the already-verified indicator on conversation requests.
  - **B'0' (NO)** specifies that the partner LU does not accept already-verified requests.
  - **B'1' (YES)** specifies that the partner LU does accept already-verified requests.
- **AUTOSES (SLDAUTO)** returns the value specified by the application program for the number of contention-winner sessions that the application program requests VTAM to activate automatically.
- **AUTOSET (SLDAUTOS)** specifies whether the content of the AUTOSES field is to be used to override the default values provided by the APPL definition statement. If a value is provided but the bit is not set, VTAM uses the default values from the APPL definition statement.
- **CONVCAP (SLDCNVCP)** indicates the negotiated conversation capability of the partner LU.
  - **B'10' (FULL-DUPLEX)** indicates that sessions with the partner LU are capable of full-duplex or half-duplex conversations and expedited data is allowed.
  - **B'01' (HALF-DUPLEX)** indicates that sessions with partner LU are capable of half-duplex conversations only.
  - **B'00' (UNKNOWN)** indicates that the conversation capability of the partner LU is unknown.
- **CONVSECL (SLDLCLSA)** indicates whether the application program accepts FMH-5s that include access security subfields.
  - **B'0' (NO)** indicates that the application program does not accept security information when receiving allocation requests.
  - **B'1' (YES)** indicates that the application program does accept security subfields on an FMH-5.
- **CONVSECP (SLDPCLSA)** indicates whether the partner LU accepts FMH-5s that include access security subfields.

  - **B'0' (NO)** indicates that the partner LU does not accept security information when receiving allocation requests.
  - **B'1' (YES)** indicates that the partner LU does accept security subfields on an FMH-5.
- **CONVSECV (SLDCLSV)** indicates whether the returned security information in the control block is valid. VTAM does not set the security information until the first session is established with a partner LU.
  - **B'0' (NO)** indicates that a session has not been established with the partner LU; therefore, the security information in the control block cannot be used.
  - **B'1' (YES)** indicates that a session has been established and that the security information in the control block can be used.
- **DELUNUSE (SLDDNUNS)** indicates that the MODIFY DEFINE command is being used to delete LU-mode table entries that are marked UNUSABLE. For more information about the types of LU-mode table entries, see "LU entries in the LU-mode table" on page 94. For more information about using the MODIFY DEFINE command, refer to z/OS Communications Server: SNA Operation.
- **DDRAINL (SLDDDRAL)** specifies whether VTAM should accept permission on a CNOS request, on behalf of the local LU, to drain the local LU's allocation requests. This field applies to the local LU when it is the target. (The local LU can be either the source or the target.) This field applies only if the session limit is being reset to 0.
  - **B'1' (ALLOW)** specifies that VTAM accepts a CNOS request specifying the LU can drain its allocation requests. If DDRAINL=ALLOW is specified, VTAM performs no negotiation of the drain-target value carried in the received CNOS request.
  - **B'0' (NALLOW)** specifies that VTAM does not accept a CNOS request specifying the LU can drain its allocation requests. If DDRAINL=NALLOW is specified, VTAM negotiates the drain-target value to indicate no draining is to be performed.

  For single-session connections, this field is not applicable.

  Note: If a CNOS request for *all* mode names is received, the DDRAINL parameter is used to set the DRAINL value for each mode name associated with the LU. The DDRAINL value has to be set for each mode name.
- **DELETE (SLDDELET)** indicates whether a mode name entry can be considered for deletion from the LU-mode table. This field applies only if the session limit is being reset to 0.
  - **B'1' (ALLOW)** specifies that a mode name entry can be considered for deletion from the LU-mode table.
  - **B'0' (NALLOW)** specifies that a mode name entry will not be considered for deletion from the LU-mode table.

  Note: Deletion saves storage because subsequent CNOS requests, if any, use the values in the APPL definition statement.
- **DMINWNL (SLDDMCWL)** contains the number of parallel sessions of which the application program is guaranteed to be the contention winner. This value is used when the LU is the target side of the negotiation and also when the LU is the source side but did not specify session negotiation limits on the APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS macroinstruction. (When an LU sends a CNOS request to the local LU, the local LU becomes the target .) The specified number can range from 0–32767.

  For single-session connections, this field is not applicable.
- **DMINWNR (SLDDMCWP)** contains the number of parallel sessions of which the partner LU is guaranteed to be the contention winner. This value is used for CNOS negotiation. The specified number can range from 0–32767. This value is used when the LU is the target side of the negotiation and also when the LU is the source side but did not specify session negotiation limits on the APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS macroinstruction.

  For single-session connections, this field is not applicable.
- **DRAINL (SLDDRAL)** indicates whether the application program currently enables draining of allocation requests. This field applies only if the session limit is 0. This field also is not applicable to the SNASVCMG sessions. This is not the defined drain-local parameter, as provided on the APPL

definition statement or the APPCCMD CONTROL=OPRCNTL, QUALIFY=DEFINE macroinstruction. It is the actual drain-local parameter supplied through the APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS macroinstruction or agreed upon during a CNOS negotiation. (The defined drain-local parameter is returned to the application program through the DDRAINL field described in this section.)

  - **B'0' (NO)** specifies that the LU cannot drain its allocation requests. All requests currently awaiting allocation, or subsequently issued, at the LU are rejected with a return code indicating the session limit is 0.
  - **B'1' (YES)** specifies that the LU can drain its allocation requests. The LU continues to allocate conversations to the sessions until no requests are awaiting allocation, at which time its draining is ended. Allocation requests that are issued at the LU while draining is taking place are added to the LU's queue of allocation requests to be serviced. All allocation requests issued at the LU after draining is ended are rejected with a return code indicating the session limit is 0.

- **DRAINR (SLDDRAP)** indicates whether the partner LU can drain its allocation requests. This field applies only if the limit is 0.

  - **B'0' (NO)** specifies that the partner LU cannot drain its allocation requests. All requests currently awaiting allocation or subsequently issued at the partner LU are rejected with a return code indicating the session limit is 0. For parallel-session connections, the partner LU cannot negotiate this field.
  - **B'1' (YES)** specifies that the partner LU can drain its allocation requests. The partner LU continues to allocate conversations to the sessions until no requests are awaiting allocation, at which time its draining is ended. All allocation requests issued at the partner LU after draining is ended are rejected with a return code indicating the session limit is 0.

  Note: Because of the method of negotiating a CNOS request for all mode names, the value of the DRAINR parameter cannot be the same as the value of the partner LU's DRAINL parameter.

- **DRESPL (SLDDRSPL)** specifies whether the LU is willing to assume responsibility for deactivating sessions if a CNOS request is received specifying it as the responsible LU. VTAM handles this responsibility on behalf of the local LU. This DRESPL value is used when the LU is the target side of the negotiation. The application program can indicate either ALLOW or NALLOW.

  - **B'1' (ALLOW)** specifies that VTAM accepts a CNOS request specifying that the LU is responsible for deactivating sessions. If DRESPL=ALLOW is specified, VTAM performs no negotiation of the responsibility value carried in the CNOS request.
  - **B'0' (NALLOW)** specifies that VTAM does not accept a CNOS request specifying the LU is responsible for deactivating sessions. If DRESPL=NALLOW is specified, VTAM negotiates the responsibility value for the CNOS reply to be the sender of the CNOS request.

  For single-session connections, this field is not applicable.

  Note: When a CNOS request for *all* mode names is received from the partner LU, VTAM assigns the responsibility for deactivating sessions to the source side of the request.

- **DSESLIM (SLDDSESL)** contains the value of the maximum session limits for the mode name group that VTAM is to accept when it receives a CNOS request, or when the LU initiates a CNOS request and did not specify any session limits for CNOS negotiation. The specified number can range from 0–32767.

  This session limit value is used only for CNOS negotiation. For single-session connections, this field is not applicable.

  Note: The value specified for the DSESLIM parameter must be greater than or equal to the value of the DMINWNL parameter plus the value of the DMINWNR parameter.

- **FQNAME (SLDFQNAM)** contains the partner LU's fully qualified name. The FQNLEN field contains the actual length of the name. The fully qualified network name consists of an optional 1- through 8-byte network identifier and a 1- through 8-byte LU name. When present, the network identifier is concatenated to the left of the LU name, using a separating period and has the form `NETID.name` . When the network identifier is omitted, the period is omitted. The fully qualified name is available only after successful establishment of a session with the partner LU.

  Note: The partner LU provides its fully qualified network name in the Userdata subfield of the BIND. Because this name is optional information, it might not be provided. It is returned in the SLD only when

it has been included in the BIND. It is not available until the LU receives a BIND response from the partner LU. If a CNOS request is negotiated with a partner LU that is single-session-capable, the name might not be available. In this case, the length field will be 0. Check the FQNLEN (SLDFQNLN) field to determine whether it is not 0 before checking the name.

- **FQNLEN (SLDFQNLN)** indicates the length of the partner LU's fully qualified name, contained in the FQNAME field. The value of this field can range from 0–17. Zero is returned for the length if the fully qualified name is not available. See the note in the description for the FQNAME field for information about processing this field.
- **FREECNT (SLDFREEC)** returns the number of sessions active with the partner LU that are currently available (free for use by a conversation).
- **MINWINL (SLDMCWL)** returns the current number of parallel sessions of which the application program is guaranteed to be the contention winner. This is not the defined value, as provided on the APPL definition statement or the APPCCMD CONTROL=OPRCNTL, QUALIFY=DEFINE macroinstruction. It is the actual minimum contention-winner value supplied through an APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS macroinstruction or agreed upon during a CNOS negotiation. (The defined value is returned to the application program through the DMINWNL field.)
- **MINWINR (SLDMCWP)** returns the current number of parallel sessions of which the partner LU is guaranteed to be the contention winner. This is not the defined value, as provided on the APPL definition statement or the APPCCMD CONTROL=OPRCNTL, QUALIFY=DEFINE macroinstruction. It is the actual minimum contention-winner value supplied through the APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS macroinstruction or agreed upon during a CNOS negotiation. (The defined value is returned to the application program through the DMINWNR field.)
- **NAMEUSE (SLDNMUSE)** indicates how the application is identified to the partner LU. This field can indicate whether the name used is a user variable, a generic resource name, the application network name, or is unknown.
- **NMTYPE (SLDTYPE)** indicates how the LU-mode table has classified the name for the partner LU. This field can indicate whether the name used is a supplied name, variant name, received name, disassociated name, or unusable.
- **PRSISTVL (SLDLPV)** specifies whether the local LU accepts persistent-verification indicators on conversation requests.
  - **B'0' (NO)** specifies that the local LU does not accept persistent-verification requests.
  - **B'1' (YES)** specifies that the local LU does accept persistent-verification requests.
- **PRSISTVP (SLDPPV)** specifies whether the partner LU accepts persistent-verification indicators on conversation requests.
  - **B'0' (NO)** specifies that the partner LU does not accept persistent-verification requests.
  - **B'1' (YES)** specifies that the partner LU does accept persistent-verification requests.
- **QALLOC (SLDQALLC)** returns the number of queued ALLOCATE requests when a DEFINE or DISPLAY request completes.
- **SESSCAP (SLDSCAP)** is the field in the DEFINE/DISPLAY session limits structure that indicates the session capability of the partner LU. The possible states that can be returned are:
  - **B'01' (PENDING_SINGLE)** indicates that VTAM is in the process of determining the session capability of the partner LU. The initial determination is that the partner LU is not parallel-session capable. This state changes to SINGLE after VTAM has confirmed the session capability. If this does not occur, VTAM deletes the LU entry in the LU-mode table associated with the partner LU.
  - **B'00' (SINGLE)** indicates VTAM has determined the partner LU to be single-session capable.

    When the partner LU is single-session capable (SLDSINGL field of the DEFINE/DISPLAY session limits structure), the partner name might be missing in the returned information because the BIND response has not returned. Check the FQNLEN (SLDFQNLN) field to be sure it is not 0 before checking the FQNAME (SLDFQNAM) field for the partner LU's fully qualified name.
  - **B'10' (PENDING_PARALLEL)** indicates that VTAM is in the process of determining the session capability of the partner LU. The initial determination is that the partner LU is parallel-session

capable. This state changes to either SINGLE or PARALLEL after VTAM has confirmed the session capability. If this does not occur, VTAM deletes the LU entry in the LU-mode table associated with the partner LU.

  - **B'11' (PARALLEL)** indicates VTAM has determined the partner LU to be parallel-session capable.
- **SESSCNT (SLDSESSC)** returns the session count for the number of sessions active currently with the partner LU that have the specified mode name.
- **SESSLIM (SLDSESSL)** is the field in the DEFINE/DISPLAY session limits structure in which the session limit for the current mode name group is returned. This is not the defined session limit, as provided on the APPL definition statement or the APPCCMD CONTROL=OPRCNTL, QUALIFY=DEFINE macroinstruction. It is the actual session limit supplied through the APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS macroinstruction or agreed upon during a CNOS negotiation. (The defined session limit is returned to the application program through the DSESLIM field.)
- **SYNCLVL (SLDSYNCH)** indicates the negotiated level of synchronization.
  - **B'00'** specifies that no synchronization level is supported.
  - **B'01'** specifies that the confirm synchronization level is supported.
  - **B'10'** specifies that the confirm and sync point levels are supported.
- **WINLCNT (SLDWINLC)** returns the number of currently active sessions for which the application program is the contention winner.
- **WINRCNT (SLDWINPC)** returns the number of currently active sessions for which the partner LU is the contention winner.

Example of setting the DEFINE/DISPLAY control block

The following example shows an application program with an ACB name of APPLB setting the control block and issuing the APPCCMD to define negotiation values of overall session limits of 12, 8 contention-winner sessions for APPLB, and 4 contention-winner sessions for the partner LU (12, 8, 4). The indications for draining and mode deletion are set to indicate that the LU allows draining and that this mode can be deleted from the LU-mode table. APPLB is defining these negotiation values for sessions using the EXAMPLE mode with a partner LU called APPLA. (In this example, study the description of the DEFINE/DISPLAY control block and the ISTSLD DSECT. Table 20 on page 108 shows the layout of the DEFINE/DISPLAY control block. “Defining negotiation limits and displaying session limits” on page 107 discusses the fields contained in the control block. The z/OS Communications Server: SNA Programmer's LU 6.2 Reference shows the DSECT.)

```
         LA    10,CBAREA         * LOAD ADDRESS OF CONTROL BLOCK
         USING ISTSLD,10         * ESTABLISH ADDRESSABILITY
         MVC   SLDDSESL,=X'000C' * SET OVERALL SESSION LIMITS FIELD
         MVC   SLDDMCWL,=X'0008' * SET LOCAL CONTENTION WINNERS FIELD
         MVC   SLDDMCWP,=X'0004' * SET PARTNER CONTENTION WINNERS FIELD
         MVI   SLDDEFPA,         * SET DRAINING, DELETE FLAG FIELD BITS X
               SLDDDRAL+SLDDELET
         LA    8,RPLB            * LOAD RPL ADDRESS
         USING IFGRPL,8          * ESTABLISH RPL ADDRESSABILITY
         ST    10,RPLAREA        * PUT CONTROL BLOCK ADDRESS IN RPL
         MVC   RPLRLEN,CBLEN     * PUT CONTROL BLOCK LENGTH IN RPL
*
          APPCCMD CONTROL=OPRCNTL,                                      X
               QUALIFY=DEFINE,                                          X
               RPL=RPLB,                                                X
               AAREA=RPLBX,                                             X
               ACB=APPLB,                                               X
               LUNAME=APPLA,                                            X
               LOGMODE=EXAMPLE
                     .
                     .
                     .
CBAREA   DS    XL68              * STORAGE FOR CONTROL BLOCK
         DS    0F                * ALIGN TO FULLWORD BOUNDARY
CBLEN    DC    XF68              * LENGTH OF CONTROL BLOCK
RPLB     RPL AM=VTAM             * RPL STORAGE
RPLBX    ISTRPL6                 * RPL EXTENSION
APPLB    ACB AM=VTAM,MACRF=LOGON,APPLID=APPLNAME * ACB STORAGE
```

Figure 21. Example of setting the DEFINE/DISPLAY control block

In the example, the instructions to set the RPLAREA field and RPLRLEN field in the RPL could have been omitted by using additional parameters on the macroinstruction. It could have been coded as:

```
         APPCCMD CONTROL=OPRCNTL,                                       X
               QUALIFY=DEFINE,                                          X
               RPL=RPLB,                                                X
               ACB=APPLB,                                               X
               AAREA=RPLBX,                                             X
               LUNAME=APPLB,                                            X
               LOGMODE=EXAMPLE,                                         X
               AREA=CBAREA,                                             X
               RECLEN=68
```

For a more detailed example of the use of the above macroinstruction and the defined negotiation limits in a situation involving a partner LU, see “Example of CNOS negotiation” on page 91.

The APPCCMD CONTROL=OPRCNTL, QUALIFY=DEFINE macroinstruction can be issued before a CNOS request. However, this function benefits only LU partners that support parallel sessions. In other cases, the APPCCMD CONTROL=OPRCNTL, QUALIFY=DEFINE macroinstruction might be issued after a CNOS negotiation involving the partner LU has occurred and the LU entry has been added to the LU-mode table. This might occur because the negotiation values assigned automatically by the APPL definition statement are not appropriate for a subsequent operation. The APPCCMD CONTROL=OPRCNTL,QUALIFY=DEFINE macroinstruction can be used to raise and lower the negotiation values as needed.

Note: While the application program is active, it can use the APPCCMD CONTROL=OPRCNTL, QUALIFY=DISPLAY macroinstruction to determine the session limits. The operator can change the defined negotiation limits or CNOS negotiation limits at any time by issuing the MODIFY CNOS command.

Displaying LU-mode data

An application program can query information in the LU-mode table by using the APPCCMD CONTROL=OPRCNTL, QUALIFY=DISPLAY macroinstruction. The application program specifies the DISPLAY control block on the macroinstruction using the AREA and AREALEN input parameters. When the DISPLAY request completes, the DISPLAY control block fields contain the information from the LU-mode table.

VTAM returns both the actual session limit values in use and the set of defined negotiation limits used for session limit negotiation purposes. It also returns some additional session-related information, such as the number of free sessions available (FREECNT) with a partner LU. (For details on the syntax and

operands for the macroinstruction, refer to z/OS Communications Server: SNA Programmer's LU 6.2 Reference.)

Before the DISPLAY function can be used, information about partner LUs and modes must be stored in the LU-mode table as the result of one of the following items:

- Session-establishment
- CNOS processing
- A prior APPCCMD CONTROL=OPRCNTL, QUALIFY=DEFINE macroinstruction

See Table 20 on page 108 to determine the fields that can be set for CNOS negotiation.

Example of displaying LU-mode data

As an example of displaying data, suppose an application program with an ACB name of APPLA compares the number of active local contention-winner sessions with the session limits value controlling the minimum number of contention-winner sessions guaranteed to the application program. Assume that the partner LU is APPLB and the mode is EXAMPLE. The following code might be used to issue the APPCCMD CONTROL=OPRCNTL, QUALIFY=DISPLAY macroinstruction to retrieve the information:

```
         LA    9,RPLA              * GET RPL ADDRESS
         LA    11,RPLAX            * GET RPL EXTENSION ADDRESS
         LA    10,CBAREA           * GET CONTROL BLOCK ADDRESS
         USING ISTSLD,10           * ESTABLISH ADDRESSABILITY
         L     8,CBLEN             * GET CONTROL BLOCK LENGTH
*
         APPCCMD CONTROL=OPRCNTL,                                      X
               QUALIFY=DISPLAY,                                        X
               RPL=(9),                                                X
               AAREA=(11),                                             X
               ACB=APPLA,                                              X
               OPTCD=SYN,                                              X
               LUNAME=APPLB,                                           X
               LOGMODE=EXAMPLE,                                        X
               AREA=(10),                                              X
               AREALEN=(8)
*
         LTR   15,15               * CHECK GENERAL RETURN CODE IN 15
         BNZ   BADGENRC            * HANDLE NONZERO RETURN CODE
         LTR   0,0                 * CHECK CONDITIONAL COMPLETION
         BNZ   BADCOND             * HANDLE NONZERO RETURN CODE
         CLC   SLDMCWL,SLDWINLC    * RETURN CODES OK - COMPARE VALUES
         BH    LOWROUT             * IF ACTUAL COUNT LESS, GO SOMEWHERE
         B     NOTLOW              * OTHERWISE, GO SOMEWHERE ELSE
           .
           .
           .
CBAREA   DS    XL68                * STORAGE FOR CONTROL BLOCK
         DS    0F                  * ALIGN TO FULLWORD BOUNDARY
CBLEN    DC    X'0000044'          * CONTROL BLOCK LENGTH
RPLA     RPL AM=VTAM               * RPL STORAGE
RPLAX    ISTRPL6                   * RPL EXTENSION STORAGE
APPLA    ACB AM=VTAM,MACRF=LOGON,APPLID=APPLNAME * ACB STORAGE
```

You can specify AREA and AREALEN using a keyword on the macroinstruction or by setting a keyword with a name in the APPL definition statement.

Setting session limits to 0

When session limits are set to 0, the sessions with the specified LU that is using the specified mode name (LU-mode) are deactivated. Sessions that are not needed currently are deactivated immediately. Sessions being used by a conversation are not deactivated until the conversation finishes. In addition, if draining was set on, draining now takes effect. For a description of when sessions are deactivated, see “VTAM's role in session activation and deactivation” on page 121.

If the application program requests 0 session limits on a CNOS negotiation, the session limits cannot be negotiated. The partner LU must accept 0 limits. At the time of this CNOS request to set session limits to 0 the application program can specify that the change is to apply to all mode names with the partner LU except the SNASVCMG mode. The application program does this by setting the NBRMODE (SLCALL) bit on

in the CNOS session limits control block. (NBRMODE can be set to ALL only when session limits are being set to 0.) VTAM can negotiate session draining responsibility and draining capability on the target side.

If the CNOS request is for all mode names and target side draining was specified, VTAM examines the negotiation value for each mode name and enables target side draining only for those modes whose negotiation value allows it. If the CNOS request specified that draining is not enabled on the target side, VTAM accepts that restriction without negotiation.

Closing a mode

Closing a mode means to end all conversations on a mode and to prevent any more conversations on that mode.

To close a mode, the application program must issue two macroinstructions:

- The APPCCMD CONTROL=OPRCNTL, QUALIFY=DEFINE macroinstruction, setting the defined limits to 0 and marking the mode (using the DELETE bit—SLDDELET) as not eligible for deletion from the LU-mode table.
- The APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS macroinstruction, setting session limits for the mode to 0 and specifying DRAINL=NO. (This prevents the LU from honoring future conversation requests.)

The combination of these two macroinstructions issued in this manner ensures that if a CNOS request that specifies that mode name is received from a partner LU, the request is rejected and is returned to the partner LU with a return code of MODE_NAME_CLOSED.

Not allowing the mode to be deleted when session limits are 0 prevents the partner LU from allocating conversations on the mode. To obtain this control, the application program sets the mode to 0 and marks the mode as not to be deleted. This forces the mode to retain 0 as the session limit. Deleting a mode saves storage, but because VTAM no longer has entries for this mode, the partner LU can initialize the mode by issuing another CNOS request.

Closing a SNASVCMG mode

For sessions with parallel-session-capable partners, the SNASVCMG mode name sessions are the last to be deactivated. Application programs should wait until the SNASVCMG sessions with all partners of the local LU are deactivated before closing the ACB. To help the application program determine the current number of active sessions assigned to the partner LU, VTAM returns the RPL6LAST field in the RPL extension when the ATTN(LOSS) exit is scheduled.

Deleting mode entries

To delete a mode entry, the application program issues the following macroinstructions:

- The APPCCMD CONTROL=OPRCNTL, QUALIFY=DEFINE macroinstruction, setting the defined limits to 0 and marking the mode name as a candidate for deletion with the DELETE bit (SLDDELET) set on in the DEFINE/DISPLAY session limits control block.
- The APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS macroinstruction, setting session limits for the mode to 0.

The application program can issue the QUALIFY=DEFINE request either before or after the CNOS request. As long as the negotiation values and session limits are both 0 and the mode name has been marked for deletion, the mode name will be deleted.

Additional session limit considerations

In addition to specifying the negotiation values, you might need to handle the following items:

- Parallel session support
- Session limits for single-session partners

- Session limits for SNASVCMG mode name

Parallel session support

VTAM receives data from the partner LU that indicates the partner's support of parallel sessions. The application program interrogates VTAM to determine the partner's support of parallel sessions.

How VTAM receives indicators

If the partner LU is not parallel-session capable, it indicates this to VTAM in any of the following ways:

- Partner LU receives the BIND.
  - The partner LU can negotiate the BIND and send back a positive response that indicates that it supports only single sessions. (It changes the parallel-session support bit and the CNOS support bit 6 and 7 in byte 24 of the BIND response.) In this case, VTAM deactivates the SNASVCMG session and marks the session partner as single-session capable.
  - The partner LU can reject the BIND, sending a negative BIND response or UNBIND with a sense code of X'0835'*xxxx*, where *xxxx* can be either the offset of the first character of the SNASVCMG mode name or X'0018' (the offset in the BIND of the byte containing the indication of parallel-session support). When this occurs, VTAM concludes that the partner is single-session capable.
- Partner LU sends the BIND.

  If VTAM receives a session-initiation request from the partner LU, VTAM determines from the BIND whether the partner is single- or parallel-session capable.

How an application program interrogates VTAM

To determine the capability of a potential session partner, the application program can issue the APPCCMD CONTROL=OPRCNTL, QUALIFY=DISPLAY macroinstruction with a mode name of X'0' and specify the DISPLAY control block. The returned SLDSCAP field in the DISPLAY control block indicates whether the partner LU supports parallel sessions. (The AREA field in the RPL points to the DEFINE/DISPLAY control block.) This method can be used only after a CNOS request that involves the partner LU has completed.

Note:

1. Even if parallel sessions are not going to be used by an application program, an entry for the SNASVCMG mode should be included in an installation's logon mode table. VTAM can use this mode internally as part of its processing for session limits with single-session partners if VTAM does not already know the session capability of the partner LU.
2. If a CNOS request other than (1,1,0) is specified to a partner LU that is only single-session capable, a session will not be started automatically regardless of the setting of AUTOSES.

Session limits for single-session-capable partners

CNOS negotiation does not occur when session limits are set with a single-session-capable partner. A single-session-capable LU is an LU that can have only one active session at a time with a partner LU. Therefore, the only nonzero session limits it can use are (1,1,0), (1,0,1), or (1,0,0). They can have only one mode with nonzero session limits.

VTAM determines single-session or parallel-session capability in one of the following ways:

- The application program can specify the session capability of the partner with the SLDSCAP bit field in the CNOS session limits control block. Setting the bit on indicates that the partner does not support parallel sessions. (This is applicable only for the first CNOS request involving the partner LU.)
- For an initial CNOS request with a partner LU, where the application program does not set the session capability indicator bit (SLDSCAP), VTAM assumes the partner LU is parallel-session capable and attempts to establish a SNASVCMG mode session to negotiate the session limits. VTAM receives the information regarding the partner's session capability, and the application program issues

the APPCCMD CONTROL=OPRCNTL, QUALIFY=DISPLAY macroinstruction to determine the partner's session capability. (This is true regardless of the mode to which the CNOS request applies.)

- For SLU session initiation, VTAM will use the parallel/single session indicated in the CINIT to force a nonparallel session starting BINPSS=0 on PSERVIC.

The default session limits used for single-session LUs are shown in Table 21 on page 120.

Table 21. Default session limits for single-session partners

How Capability is Detected	Default Session Limits
LU-mode table indicates partner LU is single-session capable.	(1,1,0)
LU-mode table indicates partner LU is pending parallel capable but partner LU actually is single-session capable.	(1,0,0)
BIND negotiation received from single-session-capable LU and no LU entry exists in the LU-mode table.	(1,0,0)

Some types of the APPCCMD CONTROL=ALLOC macroinstruction are not appropriate for single-session LUs. For example, the APPCCMD CONTROL=ALLOC, QUALIFY=CONWIN macroinstruction and the APPCCMD CONTROL=ALLOC, QUALIFY=IMMED macroinstruction are possible for one of the LUs, the contention winner, and the contention-winner role can be reversed at BIND (not at CNOS negotiation). The APPCCMD CONTROL=ALLOC, QUALIFY=ALLOCD macroinstruction is recommended for single-session LUs.

Note: When VTAM receives a BIND request for a single-session-capable LU and no LU-mode entry exists for the LU, VTAM creates an LU-mode entry with session limits of (1,0,0) for the LU. Because of these limits, the application program receiving the BIND cannot successfully issue an APPCCMD CONTROL=ALLOC, QUALIFY=CONWIN macroinstruction once the session is available.

Session limits for SNASVCMG mode name

Additional session limit considerations apply when changing the session limits for the SNASVCMG mode name. This is a special mode name that VTAM uses to exchange information with other LUs in implementing the LU 6.2 architecture. (Issuing a CNOS request to change the session limits for the SNASVCMG mode name alters certain information only for the local LU. The partner LU is not notified of the changes.)

Application programs can issue commands to change the session limits for this mode name. However, the maximum number of sessions can only be 2 on this mode name, and the maximum number of contention-winner or contention-loser sessions can only be 1. Therefore, the session limits for the SNASVCMG mode are set to (2,1,1) when active and (0,0,0) when reset.

Note: If default session limits of (1,1,0) are set for a single-session capable LU and AUTOSES is nonzero, one contention winner session will be activated on behalf of the application.

The session limits for the SNASVCMG mode name can be 0 only if all other session limits with a partner LU are also 0. In addition, no queued conversation requests can be awaiting completion. (For information on how the application program can specify whether queued requests are honored, see “Draining and session deactivation responsibility” on page 104.)

VTAM allows CNOS requests for modes other than the SNASVCMG mode to be issued without requiring the application program to first issue a CNOS request for the SNASVCMG mode. If the application program does not issue a CNOS request for the SNASVCMG mode, VTAM:

- Issues the CNOS request for the SNASVCMG mode on behalf of the application program
- Adds the SNASVCMG mode to the LU-mode table
- Updates the SNASVCMG mode session limit to a value of 2

This allows the successful completion of a CNOS without the additional overhead of issuing a CNOS request against the SNASVCMG mode.

If VTAM handles the CNOS request of the SNASVCMG mode, the results might be slightly different than when the application program issues the CNOS request. For example, if the application program issues the CNOS request for the SNASVCMG mode, the usage of the AUTOSES operand immediately becomes applicable for the SNASVCMG mode. If VTAM handles the CNOS request, the AUTOSES operand is not immediately activated for the SNASVCMG mode; VTAM starts only the sessions it needs for the SNASVCMG.

Activating and deactivating sessions

Both the application program and VTAM can play a role in session activation and deactivation.

VTAM's role in session activation and deactivation

CNOS requests frequently cause sessions to be activated or deactivated.

Note: If a CNOS request is issued for a partner LU that is an independent LU, the request to start a session does not complete if not active SNASVCMG session exists and if one cannot be activated. The request completes only after a session can be established with the independent LU and the CNOS negotiation can be performed.

When VTAM activates sessions

VTAM usually activates sessions only as needed to meet conversation requests. If the session limit is raised as a result of a CNOS negotiation, VTAM can activate sessions up to the new session limit. VTAM does not activate sessions that would exceed the current negotiated session limits.

You can force VTAM to activate sessions as soon as the session limits on a mode are negotiated by using the AUTOSES parameter. If you code the AUTOSES parameter on the APPL definition statement, VTAM activates contention-winner sessions up to the lesser of:

- New minimum number of contention-winner sessions
- Number of sessions specified for the AUTOSES value on the APPL definition statement
- Number of sessions specified for the AUTOSES value on the APPCCMD CONTROL=OPRCNTL, QUALIFY=DEFINE macroinstruction

The AUTOSES value specifies the number of contention winner sessions VTAM will maintain. VTAM automatically reestablishs sessions when AUTOSES is specified. For example, if the minimum negotiated contention winner limit is 5 and the AUTOSES value is 4, after the CNOS negotiation for the mode, VTAM automatically activates four sessions. When a session is deactivated, VTAM automatically starts a new session.

A CNOS that lowers the minimum number of contention winners to a value less than the value of AUTOSES causes VTAM to deactivate sessions as they become free.

Care should be taken when specifying the AUTOSES value. The operator may not be able to bring down unwanted sessions. A terminate request can result in VTAM starting a new session as stated above.

When VTAM deactivates sessions

VTAM deactivates sessions only when they are not currently assigned to a conversation and the number of sessions exceeds the negotiated session limits for the mode, and VTAM is responsible for deactivation. If session limits are lowered as the result of a CNOS negotiation, VTAM deactivates sessions to reach the lower session limits.

The negotiated draining value affects when sessions are deactivated. See “Draining and session deactivation responsibility” on page 104 for more information on draining.

If the application program specifies ATTNLOSS=ALL on the APPL definition statement, VTAM schedules the ATTN(LOSS) exit when each session is deactivated. Otherwise, VTAM schedules the ATTN(LOSS) exit when the last session on the mode is deactivated.

VTAM returns the RPL6LAST field when the ATTN(LOSS) exit is scheduled. This provides information to the application program about the active sessions for the partner LU. RPL6LAST is found in the ISTRPL6 DSECT. The constant values for RPL6LAST are shown in Table 22 on page 122.

Table 22. Constant values for RPL6LAST

Value	Description
RPL6NLST	Sessions exist for the specified mode.
RPL6MOD	Last session deactivated for the specified mode.
RPL6NCTL	Last session deactivated for non-control modes (all but the SNASVCMG mode).
RPL6ALL	All sessions for this LU have been deactivated.

Deactivation of sessions with parallel-session-capable partners

For sessions with parallel-session-capable partners, the SNASVCMG mode name sessions are the last to be deactivated. Application programs should wait until the SNASVCMG sessions with all partners of the local LU are deactivated before closing the ACB (access method control block). To help the application program determine the current number of active sessions assigned to the partner LU, VTAM returns the RPL6LAST field in the RPL extension when the ATTN(LOSS) exit is scheduled.

Application program's role in session activation and deactivation

When CNOS negotiation occurs, session limits might increase or decrease as a result of the negotiation. The resulting change in the session limits might cause VTAM to activate or deactivate sessions. A conversation allocation request also might cause session activation.

Application programs with LOGON and SCIP exits can respond to session activation requests by specifying QUALIFY=ACTSESS or QUALIFY=DACTSESS and supplying a different set of session parameters or stopping a session from being activated.

For VTAM LU 6.2 sessions, LOGON and SCIP exits are scheduled as shown in Table 23 on page 122.

Table 23. LOGON and SCIP exits are scheduled

Exit	Scheduled When
LOGON	VTAM activates a session in response to an allocation request and receives a CINIT request.
SCIP	VTAM receives a BIND as the result of an allocation request by a partner LU, or VTAM receives a session request as the result of a CNOS request by a partner LU.

The LOGON exit is not scheduled when VTAM activates a session for use in negotiating session limits on a CNOS request. (This is a SNASVCMG mode session.)

For detailed information on SCIP and LOGON exits, refer to z/OS Communications Server: SNA Programming.

Determining session type after LOGON exit

After the LOGON exit is driven, the application program checks the CINIT to determine whether the session is an LU 6.2 session. If bytes 14 and 15 of the BIND image in the CINIT are X'0602', the CINIT represents an LU 6.2 session. (The high order bit of byte 14 is not used in the comparison of bytes 14 and 15 to X'0602'.) If, in addition, the RPLVACS bit is on in the read-only RPL, the session is VTAM-initiated.

The application program can respond to a VTAM-initiated LU 6.2 session request with one of the following macroinstructions:

- APPCCMD CONTROL=OPRCNTL, QUALIFY=ACTSESS to accept new sessions
- APPCCMD CONTROL=OPRCNTL, QUALIFY=DACTSESS to reject new sessions

If the session is not VTAM-initiated, the application program can change the session parameters to create a non-LU 6.2 session and accept the session with a non-LU 6.2 record API macroinstruction. For more details about the LOGON exit, see “LOGON” on page 246.

Determining session type after SCIP exit

After the SCIP exit is driven, the application program checks the RPLVACS bit in the read-only RPL supplied in the parameter list to determine if the session is an LU 6.2 session. If this bit is set on, the application program should respond to the session request with one of the following macroinstructions:

- APPCCMD CONTROL=OPRCNTL, QUALIFY=ACTSESS to accept new sessions
- APPCCMD CONTROL=OPRCNTL, QUALIFY=DACTSESS to reject new sessions

For more details about the SCIP exit, see page “SCIP” on page 247.

When the application program issues the APPCCMD CONTROL=OPRCNTL, QUALIFY=ACTSESS or APPCCMD CONTROL=OPRCNTL, QUALIFY=DACTSESS macroinstruction, it must include the communication identifier (CID) of the session. The CID is found in the parameter list that VTAM supplies in register 1 when the exit is scheduled. The location of the CID is shown in Table 24 on page 123.

Table 24. Locating the Communication ID (CID)

Exit	CID Location
LOGON	Word 6 of the parameter list
SCIP	Word 2 of the parameter list

The ARG field on the APPCCMD macroinstruction is set to the CID when the macroinstruction is issued.

For details on the syntax and operands for the macroinstructions, refer to z/OS Communications Server: SNA Programmer's LU 6.2 Reference.

How to provide different session parameters

To supply different session parameters, the application program builds a BIND image and passes its address on the APPCCMD CONTROL=OPRCNTL, QUALIFY=ACTSESS macroinstruction.

Application programs without a LOGON or SCIP exit cannot alter session parameters. In this case, VTAM performs the LOGON or SCIP exit functions.

Application programs are limited in what they can specify when changing the BIND and BIND response. If the application program uses the APPCCMD CONTROL=OPRCNTL, QUALIFY=ACTSESS macroinstruction, VTAM automatically overrides anything the application program specifies for certain BIND image fields. If the application program does not plan to specify new session parameters, set the RPL's AREA field to 0. Otherwise, VTAM assumes that the contents of the field are an address for new parameters.

For more information on the parameter values of the BIND image, see “BIND image and response” on page 124.

Example of accepting a session

Suppose that an application program called APPLA has had its SCIP exit scheduled. The application program checks that the session will be an LU 6.2 session and then issues an APPCCMD macroinstruction to establish the session. This is shown in the following example:

```
         L     10,16(,1)         * GET RPL ADD. (5TH WORD OF PARM LIST)
         USING IFGRPL,10         * ESTABLISH ADDRESSABILITY
         TM    RPLCNTDC,RPLTBIND * WAS EXIT DRIVEN FOR BIND?
         BNO   NOTBIND           * GO ELSEWHERE FOR OTHER REQUESTS
         L     9,4(,1)           * GET CID (2ND WORD OF PARM LIST)
         TM    RPLCHN,RPLVACS    * IS THIS BIND FOR AN LU 6.2 SESSION?
         BNO   NONAPPC           * GO ELSEWHERE FOR OTHER SESSION TYPES
*
* EXIT WAS DRIVEN FOR 6.2 BIND - ACCEPT IT WITH APPCCMD.
*
         APPCCMD CONTROL=OPRCNTL,                                      X
               QUALIFY=ACTSESS,                                        X
               RPL=RPLA,                                               X
               AAREA=RPLAX,                                            X
               ACB=APPLA,                                              X
               ARG=(9),                                                X
               AREA=0
                  .
                  .
                  .
RPLA     RPL AM=VTAM             * RPL STORAGE
RPLAX    ISTRPL6                 * RPL EXTENSION STORAGE
APPLA    ACB AM=VTAM,MACRF=LOGON,APPLID=APPLNAME * ACB STORAGE
```

BIND image and response

Table 26 on page 125 and Table 27 on page 132 indicate the BIND fields that an application program can set and those that VTAM sets. The values for fields that the application program can set are obtained as shown in Table 25 on page 124.

Table 25. LOGON and SCIP exits and the BIND

Application Program	BIND Field Value Obtained
Has no LOGON exit or does not supply a BIND image request (session parameters)	From the corresponding field in the pending CINIT request or is set by VTAM
Has no SCIP exit or does not supply a BIND image response (session parameters)	From the corresponding field in the received BIND or is set by VTAM

The options available to an application program are listed in Table 26 on page 125 and Table 27 on page 132. These options apply *only* when VTAM sends the BIND or BIND image.

Note: The application program might provide session parameters. These parameters, if provided, do not include the BIND request code byte. If the BIND image that is received is to be mapped with the ISTDBIND DSECT, base the DSECT on byte 1 of the BIND image, not byte 0, which contains the X'31' request code.

For example, if you specify the BIND image using the RPLAREA of the APPCCMD CONTROL=OPRCNTL, QUALIFY=ACTSESS macroinstruction, the address specified in the RPLAREA should be the beginning of the ISTDBIND. It begins with the BIND format and type that follows the X'31' request code. Refer to z/OS Communications Server: SNA Programmer's LU 6.2 Reference for the ISTDBIND DSECT.

Although architecturally the PLU name in the BIND is a variable-length field, the ISTDBIND DSECT treats it as an 8-byte name. If the actual name is fewer than 8 bytes, pad it with blanks when using ISTDBIND to build the session parameters. The BIND area must match the ISTDBIND DSECT, not the BIND RU.

Byte 13 of the ISTDBIND (byte 14 of the BIND RU) is the first byte of the presentation services usage field, which continues through byte 24 of the ISTDBIND (byte 25 of the BIND RU). This field is used

to specify optional protocols or data streams that apply to the specific type of LU. However, an LU 6.2 application program has control of only byte 23 of the BIND RU.

Some fields that VTAM sets in the BIND are determined by vectors that are passed to VTAM when the application opens an ACB. These vectors are located in the application-ACB vector list and mapped by the ISTVACBV DSECT. See "Vector lists used during OPEN processing" on page 21 for more information.

For information on setting the session parameter fields, refer to z/OS Communications Server: SNA Programming.

Table 26. Bind request unit fields the application program can set

BIND RU	ISTDBIND	Use	Application's Options	VTAM's Action
Byte 0	(Not present)	Request code	Not accessible to the application program	VTAM sets to X'31'.
Byte 1	Byte 0 BINFMTY	Format and type **Bit** **Meaning** **0–3** Format **4–7** Type	VTAM ignores or overrides application program's setting.	VTAM sets to X'00'.
Byte 2	Byte 1 BINFM	FM profile	VTAM ignores or overrides application program's setting.	VTAM sets to X'13'.
Byte 3	Byte 2 BINTS	TS profile	VTAM ignores or overrides application program's setting.	VTAM sets to X'07'.
Byte 4	Byte 3 BINPRIP	FM usage, PLU **Bit** **Meaning** **0** Chaining **1** Request control **2–3** Chain response **4–6** Reserved **7** Send end bracket indicator	VTAM ignores or overrides application program's setting.	VTAM sets to X'B0'.

Table 26. Bind request unit fields the application program can set (continued)

BIND RU	ISTDBIND	Use	Application's Options	VTAM's Action
Byte 5	Byte 4 BINSECP	FM usage, SLU **Bit** **Meaning** **0** Chaining **1** Request control **2–3** Chain response **4–6** Reserved **7** Send end bracket indicator	VTAM ignores or overrides application program's setting.	VTAM sets to X'B0'.
Byte 6	Byte 5 BINCMNP	FM usage, common **Bit** **Meaning** **0** Segmenting **1** FM Header use **2** Bracket usage **3** Bracket term rule **4** Alternate code **5–6** Reserved **7** BIND RSP queue capability	VTAM ignores or overrides application program's setting.	VTAM sets to X'50'.

Table 26. Bind request unit fields the application program can set (continued)

BIND RU	ISTDBIND	Use	Application's Options	VTAM's Action
Byte 7	Byte 6 BINCMNP2	FM usage, common		
		Bit **Meaning** **0–1** Send/Receive Mode	VTAM ignores or overrides application program's setting.	VTAM sets to half-duplex flip-flop (B'10') if the application specified PARMS=(FDX=NO), or defaulted this field on the ACB. VTAM sets to B'00' Full-Duplex if the application specified PARMS=(FDX=YES) on the ACB.
		2 Symmetric responsibility for recovery	VTAM ignores or overrides application program's setting.	VTAM always sets this value to B'1', symmetric responsibility for recovery.
		3 Primary/secondary **Primary** 1=Primary is brackets first speaker and contention winner; secondary is brackets bidder and contention loser. **Secondary** 0=Secondary is brackets first speaker and contention winner; primary is brackets bidder and contention loser.	VTAM ignores or overrides application program's setting.	VTAM sets to B'1' or B'0' depending on session polarity.
		4–5 Alternate code processing identifier	VTAM ignores or overrides application program's setting.	VTAM APPC code sets to B'00'.
		6 Control vectors are included after the SLU name	VTAM ignores or overrides application program's setting.	VTAM's APPC code sets to B'0'. Subsequent processing may set to B'1'.
		7 Reset state for Half-Duplex flip-flop (for example, at start of session.) 1=Primary sends first.	VTAM ignores or overrides application program's setting.	VTAM sets to B'1'.

Table 26. Bind request unit fields the application program can set (continued)

BIND RU	ISTDBIND	Use	Application's Options	VTAM's Action
Byte 8	Byte 7 BINAPACE	SLU's send pacing window **Bit** **Meaning** **0** Staging indicator **1** Reserved **2–7** Secondary TC's send window size	VTAM ignores or overrides application programs' setting.	VTAM sets bits 0–1. Bits 2–7 are copied and used for tuning in adaptive session pacing. Adaptive session pacing is always used unless the pacing stage endpoints are not at the appropriate level.
Byte 9	Byte 8 BINRPACE	SLU's receive pacing window **Bit** **Meaning** **0** Adaptive pacing **1** Reserved **2–7** Secondary TC's receive window size	VTAM ignores or overrides application programs' setting.	VTAM sets bit 0 to B'1'. Bits 2–7 are copied and used for tuning in adaptive session pacing.
Byte 10	Byte 9 BINRUSZ	SLU's maximum RU size	Application program can set from X'80'–X'FF'.	If the application program sets maximum RU size to 0 (unspecified) or less than 256, VTAM overrides this value with a value of 4096 (X'89'). **Note:** RU sizes corresponding to values X'01' through X'7F' are not supported architecturally. The high order bit of the maximum RU size must be set to 1 for the byte to be interpreted as a supported value.
Byte 11	Byte 10 BINPRUSZ	PLU's maximum RU size	Application program can set from X'80'–X'FF'.	If the application program sets maximum RU size to 0 (unspecified) or less than 256, VTAM overrides this value with a value of 4096 (X'89'). **Note:** RU sizes corresponding to values X'01' through X'7F' are not supported architecturally. The high order bit of the maximum RU size must be set to 1 for the byte to be interpreted as a supported value.

Table 26. Bind request unit fields the application program can set (continued)

BIND RU	ISTDBIND	Use	Application's Options	VTAM's Action
Byte 12	Byte 11 BINSPACE	PLU's send pacing window **Bit** **Meaning** **0** Staging indicator **1** Reserved **2–7** Primary TC's send window size	VTAM ignores or overrides application programs' setting.	VTAM sets bits 0–1. Bits 2–7 are copied and used for tuning in adaptive session pacing.
Byte 13	Byte 12 BINBPACE	PLU's receive pacing window **Bit** **Meaning** **0–1** Reserved **2–7** Primary TC's receive window size	Application program can set bits 2-7 to X'00' - X'3F'. VTAM overrides if value greater than CINIT bind image.	If bits 2-7 are 0, VTAM defaults to a receive pacing window of 32767 (X'7FFF').
Byte 14	Byte 13 BINLUP	PS profile **Bit** **Meaning** **0** PS usage format **1–7** LU type 6	VTAM ignores or overrides application program's setting.	VTAM sets to X'06'.
Byte 15	Byte 14 BINPSCHR	PS usage **Bit** **Meaning** **0** LU-6 level **1–7** Level 2	VTAM ignores or overrides application program's setting.	VTAM sets to X'02'.
Bytes 16–21	Bytes 15–20	Reserved or retired	VTAM ignores or overrides application program's setting.	VTAM sets to X'00000000000000'.
Byte 22	Byte 21 BINDSSSP BINDESS	Distributed Systems SecurityExtended Security Sense Codes **Bit** **Meaning** **0** Support for third party DCE security services **1** Extended Security Sense Codes **2–7** Reserved	VTAM ignores or overrides application program's setting.	If the application supplies a local-application's-DCE-capability vector (ISTVAC82) and indicates support for DCE security, then VTAM sets bit 0 to B'1'. If the application supplies an application-capabilities vector (ISTVAC81) and indicates support for extended security sense codes (VAC81ESS), then VTAM sets bit 1 to B'1'.

Table 26. Bind request unit fields the application program can set (continued)

BIND RU	ISTDBIND	Use	Application's Options	VTAM's Action
Byte 23	Byte 22	**Bit** **Meaning** **0–2** Reserved **3** FMH-5 security subfields support indicator **4** Session-level verification protocol support indicator **5** Password substitution indicator **6** Already-verified support indicator **7** Persistent-verification support indicator (MVS only)	Application program can set bits 3, 6, and 7 but only on the initial session with a partner LU. After a session is established with a partner LU, these bits are ignored if the application program sets them. Bits 6 and 7 can be set only if bit 3 is also set. If the application sets bit 5, VTAM ignores or overrides this setting.	If the application program does not provide a BIND image, VTAM sets bits 3, 6, and 7 according to the value specified on the SECACPT operand of the APPL definition statement. If a security management product also provides conversation security information, the most secure level of security will be used. Other bits in the byte are set to B'0'. For the initial session with the partner, VTAM sets bit 5 if the application indicates support for password substitution in its application-capabilities vector. For subsequent sessions with this partner, VTAM sets bit 5 according to the negotiated support for password substitution.
Byte 24	Byte 23	**Bit** **Meaning** **0** Reserved	VTAM ignores or overrides application program's setting.	VTAM sets to B'0'.
		1–2 Sync point, backout support	VTAM ignores or overrides application program's setting.	If this is the first session to the partner, VTAM sets this field as specified by the SYNCLVL operand on the APPL definition statement. That is, VTAM sets the field to B'10' if SYNCLVL=SYNCPT and sets the field to B'01' if SYNCLVL=CONFIRM. If this is not the first session to the partner, VTAM sets this field as the prior sessions have set this field.
		3 Reconnect support	VTAM ignores or overrides application program's setting.	VTAM sets to B'0'.
		4–5 Session reinitiation	VTAM ignores or overrides application program's setting.	VTAM sets to B'00' for parallel-session partners or B'11' for single-session partners.
		6 Parallel- session support	VTAM ignores or overrides application program's setting.	VTAM sets the bit to B'0' (not supported) if the BIND image contained in the CINIT-specified single session.

Table 26. Bind request unit fields the application program can set (continued)

BIND RU	ISTDBIND	Use	Application's Options	VTAM's Action
		7 CNOS support	VTAM ignores or overrides application program's setting.	VTAM's APPC support ensures the CNOS support bit has the same value as the parallel-session support bit.
Byte 25	Byte 24	**Bit** **Meaning** **0** Reserved **1** Limited-resource support **2–5** Reserved **6–7** Data compression indicators	VTAM ignores or overrides application program's setting.	VTAM's APPC support sets these fields to zero. Other components of VTAM may override this setting. For example, limited resources support is set to 1 if LIMRES is specified on the LINE or PU definition statement.
Byte 26	Byte 25 BINCRCTL	Cryptography support	VTAM ignores or overrides application program's setting.	VTAM sets this based on the ENCR setting on the APPL statement.
Byte 27	Byte 26 BINPRIML	PLU name length	VTAM ignores or overrides application program's setting.	VTAM sets.
Bytes 28–m	Bytes 27–m BINPRIMN	PLU name	VTAM ignores or overrides application program's setting.	VTAM sets.
Byte m+1	Byte m+1 BINUSEL	User data length	Application can set.	VTAM can set.
Byte m+2	Byte m+2	User data key	Application program can set. If the application program plans to supply unformatted user data, the user data key must be set to X'00'.	VTAM can set.
Bytes m+3–o	Bytes m+3–o	Unformatted user data	Application program can set.	VTAM copies from BIND supplied by application program.
Bytes o+1–p	Bytes o+1–p	Mode name subfield	VTAM ignores or overrides application program's setting.	VTAM sets.
Bytes p+1–q	Bytes p+1–q	Session instance ID subfield	VTAM ignores or overrides application program's setting.	VTAM sets.
Bytes q+1–r	Bytes q+1–r	Network-qualified PLU network name	VTAM ignores or overrides application program's setting.	VTAM sets.
Bytes r+1–t	Bytes r+1–t	Random data	VTAM ignores or overrides application program's setting.	VTAM sets if session level verification is used.
Bytes t+1–u	Bytes t+1–u	URC	VTAM ignores or overrides application program's setting.	VTAM sets.

Table 26. Bind request unit fields the application program can set (continued)

BIND RU	ISTDBIND	Use	Application's Options	VTAM's Action
Bytes u+1–v	Bytes u+1–v	SLU name	VTAM ignores or overrides application program's setting.	VTAM sets.

Table 27. Bind response unit fields the application program can set

BIND RU	ISTDBIND	Use	Application's Options	VTAM's Action
Byte 0	(Not present)	Request code	Not accessible to the application program	VTAM sets to X'31'.
Byte 1	Byte 0 BINFMTY	Format and type **Bit** **Meaning** **0–3** Format **4–7** Type	VTAM ignores or overrides application program's setting.	VTAM sets to X'00'.
Byte 2	Byte 1 BINFM	FM profile	VTAM ignores or overrides application program's setting.	VTAM sets to X'13'.
Byte 3	Byte 2 BINTS	TS profile	VTAM ignores or overrides application program's setting.	VTAM sets to X'07'.
Byte 4	Byte 3 BINPRIP	FM usage, PLU **Bit** **Meaning** **0** Chaining **1** Request control **2–3** Chain response **4–6** Reserved **7** Send end bracket indicator	VTAM ignores or overrides application program's setting.	VTAM sets to X'B0'.
Byte 5	Byte 4 BINSECP	FM usage, SLU **Bit** **Meaning** **0** Chaining **1** Request control **2–3** Chain response **4–6** Reserved **7** Send end bracket indicator	VTAM ignores or overrides application program's setting.	VTAM sets to X'B0'.

Table 27. Bind response unit fields the application program can set (continued)

BIND RU	ISTDBIND	Use	Application's Options	VTAM's Action
Byte 6	Byte 5 BINCMNP	FM usage, common **Bit** **Meaning** **0** Segmenting **1** FM header use **2** Bracket usage **3** Bracket term rule **4** Alternate code **5–6** Reserved **7** BIND RSP queue capability	VTAM ignores or overrides application program's setting.	VTAM sets to X'50'.

Table 27. Bind response unit fields the application program can set (continued)

BIND RU	ISTDBIND	Use	Application's Options	VTAM's Action
Byte 7	Byte 6 BINCMNP2	FM usage, common		
		Bit **Meaning** **0–1** Send/Receive Mode	VTAM ignores or overrides application program's setting.	VTAM sets to B'00' (full-duplex) if both of the following items are true: • The BIND request contained B'00' • The application specified PARMS=(FDX=YES) on the ACB. Otherwise, VTAM sets B'10' (half-duplex) and the session is limited to half-duplex capability.
		2 Symmetric responsibility for recovery	VTAM ignores or overrides application program's setting.	VTAM always sets this value to B'1', symmetric responsibility for recovery.
		3 Primary/secondary **Primary** 1=Primary is brackets first speaker and contention winner; secondary is brackets bidder and contention loser. **Secondary** 0=Secondary is brackets first speaker and contention winner; primary is brackets bidder and contention loser.	VTAM ignores or overrides application program's setting.	VTAM sets to B'1' or B'0' depending on session polarity.
		4–5 Alternate code processing identifier	VTAM ignores or overrides application program's setting.	VTAM APPC code sets to B'00'.
		6 Control vectors are included after the SLU name	VTAM ignores or overrides application program's setting.	VTAM APPC code sets to B'0', subsequent processing may set to B'1'.
		7 Reset state for Half-Duplex flip-flop (for example, at start of session.) 1=Primary sends first.	VTAM ignores or overrides application program's setting.	VTAM sets to B'1'.

Table 27. Bind response unit fields the application program can set (continued)

BIND RU	ISTDBIND	Use	Application's Options	VTAM's Action
Byte 8	Byte 7 BINAPACE	SLU's send pacing window **Bit** **Meaning** **0** Staging indicator **1** Reserved **2–7** Secondary TC's send window size	Application program can set bit bits 2–7 to X'00'–X'3F'.	VTAM sets bits 0–1. If the pacing stage endpoint supports PU type 2.1 protocols, VTAM sets bits 2–7 to X'00'. Otherwise, VTAM copies bits 2–7 from the BIND response supplied by the application program.
Byte 9	Byte 8 BINRPACE	SLU's receive pacing window **Bit** **Meaning** **0** Adaptive pacing **1** Reserved **2–7** Secondary TC's receive window size	Application program can set bits 2–7 to X'00'–X'3F'.	If the pacing stage endpoint supports PU type 2.1 protocols, VTAM sets bit 0 to B'1' and bits 2–7 to X'00'. Otherwise, VTAM copies bits 2–7 from the BIND response supplied by the application program.
Byte 10	Byte 9 BINRUSZ	SLU's maximum RU size	Application program cannot change this to less than 256.	VTAM rejects the session if this value is less than 256. **Note:** RU sizes corresponding to values X'01' through X'7F' are not supported architecturally. The high order bit of the maximum RU size must be set to 1 for the byte to be interpreted as a supported value.
Byte 11	Byte 10 BINPRUSZ	PLU's maximum RU size	Application program cannot change this to less than 256.	VTAM rejects the session if this value is less than 256. **Note:** RU sizes corresponding to values X'01' through X'7F' are not supported architecturally. The high order bit of the maximum RU size must be set to 1 for the byte to be interpreted as a supported value.

Table 27. Bind response unit fields the application program can set (continued)

BIND RU	ISTDBIND	Use	Application's Options	VTAM's Action
Byte 12	Byte 11 BINSPACE	PLU's send pacing window **Bit** **Meaning** **0** Staging indicator **1** Reserved **2–7** Primary TC's send window size	Application program can set bits 2–7 to X'00'–X'3F'.	VTAM sets bits 0–1. If the pacing stage endpoint supports PU type 2.1 protocols, VTAM sets bits 2–7 to X'00'. Otherwise, VTAM copies bits 2–7 from the BIND response supplied by the application program.
Byte 13	Byte 12 BINBPACE	PLU's receive pacing window **Bit** **Meaning** **0–1** Reserved **2–7** Primary TC's receive window size	Application program can set bits 2–7 to X'00'–X'3F'.	VTAM sets bits 0–1. If the pacing stage endpoint supports PU type 2.1 protocols, VTAM sets bits 2–7 to X'00'. Otherwise, VTAM copies bits 2–7 from the BIND response supplied by the application program.
Byte 14	Byte 13 BINLUP	PS profile **Bit** **Meaning** **0** PS usage format **1–7** LU type 6	VTAM ignores or overrides application program's setting.	VTAM sets to X'06'.
Byte 15	Byte 14 BINPSCHR	PS usage **Bit** **Meaning** **0** LU-6 level **1–7** Level 2	VTAM ignores or overrides application program's setting.	VTAM sets to X'02'.
Bytes 16–21	Bytes 15–20	Reserved or retired	VTAM ignores or overrides application program's setting.	VTAM sets to X'00000000000000'.
Byte 22	Byte 21 BINDSSSP BINDESS	Distributed Systems SecurityExtended Security Sense Codes **Bit** **Meaning** **0** Support for third party DCE security services **1** Extended Security Sense Codes **2–7** Reserved	VTAM ignores or overrides application program's setting.	If the application supplies a local-application's-DCE-capability vector (ISTVAC82) and indicates support for DCE security, then VTAM sets bit 0 to B'1'. If the application supplies an application-capabilities vector (ISTVAC81) and indicates support for extended security sense codes (VAC81ESS), then VTAM sets bit 1 to B'1'.

Table 27. Bind response unit fields the application program can set (continued)

BIND RU	ISTDBIND	Use	Application's Options	VTAM's Action
Byte 23	Byte 22	**Bit** **Meaning** **0–2** Reserved **3** FMH-5 security subfields support indicator **4** Session-level verification protocol support indicator **5** Password substitution indicator **6** Already-verified support indicator **7** Persistent-verification support indicator (MVS only)	Application program can set bits 3, 6, and 7 but only on the initial session with a partner LU. After a session is established with a partner LU, these bits are ignored if the application program sets them. Bits 6 and 7 can be set only if bit 3 is also set. If the application sets bit 5, VTAM ignores or overrides this setting.	If the application program does not provide a BIND image, VTAM sets bits 3, 6, and 7 according to the value specified on the SECACPT operand of the APPL definition statement. If a security management product also provides conversation security information, the most secure level of security will be used. For the initial session with the partner, VTAM sets bit 5 if both the partner LU and the local application indicate support for password substitution. For subsequent sessions with this partner, VTAM sets bit 5 according to the negotiated support for password substitution. Other bits in the byte are set to B'0'.
Byte 24	Byte 23	**Bit** **Meaning** **0** Reserved	VTAM ignores or overrides application program's setting.	VTAM sets to B'0'.

Table 27. Bind response unit fields the application program can set (continued)

BIND RU	ISTDBIND	Use	Application's Options	VTAM's Action
		1–2 Sync point, backout support	VTAM ignores or overrides application program's setting.	If this is the first session to the partner, VTAM negotiates this value as follows: • If the value received on the BIND is B'01' (CONFIRM), then VTAM echoes this value in the response. • If the value received on the BIND is B'10' (SYNCPT) and SYNCLVL=SYNCPT on the APPL statement, then VTAM echoes B'10' on the response. • If the value received on the BIND is B'10' (SYNCPT) and SYNCLVL=CONFIRM on the APPL statement, then VTAM negotiates the response to B'01' (CONFIRM). If this is not the first session to the partner, VTAM sets this field as the prior sessions have set the field.
		3 Reconnect support	VTAM ignores or overrides application program's setting.	VTAM sets to B'0'.
		4–5 Session reinitiation	VTAM ignores or overrides application program's setting.	VTAM sets to B'00' for parallel-session partners. For single-session partners, VTAM copies the value from the BIND request.
		6 Parallel- session support	Application program can set.	VTAM overrides the application program's setting and sets the bit to B'0' (not supported) if the BIND image specified single session.
		7 CNOS support	Application program can set.	VTAM ensures that the CNOS support bit has the same value as the parallel-session support bit.

Table 27. Bind response unit fields the application program can set (continued)

BIND RU	ISTDBIND	Use	Application's Options	VTAM's Action
Byte 25	Byte 24	**Bit** **Meaning** **0** Reserved **1** Limited-resource support **2–5** Reserved **6–7** Data compression indicators	VTAM ignores or overrides application program's setting.	VTAM's APPC support echoes these fields. Other components of VTAM may modify these bits as appropriate. For example, Limited Resource will be set to 1 if LIMRES is specified on the LINE or PU macroinstruction.
Byte 26	Byte 25 BINCRCTL	Cryptography support	VTAM ignores or overrides application program's setting.	VTAM sets based on the setting of the ENCR keyword as specified, for example, on the APPL statement and other appropriate macroinstructions.
Byte 27	Byte 26 BINPRIML	PLU name length	VTAM ignores or overrides application program's setting.	Reserved on response. VTAM sets to X'00'.
Bytes 28–m	Bytes 27–m BINPRIMN	PLU name	VTAM ignores or overrides application program's setting.	Reserved on response. VTAM copies from BIND request.
Byte m+1	Byte m+1 BINUSEL	User data length	Application program can set.	VTAM can set.
Byte m+2	Byte m+2	User data key	Application can set. If the application program wishes to supply unformatted user data, the user data key must be set to X'00'.	VTAM can set.
Bytes m+3–o	Bytes m+3–o	Unformatted user data	Application program can set.	VTAM copies from BIND supplied by application program.
Bytes o+1–p	Bytes o+1–p	Mode name subfield	VTAM ignores or overrides application program's setting.	VTAM copies from BIND request.
Bytes p+1–q	Bytes p+1–q	Session instance ID subfield	VTAM ignores or overrides application program's setting.	VTAM copies from BIND request.
Bytes s+1–t	Bytes s+1–t	Network-qualified SLU network name	VTAM ignores or overrides application program's setting.	VTAM sets.
Bytes q+1–r	Bytes q+1–r	Random data X'11'	VTAM ignores or overrides application program's setting.	VTAM sets as appropriate.
Bytes r+1–s	Bytes r+1–s	Enciphered random data X'12'	VTAM ignores or overrides application program's setting. End of userdata subfields.	VTAM sets as appropriate.

Table 27. Bind response unit fields the application program can set (continued)

BIND RU	ISTDBIND	Use	Application's Options	VTAM's Action
Bytes t+1–u	Bytes t+1–u	URC	VTAM ignores or overrides application program's setting.	VTAM does not set.
Bytes u+1–v				
Bytes u+1–v	SLU name	VTAM ignores or overrides application program's setting.	VTAM does not set. Reserved upon response.	

User data structured subfields

The user data structured subfields are in the BIND, but not at fixed locations. (These fields are not always at the same offset in the BIND because of the variable-length fields.) They must be in the correct order. VTAM uses these architectural user data structured subfields during BIND processing as follows:

1. Unformatted user data

   Unformatted user data subfield X'00' is included, if provided by the user.

2. Mode name

   Mode name structured data subfield X'02' is included in all LU 6.2 BIND requests and responses sent by VTAM.

3. Session instance identifier

   All LU 6.2 BIND requests and responses sent by APPC VTAM carry the session instance identifier structured data subfield X'03'. The identifier is unique among sessions between two LUs.

4. Network-qualified network name

   Network-qualified PLU network name structured data subfield X'04' is included in all LU 6.2 BIND requests sent by VTAM.

   Network-qualified SLU network name structured data subfield X'05' is included in all LU 6.2 BIND responses sent by VTAM.

5. Random data

   Random data structured subfield X'11' is sent on BIND and BIND responses when the session is being established with session-level LU-LU session verification.

6. Security reply data

   Enciphered random data structured subfield X'12' is used on the BIND response when the session is being established with session-level LU-LU session verification.

7. Nonce data

   Nonce data is sent on the BIND in X'13' when password substitution support is indicated on the application-capabilities vector.

8. Security mechanisms

   Security policy information is sent on the BIND in X'14' when support for Generic Security Services (DCE security services) are requested on the application-capabilities vector.

LU 6.2 names used for session activation

In LU 6.2 architecture, an LU learns its partner's name through the user data field of the BIND. In Figure 22 on page 141, the PLU causes the BIND request, and its subsequent response, to flow by issuing either an APPCCMD CONTROL=ALLOC|PREALLOC or an APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS (**1**). These macroinstructions require the name of the target to be specified on the LUNAME parameter. For network-qualified names, the NETID parameter must also be specified. The PLU indicates its own name in subfield X'04' of the user data field of the BIND request unit (**2**). The SLU returns its own name in subfield

X'05' of the user data field of the BIND response unit (**3**). These are the names most often used by the LUs during operation.

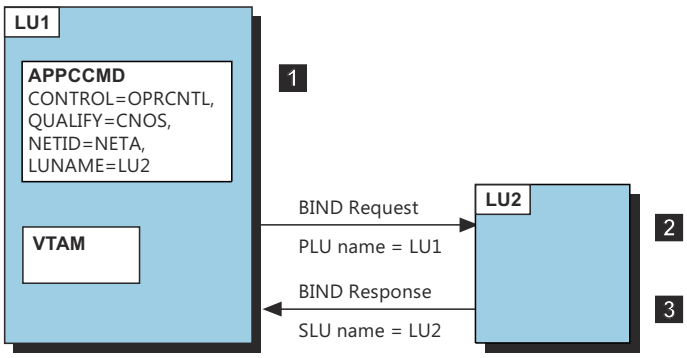


Figure 22. Exchange of LU names during session activation

In most cases, the name specified as the target (LUNAME) on the APPCCMD macroinstruction is identical to the name returned on the BIND response unit from the SLU. When these names are not identical, VTAM tracks both names in the LU-mode table. VTAM may report these name mismatches to the application on the name change vector and sometimes to the partner LU using a sense code on the session.

How can the LU names differ?

There are several circumstances that may cause the name returned in the SLU's BIND response to differ from the name specified on the PLU's BIND request. For example, in VTAM, the partner LU may be known by a generic resource name or a USERVAR. The following scenarios explain how name translations are managed by VTAM's name mismatch detection for LU 6.2 applications.

Partner LU known by multiple names

In VTAM, for example, when the LUALIAS keyword is specified on a CDRSC definition statement, the name of the target LU may be altered.

For example, if LUALIAS=LUX is specified on the CDRSC definition statement for an LU whose real name is NETA.LU2, then when an APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS is issued specifying LUNAME=LUX, the BIND response will contain NETA.LU2 in the user data field. In this case, VTAM maintains both LU names in the LU-mode table. The name specified on the APPCCMD (LUX) becomes a SUPPLIED_NAME entry and the name returned on the BIND response unit (NETA.LU2) becomes a VARIANT_NAME entry. In this example, VTAM does not allow the local application to start any sessions using the name NETA.LU2. This prevents possible errors that could occur when an LU attempts CNOS negotiations with what appears to be different partners.

Name changes while sessions are active

After a session is established, VTAM saves the partner name specified on the LUNAME parameter and the name that was returned by the partner in the BIND response unit. These names are not required to be identical. However, if a second session request is sent to the same partner (as is shown in Figure 23 on page 142), it is possible that the partner returns a name in the BIND response unit that is not identical to the name returned on the first session.

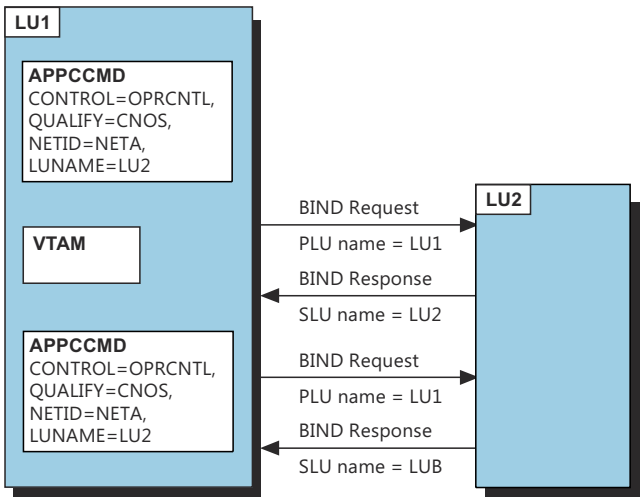


Figure 23. Name mismatch in two BIND responses for the same partner LU

Two possible explanations are that the partner LU changed its name between sessions or the second session was directed to a different LU. In either case, VTAM terminates the second session and returns an error code with an RCPRI type of X'00B0' to the APPCCMD macroinstruction.

Partner LU initiates the session

In Figure 24 on page 142, assume that VTAM is the target of a partner LU's CNOS request.

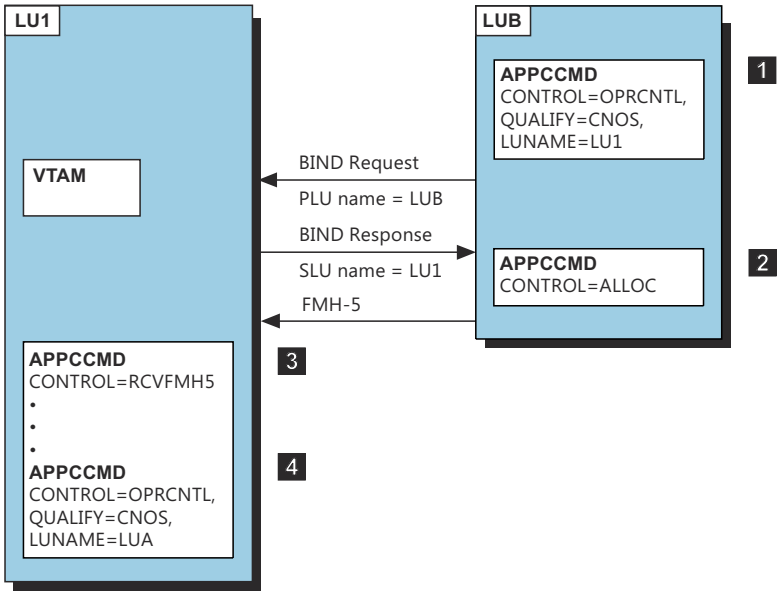


Figure 24. Name mismatch when the partner LU is the PLU

1

A remote LU issues an APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS and specifies the local application (LU1) as the SLU. After a successful CNOS negotiation, VTAM reports the results, including the partner's name (as specified on the BIND request unit) and the LOGMODE used to the local application by scheduling the ATTN(CNOS) exit. In this case, the name reported to the local application is LUB. VTAM enters LUB as a RCVD_NAME entry in the LU-mode table.

2

The partner LU then sends an FMH-5 to initiate a conversation. VTAM reports these results, again including the partner's name (LUB) and the LOGMODE used to the local application by scheduling the ATTN(FMH5) exit.

3

The local application then issues an APPCCMD CONTROL=RCVFMH5, which completes the initiation of the conversation.

Note that in the preceding steps, the local application has not specified the partner's LU name (LUB) on any APPCCMD macroinstruction.

4

The local application then issues an APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS containing LUNAME=LUA. If there exists an LUALIAS definition that converts LUA to LUB, then the following conditions occur:

- The APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS completes with an RCPRI, RCSEC combination of X'0000', X'000C' or X'0000', X'000D', which indicates a name change.
- VTAM changes the record of LUB in the LU-mode table to a VARIANT_NAME entry. LUA becomes a SUPPLIED_NAME entry for the partner LU. All subsequent APPCCMD macroinstructions that specify LUB will be rejected with an RCPRI, RCSEC combination of X'00B0', X'0001', NAME_RESOLUTION_ERROR— LUNAME_FOUND_IN_A_VARIANT_NAME_ENTRY.
- If the application requested APPCCMD vector information, a name-change vector is returned that reports the "passive" name, LUB, and the "active" name, LUA. The local application should examine the name-change vector to obtain the name that was returned.
- All subsequent reporting of actions formerly attributed to LUB, such as ATTN exit scheduling, will now be attributed to LUA.

For information about the types of LU entries, see “LU entries in the LU-mode table” on page 94.

LU 6.2 in an extended recovery facility (XRF) environment

You can use the USERVAR facility for LU 6.2 application programs in a recovery environment. The recovery environment can be XRF or another type of recovery environment.

You can run a backup application program to which a failing application program can be switched. When the application program establishes the session, VTAM substitutes a generic application name for the specific name of the XRF application program in the BIND or BIND response user data field. The partner LU sees the generic application name regardless of which specific application program is called.

You can identify the USERVAR that VTAM is to use for a specific application program. This is done by specifying APPC=YES on the MODIFY USERVAR command before VTAM negotiates the BIND. For more information about this command, refer to z/OS Communications Server: SNA Operation.

This function has restrictions that apply to LU 6.2 communications managed by VTAM because such communications customarily depend on VTAM being aware of the identity of the actual partner LU. These restrictions follow:

- Only one USERVAR ID can be assigned to an application program.
- Only one application program can be the primary provider of a service requested with a USERVAR ID.
- The MODIFY USERVAR command must be performed on the host system of the application program supporting the services requested with the USERVAR ID.
- Once a USERVAR ID is assigned to an active application program, VTAM maintains the understanding of this relationship until the application program closes its ACB.
- While a USERVAR ID is assigned to an active application program, the BIND request and response RUs contain the USERVAR ID in the appropriate User Data subfield names.
- Once you assign a USERVAR ID to an application program, any LUs trying to establish a session with the application program must use that USERVAR.

LU 6.2 applications as generic resources

You can define a group of LU 6.2 applications that provide the same function to be known and accessed by a single generic name. LUs can initiate sessions using the generic name; VTAM determines which

application program is used to establish the session. The LU knows the application program by its generic name only. This function enables VTAM to provide workload balancing by distributing incoming session initiations among a number of identical application programs, instead of overwhelming one application program.

Applications establish an association with a generic resource name after OPEN ACB and before SETLOGON OPTCD=START by issuing SETLOGON OPTCD=GNAMEADD. This association allows the application to be known by its generic resource name. An LU 6.2 application can request VTAM to use its generic name or its application network name for a particular session with an LU by specifying NAMEUSE=GNAME|APNAME on any APPCCMD macroinstruction that causes a session to be started. The application program can terminate this association with the generic resource name by issuing SETLOGON OPTCD=GNAMEDEL.

VTAM keeps track of the identity of the LU and the identity of the application that is acting as the generic resource for this session. This ensures that all subsequent session initiations from an LU that is in session with a generic resource resolve to the same application program. In this regard, the LU is said to have an "affinity" with the application.

For more information about this function, refer to z/OS Communications Server: SNA Network Implementation Guide or z/OS Communications Server: SNA Programming.

Ownership of the affinity between an LU and a generic resource member

When VTAM establishes sessions between application programs and LUs, VTAM keeps track of the LUs that are currently in session with a generic resource application. VTAM can distinguish which applications are acting as generic resources, and is aware of the affinity that is created between an application and any LU that has established a session with it. Each time the LU initiates a session using the generic resource name, VTAM establishes the session with the same generic resource application until the affinity is terminated.

The affinity between an LU and an application program is controlled by either VTAM or the application program; the controlling party *owns* the affinity. LU 6.2 application programs control this affinity if the session uses sync point services, limited resource support, or if the application specifies OPTCD=GNAMEADD on the SETLOGON macroinstruction with AFFIN=APPL in the NIB. An LU 6.2 application program can also override the ownership on a specific LU basis using an optional APPCCMD macroinstruction keyword, LUAFFIN=APPL|NOTAPPL. This optional keyword applies to the ALLOC, PREALLOC, and OPRCNTL CNOS APPCCMD macroinstructions. (Refer to those macroinstructions for additional details.) When the affinity is owned by the application program, the CHANGE macroinstruction must be used to cause VTAM to *terminate* the affinity. The LU 6.2 application can terminate this affinity between the LU and an application that is a member of a generic resource. When VTAM owns the affinity, it will be terminated when the session is ended, in most cases.

For more information about the CHANGE macroinstruction, refer to z/OS Communications Server: SNA Programming.

CNOS general data stream (GDS) variable

The GDS is architected to provide a common understanding between transaction programs (TPs) using LU 6.2 protocols. GDS variables standardize the data format so that both sending and receiving TPs can process the data.

The CNOS GDS variable is a mapping of what VTAM builds and sends on the CNOS conversation. The description of the CNOS GDS is shown in Table 28 on page 144.

Table 28. CNOS (X'1210') GDS variable

Byte	Bit	Content
0—1		Length (17 or n—1), in binary, of CNOS GDS variable, including this Length field

Table 28. CNOS (X'1210') GDS variable (continued)

Byte	Bit	Content
	GDS ID: X'1210'	
4		Service flag:
	0—3	Reserved
	4—7	Request/reply indicator: **0010** request **1000** reply, function completed abnormal **1010** reply, function accepted but not yet completed
5		Reply modifier (reserved if byte 4, bits 4—7 = 0010): **X'00'** normal—no negotiation performed **X'01'** abnormal—command race deleted **X'02'** abnormal—mode name not recognized **X'03'** reserved **X'04'** normal—negotiated reply **X'05'** abnormal—(LU-mode) session limit is 0
6		Action: **X'00'** set, (LU-mode) session limits **X'01'** reserved **X'02'** close
7		Drain immediacy:
	0—2	Reserved
	3	Source LU drain (reserved if byte 6 is not equal to 02): **0** no (send BIS at next opportunity) **1** yes
	4—6	Reserved

Table 28. CNOS (X'1210') GDS variable (continued)

Byte	Bit	Content
	7	Target LU drain (reserved if byte 6 is not equal to 02): **0** no (send BIS at next opportunity) **1** yes
8		Action flags:
	0—6	Reserved
	7	Session deactivation responsibility: **0** sender of CNOS request (source LU) **1** receiver of CNOS request (target LU)
9—10		(LU-mode) session limit:
	0	Reserved
	1—15	Maximum (LU-mode) session count, in binary
11—12		Source LU contention winners:
	0	Reserved
	1—15	Guaranteed minimum number of contention-winner sessions at source LU, in binary
13—14		Target LU contention winners:
	0	Reserved
	1—15	Guaranteed minimum number of contention-winner sessions at target LU, in binary
15		Mode name selection:
	0—6	Reserved
	7	Mode names affected by this command: **0** A single mode name is affected. **1** All mode names are affected (valid if byte 6 = 02).
16		Length (values 0 to 8 are valid; reserved if byte 15, bit 7 = 1), in binary, of mode name
	Mode name (omitted if byte 16 = X'00')	

Note: Bytes 9—14 are reserved if byte 6 is not equal to 0.

Retrieving information for a mode and sessions to be restored

When a mode is being restored, VTAM, if requested, returns information in the RESTORE (ISTSREST) control block and associated data areas. This control block contains the partner LU name, the mode name, and CNOS and session information.

A VTAM application program can use persistent LU-LU session support to facilitate recovery after a failure or to manage a planned takeover. After the recovery of the application program, the sessions and modes must be restored to permit continued use. The identifying information for the modes and sessions can be returned in the RESTORE control block.

The application program specifies the level of information that is to be returned in the RESTORE control block by using the LIST keyword in the APPCCMD CONTROL=OPRCNTL, QUALIFY=RESTORE macroinstruction. The application program can specify LU-mode table information, LU-mode table and session information, or no information. The application program also provides the area where the RESTORE control block is built, and the RPLAREA field points to that area. The length is set in the AREALEN field in the RPL.

Note: If the application program can have a network-qualified name, the length of the AREALEN field must be great enough to include the SRENETID field, which is 8 bytes in length. If NQNAMES=YES is specified, VTAM supplies the SRENETID. Therefore, provide space for it.

This control block points to the DEFINE/DISPLAY control block (ISTSLD). The layout of the control block is shown in Table 29 on page 147. For a detailed layout, refer to the ISTSLD and ISTSREST DSECTs in the z/OS Communications Server: SNA Programmer's LU 6.2 Reference.

Table 29. Layout of the RESTORE control block (ISTSREST)

Byte (Hex)	Bits	DSECT Name	Indicates
0–7	—	SRENAME	Name of the logical unit (LU name)
8–F	—	SREMODE	Name of the mode for the logical unit (LU mode)
10–13	—	SRENXTAD	Next RESTORE structure address (0 if not present)
14–17	—	SRESLDAD	SLD structure address
18–1B	—	SRESESAD	Address of first SRESESS (0 if not present)
1C–1D	—	SREMFLGS	Mode level flags
1C	0	SREMDRS	Whether the mode has been restored
1E–1F	—	SRESESCT	The number of SRESESS structures
20–27	—	SRENETID	Network ID of the logical unit

The offsets for the data being retrieved for the first session are shown in Table 30 on page 147. This structure is repeated for each session that is restored. If more than one mode is processed, a RESTORE control block for each mode follows the first one. See Appendix D, “Example of retrieving information for a mode and any restored sessions,” on page 329 for an example program for retrieving information from the RESTORE control block.

Table 30. Layout of the session information for restored sessions pending recovery (SRESESS)

Byte (Hex)	Bits	DSECT Name	Indicates
0–3	—	SRESNXTA	Pointer to next session structure (0 if not present)
4–6	—	SRESFLGS	Session-level flags

Table 30. Layout of the session information for restored sessions pending recovery (SRESESS) (continued)

Byte (Hex)	Bits	DSECT Name	Indicates
4	0	SREPCONV	Whether the conversation is pending deallocation for persistent LU-LU sessions
4	1	SRESPNDA	Whether the session is pending deactivation for persistent LU-LU sessions
7	—	SRESIDL	Length of the session instance identifier
8–F	—	SRESESID	Session instance identifier

Descriptions of the fields in the control block follow:

- **SRENAME** specifies the name of the logical unit (LU name).
- **SREMODE** specifies the mode of the logical unit (LU-mode).
- **SRENXTAD** specifies the address of the RESTORE structure for the next LU-mode. (Zero if none is present.)
- **SRESLDAD** specifies the address of the SLD structure for this LU-mode.
- **SRESESAD** specifies the address of the first SRESESS for this LU-mode. (Zero if none is present.)
- **SREMFLGS** specifies the mode level flags.
- **SREMDRS** specifies whether the mode has been restored.
  - **B'0' (NO)** specifies that the mode has not been restored.
  - **B'1' (YES)** specifies that the mode has been restored.
- **SRESESCT** specifies the number of SRESESS structures for this mode.
- **SRENETID** specifies the network ID of the logical unit.
- **SRESNXTA** points to the next session structure information for this mode. (Zero if none is present.)
- **SRESFLGS** specifies the session-level flags.
- **SREPCONV** specifies whether the conversation is pending for deallocation for persistent LU-LU sessions.
  - **B'0' (NO)** specifies that the conversation is not pending for deallocation of persistent LU-LU sessions.
  - **B'1' (YES)** specifies that the conversation is pending for deallocation of persistent LU-LU sessions.
- **SRESPNDA** specifies whether the session is pending deactivation for persistent LU-LU sessions.
  - **B'0' (NO)** specifies that the session is not pending deactivation for persistent LU-LU sessions.
  - **B'1' (YES)** specifies that the session is pending deactivation for persistent LU-LU sessions.
- **SRESIDL** specifies the length of the session instance identifier.
- **SRESESID** specifies the session instance identifier.

Chapter 7. Allocating a conversation

About this chapter

Application programs start conversations with a partner transaction program by building an FMH-5 and then using the APPCCMD CONTROL=ALLOC macroinstruction. This macroinstruction assigns a half-duplex or full-duplex-capable session to the conversation and specifies whether the conversation requested is full-duplex or half-duplex.

If the session is full-duplex capable and the conversation is full-duplex, the transaction program initiating the conversation is initially placed in SEND/RECEIVE state and the FMH-5 is sent to the partner LU immediately. The partner transaction program receiving the FMH-5 is also placed in SEND/RECEIVE state when it receives the FMH-5.

For half-duplex conversations, the transaction program initiating the conversation is initially placed in SEND state and the FMH-5 is buffered to send to the partner LU at a later time if the underlying session is half-duplex. If the underlying session is full-duplex, the FMH-5 is sent immediately. When the partner transaction program issues the APPCCMD CONTROL=RCVFMH5 macroinstruction, it is placed in RECEIVE state and receives the FMH-5.

Optionally, the application can issue an APPCCMD CONTROL=PREALLOC macroinstruction to reserve a session for a conversation without sending the FMH-5. This allows the application to obtain information about the conversation and the session it will use before issuing the APPCCMD CONTROL=SENDFMH5 macroinstruction and completing conversation allocation.

In general, the discussion of conversation allocation can be split between transaction programs allocating the conversation and transaction programs receiving the FMH-5. In both cases, however, you must understand the format of the FMH-5 and understand when the FMH-5 is sent to the partner LU.

Initiating a conversation

Depending on the session limits in effect and the type of allocation required, application programs may need to issue an APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS macroinstruction to ensure that a session is available for the conversation. If the session limits between the two LUs are 0, or no CNOS exchange has been processed on the requested mode name, a CNOS exchange must be performed to initialize or raise the session limits before the conversation can be allocated.

If an allocate request specifies QUALIFY=IMMED, a free contention-winner session must be available. If a session is not available, the application program must raise the local contention-winner session limit, if possible, before the allocation request can succeed. For more information on setting and negotiating session limits, see Chapter 6, “Managing sessions,” on page 83.

If the application program is not tracking session limits on its own, it can use the APPCCMD CONTROL=OPRCNTL, QUALIFY=DISPLAY macroinstruction to check on the session limits. If session limits have not been initialized, the DISPLAY request fails with a return code indicating that the partner LU name or the requested mode name could not be found.

If the application program attempts to allocate a conversation before session limits are initialized or when they are equal to 0, a return code of ALLOCATION_FAILURE_NO_RETRY is returned. The application program must correct the session limits before trying the allocation again.

As session limits are initialized or changed, new sessions might be started. This will result in the application's LOGON and SCIP exits being scheduled, if they are present.

After the session limits are established, an application program initiates a conversation by:

- Building an FMH-5

- Issuing the APPCCMD CONTROL=ALLOC macroinstruction (or the APPCCMD CONTROL=PREALLOC macroinstruction followed by the APPCCMD CONTROL=SENDFMH5 macroinstruction).

Building an FMH-5

The FMH-5 is a control block defined by LU 6.2 architecture. It contains control information used by LUs when establishing a conversation. It is sent from an LU to a partner LU as part of the process of establishing a conversation.

To build the FMH-5, the application program:

- Obtains storage for the FMH-5, either at assembly time or dynamically.
- Initializes the fields using the ISTFM5 DSECT and the DSECTs contained within the ISTFM5 DSECT.

The application program builds the FMH-5 in its storage and initializes it. When the application program specifies a conversation allocation macroinstruction, the RPL for the request contains a pointer to the FMH-5. For conversations allocated on full-duplex-capable sessions, the VTAM program copies the control block and sends it to the partner LU immediately; otherwise, VTAM copies the control block and buffers it until enough data is accumulated to send to the partner LU or until the application flushes the SEND buffer. The partner LU then examines the control block as part of the process of establishing its end of the conversation.

The FMH-5 specifies the name of the transaction program with which the application program is to have a conversation. For example, if the partner LU has a subroutine to store data in a database, the local application program might specify a transaction program name of DATA_BASE_MANAGER. The partner LU could then check the transaction program name when it receives the FMH-5 and schedule the subroutine. The exact manner in which the transaction program name is mapped to a processing thread in an application program is determined by the application program.

The FMH-5 also provides other information pertaining to the conversation, as well as optional program initialization parameters (PIP) data. PIP data is any information the application program sends to the transaction program. The FMH-5 includes:

- An indication of whether optional PIP data is included with the FMH-5
- Whether the conversation is basic or mapped
- Whether a token is used to protect a password
- Whether DCE security services are supported for the conversation
- Synchronization level of the conversation
- Whether the conversation is full-duplex or half-duplex
- Optional security information
- Optional information identifying the transaction being carried out

The layout of an FMH-5 and a description of each field is provided in “FMH-5 fields” on page 151.

If you are using dynamic storage allocation, a useful technique is to create templates of the FMH-5 and the subfields of the FMH-5 and copy them onto the storage that is obtained dynamically as needed.

Additional FMH-5 DSECTs

The IBM-supplied DSECT ISTFM5 enables you to refer to the fields in the FMH-5 symbolically. ISTFM5 also contains several other DSECTs that map storage in optional subfields that can be part of the FMH-5. (For an assembler listing of the ISTFM5 DSECT, refer to z/OS Communications Server: SNA Programmer's LU 6.2 Reference.)

The additional FMH-5 DSECTs and the subfields they map are:

FM5ASI
Access security subfields. Each of these subfields, in turn, is mapped by the DSECT FM5ACCSE.

FM5LUOW1
Logical unit of work identifier field.

FM5LUOW2
Actual logical unit of work field.

FM5CVCOR
Conversation correlator.

FM5ACCSE
Individual access security subfields.

FM5PIPFM
PIP format.

FM5PIPSM
Actual PIP subfield.

FMH-5 fields

The maximum length of an FMH-5 is 255 bytes; the minimum length is decimal 11.

The fields of the FMH-5 are explained in the rest of this section. For an example of an FMH-5, see "Example of an FMH-5" on page 155.

Byte
Meaning

0
Length of the FMH-5 (including this length byte, but not including any GDS data).

1
FMH type. It should be set to X'05'.

2–3
ATTACH command code of X'02FF'.

4
A flag byte. Several of the bits in this byte are reserved and should be set to 0. The application program can set the following bits:

- Bit 0 is the already-verified indicator. It is set when the user identifier access security subfield has been verified on a previous FMH-5.
- Bit 1 indicates that the LU is already in the signed-on list.
- Bit 2 indicates that the LU is to be added to the signed-on list.
- Bit 3 indicates that a token may be used in place of the password in the access security subfield.
- Bit 4 indicates whether PIP data is present in the FMH-5.
- Bit 5 indicates whether an authentication token (DCE security) GDS is present. If this bit is set to B'1', then bits 0, 1, 2, and 3 in byte 4 must all be 0.

5
Length, in binary, of the fixed length parameters field of the FMH-5. This field is 3 bytes long, so code this byte as X'03'.

6
Type of conversation. Code X'D0' for a basic half-duplex conversation or X'D1' for a mapped half-duplex conversation. Code X'D2' for a basic full-duplex conversation or X'D3' for a mapped full-duplex conversation. (VTAM does not offer support for mapped conversations. VTAM's support is for basic conversations. If the application program needs to use mapped conversations, it must include code to implement them. Refer to the *SNA Transaction Programmer's Reference Manual for LU Type 6.2* for information on mapped conversations.

7
Reserved. Set to X'00'.

8
Flags for fixed length parameters. The first 2 bits indicate the requested synchronization level of the conversation. The other bits are reserved and should be set to 0. The possible combinations of synchronization levels supported by VTAM are:

- X'00'—a synchronization level of *none* . Application programs cannot use confirmation requests and responses on such a conversation. The application program must enforce this restriction.
- X'40'—a synchronization level of *confirm*. Application programs can use confirmation requests and responses but not sync point requests and responses.
- X'80'—a synchronization level of *syncpt*. Application programs can use both confirmation and sync point requests and responses on this conversation.

Note: The value set in this field must be consistent with the sync-point level negotiated between the two LUs. The negotiated sync-point level is usually CONFIRM, unless both LUs support sync point. When the first session between the two LUs is established, the sync-point level (SYNCLVL) is negotiated. The APPCCMD CONTROL=OPRCNTL, QUALIFY=DISPLAY macroinstruction displays the SYNCLVL.

9
The length of the transaction program name (a hexadecimal value).

10–n
The transaction program name, up to 64 bytes long. (The LUs that are communicating determine the format.)

n+1
Length in binary of access security information. Code X'00' if no subfields are present, but subsequent fields follow.

n+2–m
Access security information. See “Access security subfields” on page 152.

m+1
Length in binary of the logical unit of work (LUW) identifier field. The LUW field cannot be used to support the sync point option set, but application programs can use it for accounting purposes. Code X'00' if no LUW is present, but subsequent fields follow.

m+2–k
The optional LUW field. See “LUW field” on page 153.

k+1
Length of conversation correlator of sender. Code X'00' if the correlator is not present, but subsequent fields follow.

k+2–w
Conversation correlator of sender. The conversation correlator is used to further qualify the LUW. Do not include it if the LUW is not present. (The description is specific to the application program.)

The following information describes the placement of the fields for GDS data.

w+1–y
Optional PIP data. See “PIP data field” on page 153.

y+1–z
Optional authentication token data (X'12F6') GDS variable. See “DCE security (authentication token) GDS field” on page 154.

Access security subfields

The application program can include optional access security subfields. If security subfields are included:

- If a user ID field is present, a password field must be present, or the already-verified indicator (byte 4, bit 0) or the persistent verification signed-on indicator (byte 4, bit 1) must be set.
- If a profile subfield is present, a user identifier subfield also must be present.
- If a password subfield is present, a user identifier subfield also must be present.

The layout of an access security subfield follows:

Byte
Meaning

0
Length in binary of the rest of the subfield. The value does *not* include this length byte.

1
The subfield type. The following values are allowed:

X'00'
Profile

X'01'
Password

X'02'
User identifier

2–i
Data, such as the actual password or identifier.

LUW field

The LUW field allows the application program to identify each conversation for accounting and sync point purposes. The layout of the LUW field follows:

Byte
Meaning

0
Length of the LUW field (not including this length byte)

1
Length of the network-qualified LU network name

2–n
Network-qualified LU network name

n+1–n+6
LUW instance number

n+7–n+8
LUW sequence number

PIP data field

The optional PIP data can be anything that the application program includes as parameter data when setting up the conversation. The layout of the PIP variable follows:

Byte
Meaning

0–1
Length in binary of the PIP variable, including the length bytes

2–3
General data stream (GDS) identifier (X'12F5')

4–n
Zero or more PIP subfields, each of which has the following format:

Byte
Meaning

0–1
Length in binary of the subfield, including the length bytes

2–3
GDS ID (X'12E2')

4–m
PIP subfield data

DCE security (authentication token) GDS field

The optional authentication token data (X'12F6') GDS variable is used to convey authentication tokens on a conversation before user data. The layout of the authentication token is as follows:

Byte
Meaning

0–1
Length (p+1), in binary, of authentication token data GDS variable, including this length field

2–3
GDS ID: X'12F6'

4–5
Header Length: length of authentication header. Valid values are 0 to n-5.

6–*n*
SNA specific header

6
Header byte bit 0 has the following meaning:

B'0'
Token exchanges are to continue using the conversation's session.

B'1'
Additional token exchanges for this conversation are to be performed using the distributed authentication service TP. If on, the associated FMH5 must contain a valid conversation correlator.

7
Length of security mechanism object identifier. Valid values are 0 to 32.

8—*m*
BER encoded form of the security mechanism's object identifier. Only required in initial flow from Attach sender. If omitted, associated length contains 0.

The rest of the SNA specific header is reserved.

***n*+1–*n*+2**
Length of the generic security services (GSS) API authentication token. Valid values are 0 – 24 576. A length of 0 is referred to as a *null token*.

n*+3–*p
A string of bytes containing the GSS-API authentication token.

Checking errors in the FMH-5

VTAM performs relatively little error checking on the FMH-5. More checks are performed when VTAM receives an FMH-5 for an application program than when an application program gives an FMH-5 to VTAM for processing. In both cases, VTAM checks the length of the FMH-5 by summing up the totals of the length fields of the various subfields and comparing the result to the overall length field for the FMH-5. For FMH-5s supplied as part of conversation allocation, VTAM checks that:

- The application program is not requesting a conversation with a control operator transaction program (specified by a transaction program name of X'06F1').
- The application program is not requesting a full-duplex conversation when full-duplex is not supported.
- The sync point capability specified is supported by the session.
- Security subfields, if present, are accepted by the partner LU.

For FMH-5s received from another LU, VTAM also checks that:

- RECONNECT support is not specified in the synchronization field.

- Valid characters are used in the network name in the logical unit of work field.
- The application program is not requesting a full-duplex conversation when full-duplex is not supported.
- Security subfields, if present, are accepted by the application program.
- Only one of each type of access security subfield is present.
- An unrecognizable security subfield is not present.

The application program must check for many of the potential errors. The application program ensures that:

- If a user identifier field is present, either a password field is present or the already-verified indicator or the persistent verification signed-on indicator is set.
- If a profile subfield is present, a user identifier subfield is also present.
- Any other security information in the FMH-5 that is not checked by VTAM is accurate.
- If specified, security indicators are in architecturally valid combinations.
- The transaction name is not null (length of 0).
- The transaction name does not specify an SNA service transaction program unless the conversation is authorized.
- The conversation type (basic or mapped) is correct.
- No attempt is made to allocate a conversation with a mode name of SNASVCMG unless the transaction programs have control operator status.

Example of an FMH-5

As an example, suppose that an FMH-5 header needs to be built for a transaction program named TP1. This FMH-5 includes access security subfields specifying an identifier of PROG1, a password of SAMPLPW, and appended PIP data of TEST. The conversation type is basic and the synchronization level is confirm. There is no LUW or conversation correlator.

In EBCDIC, TP1, PROG1, SAMPLPW, and TEST have the following values:

TP1
X'E3D7F1'

PROG1
X'D7D9D6C7F1'

SAMPLPW
X'E2C1D4D7D3D7E6'

TEST
X'E3C5E2E3'

Therefore, the access security subfield information is a hex string of:

```
100801E2C1D4D7D3D7E60602D7D9D6C7F1
```

The PIP parameter is a hex string of:

```
000C12F5000812E2E3C5E2E3
```

The length of the entire FMH-5, including the access information, but not including the PIP data, is 32 bytes (X'20'). The entire FMH-5, including appended PIP data, is a hex string of:

```
200502FF 0803D000 4003E3D7 F1100801 E2C1D4D7 D3D7E606
02D7D9D6 C7F10000 000C12F5 000812E2 E3C5E2E3
```

Figure 25 on page 156 shows the final FMH-5 broken down into its various fields. The FMH-5 length field does not include the PIP data, but byte 4 must be set to X'08' to indicate that the PIP data is there.

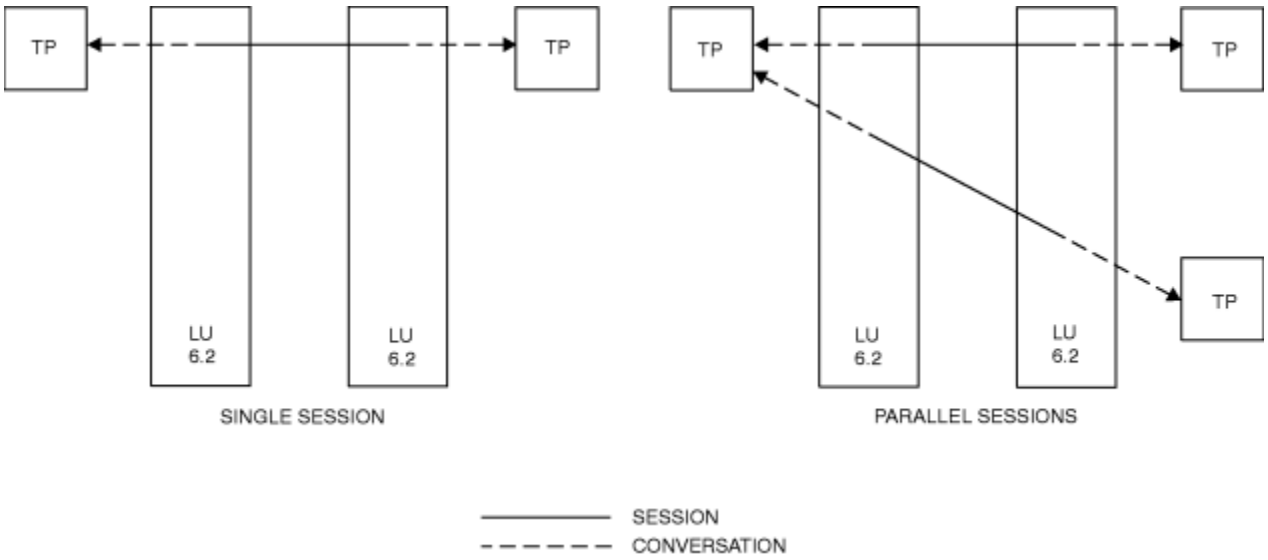


Figure 25. Sample FMH-5 divided into its component fields

Issuing the CONTROL=ALLOC macroinstruction

After the FMH-5 is built, the application program issues the APPCCMD CONTROL=ALLOC macroinstruction to place the FMH-5 in the SEND buffer destined for the partner LU. For conversations allocated on full-duplex-capable sessions, the FMH-5 is not buffered, but sent immediately. For conversations established on half-duplex sessions, the FMH-5 is not sent through the network until the application program issues an APPCCMD macroinstruction that flushes the SEND buffer, or until the application program sends enough additional data to cause VTAM to flush the buffer. The application program uses the AREA parameter on the macroinstruction to point to the FMH-5. The RECLEN parameter specifies the length of the FMH-5.

As an alternative, application programs can issue the APPCCMD CONTROL=PREALLOC macroinstruction, which reserves a session for a conversation without sending the FMH-5. For more information about preallocating a conversation, see “Reserving a session for a conversation” on page 162.

Application programs have several choices of what to specify on the APPCCMD CONTROL=ALLOC macroinstruction. These choices are shown in Table 31 on page 156.

Table 31. Types of conversation allocation

Allocation Type	Description
QUALIFY=ALLOCD	Attempts to start a session for the conversation if one is not available. If a session cannot be established, the session request is queued until a session is available.
QUALIFY=CONVGRP	Can complete successfully if the session specified by the conversation group identifier is available. If the session is not available, the request is queued until it is available.
QUALIFY=CONWIN	Can complete successfully only after a contention-winner session has been allocated for use by the newly created conversation. If no contention-winner session is currently available, the APPCCMD is queued for later completion.
QUALIFY=IMMED	Can complete successfully only if a contention-winner session is available immediately for use.
QUALIFY=WHENFREE	Can complete successfully only if a session is available or pending or one can be activated. This request is never queued waiting for a conversation to complete and free a session.

For an explanation of the types of conversation allocation, see “Types of conversation allocation” on page 59. For details on the syntax and operands for the different forms of the CONTROL=ALLOC macroinstruction, refer to the descriptions in the z/OS Communications Server: SNA Programmer's LU 6.2 Reference.

For optimal performance in allocating a session for a conversation, design the application program so that a contention-winner session is always available.

Buffering requirements

For conversations allocated on full-duplex-capable sessions, the FMH-5 is not buffered when the APPCCMD CONTROL=ALLOC macroinstruction completes, but sent immediately to the partner LU. For conversations allocated on half-duplex sessions, VTAM stores the FMH-5 in the SEND buffer rather than sending it to the partner LU immediately. The application program can issue an APPCCMD CONTROL=SEND, QUALIFY=FLUSH macroinstruction that flushes the SEND buffer if the partner LU needs to receive the FMH-5 promptly. For details on flushing the send buffer, see “Flushing the buffer” on page 178.

Because the conversation allocation does not complete until the FMH-5 is received by the partner LU, many conversation allocation errors cannot be reported on the APPCCMD CONTROL=ALLOC macroinstruction. The errors that are reported are those involving the session used by the conversation.

For example, if no sessions are available to satisfy an APPCCMD CONTROL=ALLOC, QUALIFY=IMMED macroinstruction, an error would be reported to the application program when the macroinstruction completed. Errors involving the conversation, however, are not reported until a subsequent APPCCMD macroinstruction completes. These errors include invalid transaction program names, invalid PIP data, conversation type mismatches, and security errors. Consequently, the application program must be prepared to deal with allocation errors after the APPCCMD CONTROL=ALLOC completes successfully.

Note: The FMH-5 is always buffer traced at the user level if buffer trace is active for the application program.

Example of allocating a conversation

Suppose that an application program with an ACB name of APPLA is to establish a conversation with a partner LU known as APPLB. The conversation should use the EXAMPLE mode, and connect to a transaction program at APPLB known as INQUIRY_MANAGER. APPLA also includes PIP data on the allocation request of "INQUIRY=BALANCE CUSTOMER=899902". The APPCCMD is issued synchronously, and the returned conversation identifier is stored by APPLA in the data area referenced by the CONVERID label. (Assume that the other fields in the FMH-5 used in this example were previously initialized properly and that a CNOS request has been successfully negotiated between the two LUs.)

```
         LA    10,FMH5STOR       * GET FMH-5 ADDRESS
         USING ISTFM5,10         * ESTABLISH ADDRESSABILITY
         MVI   FM5LNTPN,X'0F'    * SET TRANSACTION PROGRAM NAME LENGTH
         MVC   FM5TPNAM(15),=C'INQUIRY_MANAGER' * SET TP NAME
         OI    FM5FLAG2,FM5PIPPR * SET BIT TO INDICATE PIP PRESENT
         MVC   FM5TPNAM+15(3),=X'000000' * SET REST OF FMH5 TO ZEROS
         MVI   FM5LENTH,X'1C'    * SET LENGTH OF FMH5 WITHOUT PIP DATA
         LA    9,28(,10)         * SET ADDRESS OF START OF PIP DATA
         USING FM5PIPFM,9        * ESTABLISH ADDRESSABILITY
         LA    8,32(,10)         * SET ADDRESS OF PIP DATA SUBFIELD
         USING FM5PIPSM,8        * ESTABLISH ADDRESSABILITY
         MVC   FM5PIPLN,=X'0027' * SET LENGTH OF PIP SUBFIELDS AND DATA
         MVC   FM5PIPGD,=X'12F5' * SET PIP GDS ID IN FMH-5
         MVC   FM5PIPSL,=X'0023' * SET LENGTH OF PIP DATA SUBFIELD
         MVC   FM5PIPSG,=X'12E2' * SET SUBFIELD ID IN FMH-5
         MVC   FM5PIPSD(31),=C'INQUIRY=BALANCE CUSTOMER=899902' * DATA
         DROP  8,9               * DROP PIP DSECT REGISTERS

*
* FMH-5 AND PIP ARE SET.  NOW FOR THE APPCCMD.
*
         APPCCMD CONTROL=ALLOC,                                        X
               QUALIFY=ALLOCD,                                         X
               RPL=RPLA,                                               X
               AAREA=RPLAX,                                            X
               ACB=APPLA,                                              X
```

```
               LUNAME=APPLB,                                          X
               LOGMODE=EXAMPLE,                                       X
               OPTCD=SYN,                                             X
               AREA=(10),                                             X
               RECLEN=67
*
         LTR   15,15             * CHECK GENERAL RETURN CODE IN 15
         BNZ   BADGENRC          * HANDLE NONZERO RETURN CODE
         LTR   0,0               * CHECK CONDITIONAL COMPLETION
         BNZ   BADCOND           * HANDLE NONZERO RETURN CODE
         LA    8,RPLAX           * GET RPL EXTENSION ADDRESS
         USING ISTRPL6X,8        * ESTABLISH ADDRESSABILITY
         MVC   CONVERID,RPL6CNVD * STORE THE CONVERSATION ID
                  .
                  .
                  .
CONVERID DS    XL4               * RETURNED CONVERSATION ID
FMH5STOR DS    XL255             * FMH-5 AND PIP STORAGE
RPLA     RPL AM=VTAM             * RPL STORAGE
RPLAX    ISTRPL6                 * RPL EXTENSION STORAGE
APPLA    ACB AM=VTAM,MACRF=LOGON,APPLID=APPLNAME * ACB STORAGE
```

Responding to an FMH-5

An application program participates in a conversation initiated by another transaction program by issuing APPCCMD CONTROL=RCVFMH5 to receive the FMH-5 that was sent.

Application programs must always issue APPCCMD CONTROL=RCVFMH5 when notified of the arrival of an FMH-5. They cannot reject a conversation request by refusing to receive the FMH-5.

If an application program cannot support a conversation when notified of the arrival of the FMH-5, it can wait to issue APPCCMD CONTROL=RCVFMH5 until it can do so, or it can issue the macroinstruction and then use one of the abnormal conversation termination macroinstructions to immediately end the conversation. (See “Checking the received FMH-5” on page 160 for more details on ending the conversation.)

The RCVFMH5 macroinstruction specifies the following items:

- A buffer in the application program's storage where VTAM can store the FMH-5
- The length of the buffer area
- The continuation modes for receiving normal and expedited information

(For details on the syntax and operands for the macroinstruction, refer to the z/OS Communications Server: SNA Programmer's LU 6.2 Reference.)

If several FMH-5s are waiting to be received, and those FMH-5s can be received, they are processed in a first-in, first-out manner. When persistent sessions are used, the application program might be unable, temporarily, to receive FMH-5s.

Application programs are made aware of an FMH-5 to be processed from the following sources:

- Feedback from the APPCCMD macroinstruction, which returns two parameters (FMH5RCV and FMH5LEN) that indicate there is an FMH-5 waiting to be received. FMH5LEN indicates the length of the FMH-5. The RPL fields are set as long as an FMH-5 is waiting to be received. See “Keywords and returned parameters” on page 72 for a complete list of the macroinstructions on which these parameters are returned.
- The ATTN exit, which is driven when an FMH-5 is received. The ATTN exit normally is driven only once for each FMH-5. With persistent LU-LU sessions, it might be driven once for each FMH-5 for each application instance.

If the ATTN exit is present, both methods are available to the application program.

As an alternative to these types of notification, the application can bypass ATTN exit processing by issuing an RCVFMH5 request that is queued by VTAM before the FMH-5 is received. See “Queuing the RCVFMH5 macroinstruction” on page 161 for more information. Application programs can also maintain an internal timer and periodically issues APPCCMD CONTROL=RCVFMH5, QUALIFY=NULL. In this case, the application program must deal with nonzero return codes when no FMH-5s are available to receive.

Restrictions on types of notification

If you are using feedback from the APPCCMD macroinstruction, remember that at times this notification is not available. If the application program is not participating in a conversation, or is participating in a conversation but has no APPCCMD macroinstructions outstanding, the application program cannot be informed of the receipt of an FMH-5 through the feedback fields in the RPL.

If you are using the ATTN exit, be aware that the information on the length of the FMH-5, shown in the RPL extension, might not reflect the length of the next FMH-5 to be received. When set by VTAM, the RPL extension FMH5LEN field contains the length of the longest FMH-5, if FMH-5s are queued. When an ATTN exit is driven, the length of the FMH-5 for which the ATTN exit is driven is found in the same RPL extension feedback field (the exit's parameter list points to a read-only RPL). If a queue of FMH-5s has formed, this will not be the next one to be received. Any problems with the length can be avoided by reserving 255 bytes for any FMH-5 that needs to be received, because an FMH-5 cannot exceed that length. When the application program receives the FMH-5, the RPL's RECLEN field contains the length of the FMH-5.

If you are using the ATTN exit, remember that it is driven only once for each FMH-5. With persistent LU-LU sessions, it might be driven once for each FMH-5 for each application instance. If the application program lacks the resources to start a conversation at that time, the exit can note in any way convenient for the application program that an FMH-5 needs to be received. The mainline program can receive the FMH-5 at a later time.

Example of receiving an FMH-5

This section provides an example for receiving an FMH-5. Figure 26 on page 160 shows an application program, APPLA, that has its ATTN exit driven for the receipt of an FMH-5. The application program must issue RCVFMH5 to accept the FMH-5. The application program must save the LU name of the conversation partner, the mode being used by the conversation, and the returned conversation identifier. This information is saved in the storage areas LUNAME, MODE, and CONVID.

```
         CLC   12(4,1),=C'FMH5'  * WAS EXIT DRIVEN FOR FMH-5?
         BE    FMH5IN            * GO TO FMH5 CODE IF SO
               .
               .
               .
FMH5IN   DS    0H
               .
               .
               .

*
         APPCCMD CONTROL=RCVFMH5,                                      X
               RPL=RPLA,                                               X
               AAREA=RPLAX,                                            X
               ACB=APPLA,                                              X
               OPTCD=SYN,                                              X
               CONMODE=LLCA,                                           X
               CONXMOD=CA,                                             X
               AREA=FMH5INST,                                          X
               AREALEN=255
*
         LTR   15,15             * CHECK GENERAL RETURN CODE IN 15
         BNZ   BADGENRC          * HANDLE NONZERO RETURN CODE
         LTR   0,0               * CHECK CONDITIONAL COMPLETION
         BNZ   BADCOND           * HANDLE NONZERO RETURN CODE
         LA    9,RPLAX           * GET RPL EXTENSION ADDRESS
         USING ISTRPL6X,9        * ESTABLISH ADDRESSABILITY
         MVC   LUNAME,RPL6LU     * SAVE PARTNER LU NAME
         MVC   MODE,RPL6MODE     * SAVE MODE NAME
         MVC   CONVID,RPL6CNVD   * STORE THE CONVERSATION ID
                  .
                  .
                  .
MODE     DS    XL8               * MODE NAME
LUNAME   DS    XL8               * PARTNER LU NAME
CONVID   DS    XL4               * RETURNED CONVERSATION ID
FMH5INST DS    XL255             * RECEIVED FMH-5
RPLA     RPL AM=VTAM             * RPL STORAGE
RPLAX    ISTRPL6                 * RPL EXTENSION STORAGE
APPLA    ACB AM=VTAM,MACRF=LOGON,APPLID=APPLNAME * ACB STORAGE
```

Figure 26. Example of receiving an FMH-5

Receiving PIP and DCE data

To receive PIP or DCE data that accompanies the FMH-5, the application program must either specify QUALIFY=DATQUE on the RCVFMH5 or issue a separate APPCCMD CONTROL=RECEIVE. GDS data included with the FMH-5 is not received by an APPCCMD CONTROL=RCVFMH5, QUALIFY=NULL|QUEUE. (VTAM treats PIP and DCE data as a separate logical record.)

Receiving the GDS data is no different than receiving any other logical record. Once the GDS data is received, the application program must extract the data from the GDS variable's subfields. (See “Logical records versus buffers” on page 211 for information on receiving logical records.)

Checking the received FMH-5

The application program must validate the received FMH-5. For details on the format of the FMH-5, see “Building an FMH-5” on page 150.

If an error is detected in the FMH-5, the application program must issue an APPCCMD CONTROL=DEALLOC, QUALIFY=ABNDUSER, or an APPCCMD CONTROL=REJECT, QUALIFY=CONV macroinstruction. The application program must specify in the SENSE field of the RPL extension one of the valid sense codes shown in the following list. The specified sense code is sent to the partner LU in an FMH-7 (error description). The possible sense codes applicable to allocation errors and their meanings follow:

Code
Meaning

X'080F6051'
Some type of security violation has occurred. The FMH-5 might have specified a transaction program name that the partner LU cannot have a conversation with, or information in an access security subfield is not valid.

X'084B6031'
Transaction program not available, but retry allowed. The FMH-5 specifies a transaction program that the receiver is unable to start. Either the transaction program is not authorized to execute or the resources to execute it are not immediately available. The condition is temporary. The FMH-5 sender is responsible for trying again.

X'084C0000'
Long-term insufficient resources. The FMH-5 specifies a transaction program that the receiver is unable to start. The sender should not try again immediately.

X'1008600B'
Unrecognized FM header command code. The partner LU received an FM header command code that it does not recognize.

X'10086021'
Transaction program name not recognized. The FMH-5 specifies a transaction program name that the receiver does not recognize.

X'10086031'
PIP not allowed. The FMH-5 specifies that PIP data is present, but the receiver does not support PIP data for the specified transaction program.

X'10086032'
PIP not specified correctly. The FMH-5 specifies a transaction program name that requires PIP data and either the FMH-5 specifies that PIP data is not present or the number of PIP subfields present does not agree with the number required for the program.

X'10086034'
Conversation type mismatch. The FMH-5 specifies a conversation type that the receiver does not support for the specified transaction program.

X'10086041'
Synchronization level not supported. The FMH-5 specifies a synchronization level that the receiver does not support for the specified transaction program.

X'10086042'
Reconnection not supported. The FMH-5 specifies reconnection support but the receiver does not support reconnection for the specified transaction program.

Queuing the RCVFMH5 macroinstruction

The QUALIFY=QUEUE|DATAQUE option on the APPCCMD CONTROL=RCVFMH5 macroinstruction allows the application to bypass ATTN(FMH-5) exit processing when VTAM receives an FMH-5. If the QUEUE option is specified, VTAM moves the FMH-5 immediately to the application's buffer and bypasses ATTN(FMH-5) exit processing.

Receiving data along with the FMH-5

Conversation allocation requests can contain data in addition to the FMH-5. If the application receives the FMH-5 with QUALIFY=NULL or QUALIFY=QUEUE, the application's buffer is not readily available to accept data. Instead, VTAM moves the data to a data space to free I/O buffers. This requires the application to issue a CONTROL=RECEIVE to accept the data after receiving the FMH-5. Applications can reduce both macroinstruction processing and receive pathlength by issuing an APPCCMD CONTROL=RCVFMH5, QUALIFY=DATAQUE before VTAM receives the partner's allocation request. VTAM queues the request if no FMH-5 are outstanding. When an FMH-5 is received by VTAM, VTAM moves the FMH-5 immediately to the application's buffer and bypasses ATTN(FMH-5) exit processing. Any data that accompanies the FMH-5 is transferred to the application. VTAM examines the setting of the application's FILL parameter to determine how to handle the data. The remainder of the macroinstruction is processed similar to a receive request.

Reserving a session for a conversation

An application program can obtain information about a conversation before initiating the conversation by issuing the APPCCMD CONTROL=PREALLOC macroinstruction. This puts the conversation in PENDING_ALLOCATE state. VTAM returns a conversation identifier (CONVID) and related information for the pending conversation.

VTAM may also return information in the VTAM-APPCCMD vector list, which can be requested by the application by specifying a vector list area (VTRINA) and the length of that area (VTRINL) on an APPCCMD macroinstruction. Applications should examine the VTAM-APPCCMD vector list when using any of the following functions:

Password Substitution
Applications that implement LU 6.2 option set 223 for password substitution must examine the following vectors:

- Local-nonce vector (ISTAPC13)
- Partner's-nonce vector (ISTAPC14)
- Send-FMH_5-sequence-number vector (ISTAPC15)
- Receive-FMH_5-sequence-number vector (ISTAPC16)
- Partner-application-capabilities vector (ISTAPC1A)

Extended Security Sense Codes
Applications that implement LU 6.2 option set 225 should examine the partner-application-capabilities vector (ISTAPC1A) to determine whether the partner can interpret the extended security sense codes.

DCE Security
Applications that implement LU 6.2 option sets 230 and 231 should examine the partner's-DCE-capability vector (ISTAPC12).

High Performance Data Transfer (HPDT)
Applications using the HPDT interface should examine the session-information vector (ISTAPC19).

The application can also obtain other information about the conversation from VTAM-APPCCMD vector list. See"Vector lists used during APPCCMD processing" on page 25 for more information.

To establish a conversation reserved by the APPCCMD CONTROL=PREALLOC macroinstruction, the application issues the APPCCMD CONTROL=SENDFMH5 macroinstruction specifying the CONVID of the pending conversation. This places the FMH-5 in the SEND buffer destined for the partner LU. For conversations whose preallocated sessions are full-duplex-capable, the FMH-5 is not buffered, but sent immediately. For conversations whose preallocated sessions are half-duplex-capable, the FMH-5 is not sent until one of the following conditions occurs:

- The application program issues an APPCCMD macroinstruction that flushes the SEND buffer.
- The application program sends enough additional data to cause VTAM to flush the buffer.

The application program uses the AREA parameter on the macroinstruction to point to the FMH-5. The RECLEN parameter specifies the length of the FMH-5.

On the APPCCMD CONTROL=PREALLOC macroinstruction, applications can specify any of the QUALIFY values that are available for the APPCCMD CONTROL=ALLOC macroinstruction. SeeTable 31 on page 156 for more information about these QUALIFY values.

To deallocate a conversation reserved by the APPCCMD CONTROL=PREALLOC macroinstruction, use one of the following macroinstructions:

- APPCCMD CONTROL=DEALLOC, QUALIFY=ABNDPROG
- APPCCMD CONTROL=DEALLOC, QUALIFY=ABNDSERV
- APPCCMD CONTROL=DEALLOC, QUALIFY=ABNDTIME
- APPCCMD CONTROL=DEALLOC, QUALIFY=ABNDUSER

- APPCCMD CONTROL=DEALLOCQ

Description of the conversation identifier

When a conversation is successfully allocated, VTAM assigns a unique conversation identifier (CONVID) to it. This identifier is returned to the application program in the RPL6CNVD field of the RPL extension on completion of the APPCCMD CONTROL=ALLOC and APPCCMD CONTROL=RCVFMH5 macroinstructions.

The application program uses the conversation identifier to specify which conversation it will use when it issues conversation-based macroinstructions such as APPCCMD CONTROL=SEND or APPCCMD CONTROL=RECEIVE. For a complete list of the APPCCMDs that can specify a conversation identifier, see “Keywords and returned parameters” on page 72.

Description of the session instance identifier

When an APPCCMD CONTROL=ALLOC, CONTROL=PREALLOC, or CONTROL=RCVFMH5 completes successfully, VTAM places a session identifier in the SESSID field of the RPL extension (RPL6SSID in the ISTRPL6X DSECT). This session identifier uniquely identifies the session being used by the conversation. The PLU is responsible for generating the SESSID. When VTAM generates this SESSID, it is derived from the procedure-correlation identifier (PCID).

The session identifier may be the same as the session ID displayed by the DISPLAY NET,SESSIONS VTAM operator command. This case is true if the bind request contains a session instance identifier structure subfield with a format byte equal to X'01' and the corresponding BIND response contains a session instance identifier structure subfield with a format byte equal to X'02'. If the retired format bytes (X'00' in the BIND request, X'00' or X'F0' in the BIND response) are present in the session instance identifier subfield, then the session identifier will not be the same as that displayed by the D NET,SESSIONS VTAM operator command.

The identifier ranges from 2 to 8 bytes in length. The length of the session identifier is returned in SESSIDL (RPL6SIDL in the DSECT).

An APPCCMD CONTROL=RCVFMH5 macroinstruction can complete successfully without having a session identifier returned (SESSIDL=0). This is the case when the use of an associated session for a pending conversation is complete, and the session has been released for use by another conversation.

Session activation

In addition to automatically activating sessions as the result of a CNOS request, VTAM can activate sessions as the result of an allocation request if no free sessions are available. In such cases, when the number of active contention-winner sessions for an application program is less than its minimum number of contention-winner sessions, VTAM activates a contention-winner session. If the number of active contention-winner sessions for an application program equals or exceeds its minimum number of contention-winner sessions, but the overall session limit has not been reached, VTAM can activate either a contention-winner or contention-loser session.

Type of session activated

The type of session that VTAM activates in response to the allocation request depends on the following conditions:

- If the number of active sessions is less than the new overall session limit, and the sum of the new minimum number of contention-winner sessions for both the source side and target side is less than the new session limit, both LUs can activate additional contention-winner sessions on a first-come-first-serve basis up to the new session limit. These sessions can be activated only in response to allocation requests.

  In this case, the difference between the overall session limit and the sum of the contention-winner sessions can be thought of as a pool of potential contention-winner sessions that either LU can use.

- If the number of active contention-winner sessions for either LU equals the overall session limit minus the minimum number of contention-winner sessions for its partner, and the number of active sessions is less than the overall session limit, the LU can activate additional contention-loser sessions up to the new session limit. These sessions can be activated only in response to allocation requests.

  In this case, any pool of potential contention-winner sessions that might have existed has been used. The only available session resources are from the pool of potential contention-winner sessions guaranteed to the partner LU. Consequently, VTAM can activate only a contention-loser session.

Number of sessions activated

The AUTOSES value, which is specified either on the APPL definition statement or with the APPCCMD CONTROL=OPRCNTL, QUALIFY=DEFINE macroinstruction, plays an important role in determining the number of sessions activated as the result of a CNOS macroinstruction. As stated in the rules for session activation, the AUTOSES value places a ceiling on the number of sessions that are automatically activated using a given mode name. It has no impact on the actual limits, and new sessions beyond the AUTOSES number can be activated in response to allocation requests. (See “VTAM's role in session activation and deactivation” on page 121 for information on activating sessions.)

Sync point capability

During session activation, the BIND for the requested session is also negotiated for sync point capability. If both LUs have SYNCLVL=SYNCPT coded on the APPL definition statement, all sessions are bound as sync point sessions. These sessions support both protected and unprotected conversations. (The SLDSYNCH field in the DEFINE/DISPLAY control block is set to B'10'.)

If one or both of the LUs have SYNCLVL=CONFIRM coded on the APPL definition statement or if no synchronization level is specified, all the sessions between the two LUs are bound as CONFIRM level sessions. Protected conversations are not allowed on these sessions (the SLDSYNCH field in the DEFINE/DISPLAY control block is set to B'01').

Full-duplex session capability

The full-duplex capability of sessions between two LUs is determined during BIND negotiation. If both applications specify PARMS=(FDX=YES) on the OPEN for the ACB, then all sessions will be full-duplex capable. Otherwise, all sessions will be half-duplex capable. If the BIND exchange results in a full-duplex-capable session, transaction programs are able to send data in an expedited manner, bypassing normal paced protocols. Sessions that are negotiated to a full-duplex-capable level support sending and receiving expedited data whether the conversations are full-duplex or half-duplex. If the BIND exchange does not result in a full-duplex-capable session, sending and receiving expedited data is not supported.

Synchronizing end points after session activation failure

In certain circumstances, VTAM attempts to recover from an LU failure by automatically issuing a CNOS to resynchronize session limits. If a VTAM negotiates a CNOS request successfully, the negotiated information is saved. If the partner experiences a power failure or other event and terminates all sessions and also loses its session limits, VTAM automatically reissues an internal CNOS request on a subsequent allocation request. (The partner would have to initiate a CNOS request before it could issue an allocation request.) For example, if VTAM cannot activate a session in response to the allocation request, VTAM fails the allocation request, unless it receives one of the following sense codes:

- X'08050000'
- X'08050001'
- X'0835'*nnnn*, where bytes 2 and 3 contain an offset pointing to a mode name in the user data field and that mode name is not SNASVCMG or CPSVCMG

When VTAM receives any one of these sense codes, VTAM attempts to renegotiate the session limits using an internal CNOS request and issuing the session activation request again. The internal CNOS request

uses the values currently in the LU-mode table. These values would have been negotiated successfully on a previous CNOS request.

Chapter 8. Deallocating a conversation

About this chapter

The deallocation process ends a conversation. Deallocation occurs under two circumstances:

- Normal conversation deallocation
- Deallocation due to errors on the conversation (abnormal deallocation)

Normal conversation deallocation differs for full-duplex and half-duplex conversations. Full-duplex conversations require both transaction programs to issue a deallocate to complete normal conversation deallocation. Half-duplex conversations can be normally deallocated by either transaction program as long as it is in SEND state. The process for abnormal conversation deallocation also differs according to the type of conversation. The processes for each type of conversation deallocation are explained in this chapter.

Deallocating a half-duplex conversation

Normal and abnormal deallocation processes are discussed in this section.

Deallocating a half-duplex conversation normally

Half-duplex conversations are normally deallocated by the transaction program in SEND or PENDING_SEND state. The deallocation can be unconditional or conditional. It can also include a request to send data or transmit buffered data to the partner LU.

Normal conversation deallocation occurs when the side of the conversation in a sending state issues an APPCCMD CONTROL=DEALLOC macroinstruction. The partner application program is informed of the deallocation request through feedback on an APPCCMD macroinstruction (probably an APPCCMD CONTROL=RECEIVE macroinstruction).

The application program in a receiving state can be informed of the deallocation by way of a bit (DEALLOCATE) in the What-Received field of the RPL extension on an APPCCMD CONTROL=RECEIVE macroinstruction. It also can be informed of the deallocation if it attempts to issue an APPCCMD CONTROL=SEND, QUALIFY=ERROR macroinstruction. If it attempts to issue an APPCCMD CONTROL=SEND, QUALIFY=ERROR macroinstruction, it receives an error return code indicating the partner has deallocated the conversation.

In any of the varieties of the CONTROL=DEALLOC macroinstruction, any buffered data that has not yet been sent to the partner LU is transmitted through the network. The various forms of this macroinstruction involve two major choices:

- Whether to send more data
- Whether to send a confirmation request

If the application program will do neither, it should use the QUALIFY=FLUSH form of the macroinstruction. The QUALIFY=DATAFLU form sends additional data and deallocates the conversation. The QUALIFY=CONFIRM form sends a confirmation request and deallocates the conversation. The QUALIFY=DATACON form sends both data and a confirmation request in addition to deallocating the conversation. These choices are shown in Table 32 on page 167.

Table 32. Macroinstructions for normal deallocation of half-duplex conversations

Sends Data	Confirm	Macroinstruction
Yes	Yes	DEALLOC/DATACON
No	Yes	DEALLOC/CONFIRM

Table 32. Macroinstructions for normal deallocation of half-duplex conversations (continued)

Sends Data	Confirm	Macroinstruction
Yes	No	DEALLOC/DATAFLU
No	No	DEALLOC/FLUSH

Sending information

Most of the decisions involved in sending information with a deallocation macroinstruction are the same as for sending information with a CONTROL=SEND macroinstruction. For general information on sending information, see Chapter 9, “Sending information,” on page 177.

Because the conversation is terminated if the macroinstruction completes successfully, the data sent must complete a logical record. The VTAM program checks for the end of the record and fails the macroinstruction if the data does not complete a logical record.

Sending confirmation requests

Requests for confirmation are valid only for half-duplex conversations. Requests for confirmation on full-duplex conversations result in a parameter check.

The choice of whether to send a confirmation request determines whether the deallocation request is conditional or unconditional.

When a confirmation request is sent, the conversation is not deallocated until a positive response is received. If the partner application program sends notification of an error in response to the confirmation request, the conversation is not deallocated.

Conditional deallocation request

A deallocation request that includes a confirmation request is conditional. As is the case with a confirmation request on an APPCCMD CONTROL=SEND macroinstruction, a negative response is reported as an error code on the macroinstruction.

Unconditional deallocation request

If a deallocation is unconditional (specified with QUALIFY=FLUSH or QUALIFY=DATAFLU), the application program in RECEIVE state cannot stop the deallocation. The receiving application program cannot supply feedback information. If it does send error data, the data is lost. Error data unrelated to the deallocation request can also be lost. As an example, assume that an application program issues the following macroinstructions:

- APPCCMD CONTROL=ALLOC, QUALIFY=IMMED
- APPCCMD CONTROL=DEALLOC, QUALIFY=DATAFLU

For half-duplex sessions, VTAM buffers the FMH-5 created as a result of the allocation, and *nothing* is sent to the partner LU until after the application program has ended the conversation. If the partner detects an error in the FMH-5, it has no way to report it to the application program that initiated the conversation, unless it starts another conversation. (The conversation is deallocated before the application program can determine whether the partner received data.) For half-duplex conversations on full-duplex-capable sessions, the FMH-5 is not buffered as a result of the allocation, but is sent immediately to the conversation partner.

If the potential loss of error information is unacceptable, the application must use a conditional deallocation request that includes a confirmation request.

When the application program requests conditional deallocation, it must wait for a confirmation response. The partner can reject the deallocation request.

The partner can respond negatively to the confirmation request, indicating an error, and also deallocate the conversation using one of the abnormal termination macroinstructions. This would be reported

to the application program that is attempting to deallocate the conversation normally as one of the DEALLOCATE_ABEND return codes.

When the partner responds negatively to a confirmation request, it can send error data to the application program to be placed in the system log. The LOGRCV field in the RPL extension is set to YES to indicate this to the application program. When the LOGRCV field in the RPL extension is set to YES, the application program must issue APPCCMD CONTROL=RECEIVE, QUALIFY=SPEC|ISPEC to receive the error log data. If the data does not arrive, or has the wrong format, the application program must issue APPCCMD CONTROL=REJECT to both terminate the conversation and end the session. The application program includes a sense code on the macroinstruction to describe the exact nature of the error. Refer to the *SNA Transaction Programmer's Reference Manual for LU Type 6.2* for a list of possible sense codes.

If the partner LU responds negatively to the confirmation request, VTAM puts the application program in RECEIVE state. The application program must wait for the partner LU to deallocate the conversation or attempt to enter SEND state before it can again try to deallocate the conversation.

Example of normal half-duplex conversation deallocation

Suppose that an application program known as APPLA is deallocating a conversation. The conversation is known by a conversation identifier saved in the storage area CONVERID. APPLA conditionally deallocates the conversation.

```
         L     9,CONVERID        * LOAD SAVED CONVERSATION ID

*
         APPCCMD CONTROL=DEALLOC,                                      X
               QUALIFY=CONFIRM,                                        X
               RPL=RPLA,                                               X
               AAREA=RPLAX,                                            X
               ACB=APPLA,                                              X
               CONVID=(9),                                             X
               OPTCD=SYN
*
         LTR   15,15             * CHECK GENERAL RETURN CODE IN 15
         BNZ   BADGENRC          * HANDLE NONZERO RETURN CODE
         LTR   0,0               * CHECK CONDITIONAL COMPLETION
         BNZ   BADCOND           * HANDLE NONZERO RETURN CODE
*
* GOOD RETURN CODES MEAN THAT THE CONFIRMATION REQUEST WAS
* POSITIVE AND THE CONVERSATION HAS BEEN DEALLOCATED.  NOTE THAT
* THE LOGMODE AND LUNAME WERE NOT NEEDED FOR DEALLOCATION — JUST
* THE CONVERSATION ID.  IF THE RETURN CODES WERE NONZERO, THE
* CONVERSATION IS STILL ACTIVE UNLESS THE CONVERSATION PARTNER
* ABNORMALLY DEALLOCATED THE CONVERSATION FROM ITS END.  APPLA
* AT THAT POINT MUST CHECK THE RCPRI AND RCSEC CODES TO FURTHER
* DETERMINE THE STATUS OF THE CONVERSATION AND POSSIBLE CAUSES FOR
* THE NEGATIVE RESPONSE.  APPLA WOULD ALSO CHECK THE RPL6RTUN BYTE
* IN THE RPL EXTENSION TO SEE IF THE RPL6RLOG BIT HAS BEEN SET.  IF
* SET, ERROR LOG DATA WOULD NEED TO BE RECEIVED.
*
                .
                .
                .
CONVERID DS    XL4              * CONVERSATION ID
RPLA     RPL AM=VTAM            * RPL STORAGE
RPLAX    ISTRPL6                * RPL EXTENSION STORAGE
APPLA    ACB AM=VTAM,MACRF=LOGON,APPLID=APPLNAME * ACB STORAGE
```

Reacting to a Half-duplex conversation deallocation

In cases where the conversation is unconditionally deallocated, the receiving application program can do little more than recognize that its partner has deallocated the conversation. Receiving application programs are informed by the deallocate indicator (RPL6WDAL) in the What-Received field in the RPL extension. In such cases, the deallocate bit is set on, and the confirm bit is set off. The application program does not have to report errors to the partner because the conversation is gone. The application program can attempt to initiate a new conversation with the former partner and attempt to communicate the unreported error.

If a confirmation request is included with the deallocation request (RPL6WCFM is set on), the receiving application program has much more flexibility. The conversation is not deallocated unless the application program responds to the confirmation request positively. An application program uses the APPCCMD CONTROL=SEND, QUALIFY=CONFRMD macroinstruction to respond positively. It can issue the APPCCMD CONTROL=SEND, QUALIFY=ERROR macroinstruction or one of the abnormal termination macroinstructions to respond negatively. If it uses APPCCMD CONTROL=SEND, QUALIFY=ERROR, it is placed in SEND state. An abnormal termination macroinstruction ends the conversation.

On both the APPCCMD CONTROL=SEND, QUALIFY=ERROR macroinstruction and the various forms of APPCCMD CONTROL=DEALLOC|DEALLOCQ that abnormally terminate the conversation, the application program can specify error log data on the macroinstruction. The error data is pointed to by the AREA field of the RPL. The length of the error data is indicated by the RECLEN field. If RECLEN has a value of 0, the application program has not provided any error data. Error data is in the form of a GDS error log variable. For information on the layout of the error log variables, see “Error log variables” on page 272.

Example of receiving a normal Half-duplex conversation deallocation

In the following example, an application program known as APPLA is receiving data over a conversation identified by the identifier stored in the CONVERID storage area. APPLA issues a RECEIVE and is returned information indicating that the partner has conditionally deallocated the conversation. For the purposes of this example, APPLA always responds positively to the confirmation request if a flag byte referred to by the STATUS label is set to zeros.

```
         L     9,CONVERID        * LOAD SAVED CONVERSATION ID
*
         APPCCMD CONTROL=RECEIVE,                                      X
               QUALIFY=SPEC,                                           X
               RPL=RPLA,                                               X
               AAREA=RPLAX,                                            X
               ACB=APPLA,                                              X
               CONVID=(9),                                             X
               OPTCD=SYN,                                              X
               AREA=RECAREA,                                           X
               AREALEN=255,                                            X
               FILL=LL,                                                X
               CONMODE=CS
*
         LTR   15,15             * CHECK GENERAL RETURN CODE IN 15
         BNZ   BADGENRC          * HANDLE NONZERO RETURN CODE
         LTR   0,0               * CHECK CONDITIONAL COMPLETION
         BNZ   BADCOND           * HANDLE NONZERO RETURN CODE
         LA    9,RPLAX           * LOAD RPL EXTENSION ADDRESS
         USING ISTRPL6X,9        * ESTABLISH ADDRESSABILITY
         MVC   SAVEMASK,RPL6RCV1 * SAVE WHAT-RECEIVED MASK
         BAL   14,DATACHK        * CHECK DATA, PERHAPS SET STATUS BYTE
         TM    SAVEMASK,RPL6WCFM * CONFIRMATION REQUEST INCLUDED?
         BNO   DEALCHCK          * IF NOT, CHECK DEALLOCATION INDICATOR
         CLI   STATUS,X'00'      * IF CONFIRMATION REQUEST, STATUS OK?
         BNE   NEGRESP           * IF NOT ISSUE NEGATIVE RESPONSE
*
* STATUS BYTE WAS OK, SO RESPOND POSITIVELY TO CONFIRMATION REQUEST.
*
         APPCCMD CONTROL=SEND,                                         X
               QUALIFY=CONFRMD,                                        X
               RPL=RPLA,                                               X
               AAREA=RPLAX,                                            X
               ACB=APPLA,                                              X
               CONVID=CONVERID,                                        X
               CONMODE=CS,                                             X
               OPTCD=SYN
*
         LTR   15,15             * CHECK GENERAL RETURN CODE IN 15
         BNZ   BADGENRC          * HANDLE NONZERO RETURN CODE
         LTR   0,0               * CHECK CONDITIONAL COMPLETION
         BNZ   BADCOND           * HANDLE NONZERO RETURN CODE
*
* AT THIS POINT THE CONFIRMATION RESPONSE HAS BEEN MADE AND THE
* RETURN CODES FROM IT WERE GOOD.  NOW CHECK FOR DEALLOCATION.
*
DEALCHCK TM    SAVEMASK,RPL6WDAL * DEALLOCATION INDICATOR SET?
         BNO   SENDCHCK          * IF NOT, GO TO NEXT INDICATOR CHECK
*
* AT THIS POINT, APPLA KNOWS IF THE CONVERSATION HAS BEEN DEALLOCATED.
*
```

```
                    .
                    .
                    .
CONVERID DS    XL4              * CONVERSATION ID
SAVEMASK DS    XL1              * STORAGE TO SAVE WHAT-RECEIVED MASK
STATUS   DC    X'00'            * RECEIVED DATA OK FLAG
RECAREA  DS    XL255            * RECEIVE STORAGE AREA
RPLA     RPL AM=VTAM            * RPL STORAGE
RPLAX    ISTRPL6                * RPL EXTENSION STORAGE
APPLA    ACB AM=VTAM,MACRF=LOGON,APPLID=APPLNAME * ACB STORAGE
```

Deallocating a Half-duplex conversation abnormally

To deallocate a conversation because of an error, application programs can use any of the following macroinstructions:

- APPCCMD CONTROL=DEALLOC|DEALLOCQ, QUALIFY=ABNDPROG
- APPCCMD CONTROL=DEALLOC|DEALLOCQ, QUALIFY=ABNDSERV
- APPCCMD CONTROL=DEALLOC|DEALLOCQ, QUALIFY=ABNDTIME
- APPCCMD CONTROL=DEALLOC|DEALLOCQ, QUALIFY=ABNDUSER
- APPCCMD CONTROL=REJECT

For details on the syntax and operands for these macroinstructions, refer to the z/OS Communications Server: SNA Programmer's LU 6.2 Reference.

The APPCCMD CONTROL=DEALLOC|DEALLOCQ macroinstructions abnormally end a conversation without ending the session. The APPCCMD CONTROL=DEALLOQ macroinstruction queues a deallocation request if the conversation is in the RECEIVE state and has not yet received data. When the data is received, VTAM deallocates the conversation. The REJECT macroinstructions end both the conversation and the session.

Using CONTROL=DEALLOC and CONTROL=DEALLOCQ

The application initiates the abnormal deallocation by issuing either the APPCCMD CONTROL=DEALLOC or the APPCCMD CONTROL=DEALLOCQ macroinstructions.

The APPCCMD CONTROL=DEALLOC macroinstruction should be used when it is not desirable to wait for the arrival of information from the partner before deallocating the conversation. VTAM waits for information from the partner before sending a negative response when the conversation is in receive state and information has not been received from the partner to which it can respond. In this case, the APPCCMD CONTROL=DEALLOC is returned to the application program with error return codes. The application program can reissue the macroinstruction or issue APPCCMD CONTROL=REJECT, QUALIFY=CONV to abnormally end the conversation and session.

The APPCCMD CONTROL=DEALLOCQ macroinstruction should be used when it is not desirable to reject the macroinstruction for this situation, but it is desirable to wait for the arrival of information. APPCCMD CONTROL=DEALLOCQ can result in deadlock situations if some condition, such as a failed partner LU, prevents VTAM from receiving additional information. A user who codes the APPCCMD CONTROL=DEALLOCQ macroinstruction on existing application programs must verify that the program logic can tolerate this possible hang situation.

The processing of these macroinstructions causes the LU's send buffer to be flushed and an FMH-7 and any error log data to be sent to the partner LU.

The APPCCMD CONTROL=DEALLOC and APPCCMD CONTROL=DEALLOCQ macroinstructions include four types:

- QUALIFY=ABNDPROG
- QUALIFY=ABNDSERV
- QUALIFY=ABNDTIME
- QUALIFY=ABNDUSER

Each of these types is described in the following section.

ABNDPROG
Specifies that an error makes it impossible to continue meaningful communication over the conversation. It is issued for errors detected by the processing threads in the application program that correspond to transaction programs. An example of this type of error could be the receipt of incorrect data on a CONTROL=RECEIVE macroinstruction. This type of error causes a sense code of X'08640000' to be generated in the FMH-7 used by VTAM in processing the error. (See “Sense codes for FMH-7” on page 273 for a list of FMH-7 sense codes.)

ABNDSERV
Specifies that an LU services component has detected an error. This type of error occurs when the application program has implemented an LU 6.2 function in addition to that provided by VTAM. For example, an application program might implement mapped conversations. Errors relating to the application's implementation would be service errors. If the errors were serious enough to deallocate the conversation, the application program would issue this QUALIFY variation. This type of error causes a sense code of X'08640001' to be generated in the FMH-7 used by VTAM in processing the error. (See “Sense codes for FMH-7” on page 273 for a list of FMH-7 sense codes.)

ABNDTIME
Specifies that the application program has not received expected information within an application-determined length of time. This type of error causes a sense code of X'08640002' to be generated in the FMH-7 used by VTAM in processing the error. (See “Sense codes for FMH-7” on page 273 for a list of FMH-7 sense codes.) ABNDTIME can be used by the application program to handle errors originating on its side of the conversation or when it fails to receive information from a partner LU. Its primary intent is to handle timing errors on the local side of a conversation. For example, if an application program has been waiting for one of its transaction programs to send data, and the wait exceeds the time limit, it would use this type of deallocation to end the conversation.

ABNDUSER
Specifies that the application program is ending the conversation and provides VTAM with a user-specified sense code. The sense code must be valid for an FMH-7. The application program must ensure that it is. An example of the type of error this is used for would be the receipt of an FMH-5 with errors. (See “Sense codes for FMH-7” on page 273 for a list of FMH-7 sense codes.)

Using CONTROL=REJECT

APPCCMD CONTROL=REJECT is used when a severe error indicates that data will not flow properly on the session. For example, an application program is informed through the LOGRCV field that the application program is to receive error log data. If the application program fails to receive the data, an APPCCMD CONTROL=REJECT macroinstruction is issued. (For details on the syntax and operands for the macroinstruction, refer to z/OS Communications Server: SNA Programmer's LU 6.2 Reference.)

When application programs issue the APPCCMD CONTROL=REJECT macroinstruction, they must specify an UNBIND sense code describing the nature of the error.

Deallocating a full-duplex conversation

Due to the nature of full-duplex conversations, the processes for deallocating the conversation differs from the deallocation process for half-duplex conversations. Normal and abnormal conversation deallocation processes are discussed in this section.

Deallocating a full-duplex conversation normally

Both transaction programs must issue an APPCCMD CONTROL=DEALLOC to normally deallocate the conversation. This ensures that data in transit in both directions is delivered before ending the conversation. Normal conversation deallocation can be initiated by either transaction program in SEND/RECEIVE state. The deallocation request can also send additional information to be transmitted to the partner.

Normal conversation deallocation begins when a transaction program issues an APPCCMD CONTROL=DEALLOC, QUALIFY=FLUSH|DATAFLU macroinstruction. The conversation then enters RECEIVE-ONLY state. The transaction program can no longer send information on this conversation.

The transaction program can, however, continue to receive information until an indication of a deallocate issued by the partner is received, at which point its conversation state is changed to FDX_RESET. The partner transaction program enters SEND_ONLY state upon receipt of the deallocate indication and completes sending its data. It then issues a deallocate, changing its conversation state to FDX_RESET.

Sending information

The transaction program initiating a normal deallocation of a full-duplex conversation enters RECEIVE_ONLY state. Therefore, it can no longer send information.

The transaction program receiving the deallocation request enters SEND_ONLY state. This transaction program can continue to send information until it decides to end the conversation by issuing an APPCCMD CONTROL=DEALLOC, QUALIFY=FLUSH|DATAFLU macroinstruction.

Most of the decisions involved in sending information with a deallocation macroinstruction are the same as for sending information with a CONTROL=SEND macroinstruction. For general information on sending information, see Chapter 9, “Sending information,” on page 177.

Because sending of information is no longer allowed if the DEALLOC macroinstruction completes successfully, the data sent must complete a logical record. VTAM checks for the end of the record and fails the macroinstruction if the data does not complete a logical record.

Reacting to a full-duplex conversation deallocation

Application programs are informed by the deallocate indicator (RPL6WDAL) in the What-Received field of an APPCCMD CONTROL=RECEIVE macroinstruction that the partner application has begun deallocation of the conversation. The conversation is then placed in SEND_ONLY state. The application program can continue to send information. When all information has been sent, the application program issues an APPCCMD CONTROL=DEALLOC, QUALIFY=FLUSH|DATAFLU. This ends the full-duplex conversation and the conversation enters FDX_RESET state.

Deallocating a full-duplex conversation abnormally

Conversations are abnormally deallocated when an error is encountered. Application programs can use any of the following macroinstructions for deallocating a full-duplex conversation abnormally:

- APPCCMD CONTROL=DEALLOC|DEALLOCQ, QUALIFY=ABNDPROG
- APPCCMD CONTROL=DEALLOC|DEALLOCQ, QUALIFY=ABNDSERV
- APPCCMD CONTROL=DEALLOC|DEALLOCQ, QUALIFY=ABNDTIME
- APPCCMD CONTROL=DEALLOC|DEALLOCQ, QUALIFY=ABNDUSER
- APPCCMD CONTROL=REJECT

The APPCCMD CONTROL=DEALLOC|DEALLOCQ macroinstructions abnormally end a conversation without ending the session. The APPCCMD CONTROL=REJECT macroinstructions end both the conversation and the session.

Using CONTROL=DEALLOC and CONTROL=DEALLOCQ

The application program initiates the abnormal deallocation by issuing one of the APPCCMD CONTROL=DEALLOC|DEALLOCQ macroinstructions. Processing of these macroinstructions causes the send buffer to be flushed and an FMH-7 and any error log data to be sent to the partner. The conversation is ended and enters FDX_RESET state.

Note: An FMH-7 and any error log data are not sent to the partner when the conversation is in RECEIVE-ONLY state.

There are four types of the APPCCMD CONTROL=DEALLOC and the APPCCMD CONTROL=DEALLOCQ macroinstructions that can be used to abnormally deallocate a full-duplex conversation:

- QUALIFY=ABNDPROG
- QUALIFY=ABNDSERV

- QUALIFY=ABNDTIME
- QUALIFY=ABNDUSER

Each of these types is described in the following section.

ABNDPROG
Specifies that an error makes it impossible to continue meaningful communication over the conversation. It is issued for errors detected by the processing threads in the application program that correspond to transaction programs. An example of this type of error could be the receipt of incorrect data on a CONTROL=RECEIVE macroinstruction. This type of error causes a sense code of X'08640000' to be generated in the FMH-7 used by VTAM in processing the error. (See “Sense codes for FMH-7” on page 273 for a list of FMH-7 sense codes.)

ABNDSERV
Specifies that an LU services component has detected an error. This type of error occurs when the application program has implemented an LU 6.2 function in addition to that provided by VTAM. For example, an application program might implement mapped conversations. Errors relating to the application's implementation would be service errors. If the errors were serious enough to deallocate the conversation, the application program would issue this QUALIFY variation. This type of error causes a sense code of X'08640001' to be generated in the FMH-7 used by VTAM in processing the error. (See “Sense codes for FMH-7” on page 273 for a list of FMH-7 sense codes.)

ABNDTIME
Specifies that the application program has not received expected information within an application-determined length of time. This type of error causes a sense code of X'08640002' to be generated in the FMH-7 used by VTAM in processing the error. (See “Sense codes for FMH-7” on page 273 for a list of FMH-7 sense codes.) ABNDTIME can be used by the application program to handle errors originating on its side of the conversation or when it fails to receive information from a partner LU. Its primary intent is to handle timing errors on the local side of a conversation. For example, if an application program has been waiting for one of its transaction programs to send data, and the wait exceeds the time limit, it would use this type of deallocation to end the conversation.

ABNDUSER
Specifies that the application program is ending the conversation and provides VTAM with a user-specified sense code. The sense code must be valid for an FMH-7. The application program must ensure that it is valid. An example of the type of error this is used for would be the receipt of an FMH-5 with errors. (See “Sense codes for FMH-7” on page 273 for a list of FMH-7 sense codes.)

Refer to z/OS Communications Server: SNA Programmer's LU 6.2 Reference for details on the syntax and operands for these macroinstructions.

Restrictions on abnormally deallocating conversations

Conversations can be abnormally terminated by either partner. The macroinstructions used to abnormally end the conversation can be issued while other APPCCMD macroinstructions are outstanding on the conversation, in which case the outstanding macroinstruction is canceled and the conversation terminated. Cases do exist where the CONTROL=DEALLOC or CONTROL=DEALLOCQ macroinstructions cannot be issued while another APPCCMD macroinstruction is outstanding on a conversation. In addition, the abnormal termination macroinstructions cannot be used to cancel a macroinstruction that does not specify a conversation resource, such as the APPCCMD CONTROL=OPRCNTL macroinstructions. The macroinstructions that cannot be canceled by one of the abnormal termination APPCCMD CONTROL=DEALLOC or CONTROL=DEALLOCQ macroinstructions are:

- APPCCMD CONTROL=SEND, QUALIFY=CONFIRM (or any other APPCCMD that is waiting for a confirmation response or error log data)
- APPCCMD CONTROL=RECEIVE, QUALIFY=SPEC (that has not received any data from the partner LU to send a negative response to)
- APPCCMD CONTROL=RECEIVE, QUALIFY=ANY (that has not been matched to a conversation)
- APPCCMD CONTROL=RCVFMH5
- APPCCMD CONTROL=RESETRCV

- APPCCMD CONTROL=OPRCNTL
- APPCCMD CONTROL=REJECT, QUALIFY=CONV
- APPCCMD CONTROL=REJECT, QUALIFY=SESSION
- One of the other abnormal termination CONTROL=DEALLOC macroinstructions

The application program can wait for these macroinstructions to complete, or it can issue APPCCMD CONTROL=REJECT, QUALIFY=CONV to cancel an outstanding macroinstruction on a conversation. The only conversation-related macroinstruction that APPCCMD CONTROL=REJECT, QUALIFY=CONV cannot cancel is another APPCCMD CONTROL=REJECT, QUALIFY=CONV macroinstruction.

The abnormal termination APPCCMD CONTROL=DEALLOC|DEALLOCQ macroinstructions are always unconditional. The conversation terminates after their completion. If the application program needs to supply error log data on the macroinstruction, it must supply the complete error log variable. It cannot use any subsequent macroinstructions to finish sending the data. (See "Error log variables" on page 272 for a description of the error log variable.) The partner can only receive error log data. As soon as it finishes receiving the data (if supplied), the conversation is finished.

When an application program issues an abnormal termination APPCCMD CONTROL=DEALLOC|DEALLOCQ macroinstruction, data purging or truncation can occur. Data truncation occurs when the application program is in a sending state and issues the macroinstruction before providing the remainder of a logical record. Data purging occurs when the application program issues the macroinstruction in a receiving state before receiving all the information sent by the partner. Any data buffered by VTAM and waiting for a RECEIVE macroinstruction is discarded. For more information on purging, see "Data purging and truncating" on page 271.

Deallocating a pending conversation

There may be situations (such as a timeout) in which the application wants to abnormally terminate a conversation before the APPCCMD CONTROL=ALLOC or APPCCMD CONTROL=RCVFMH5, QUALIFY=DATAQUE macroinstruction has completed. Normally, the CONVID is returned in the RPL6CNVD when the CONTROL=ALLOC macroinstruction completes. However, if the macroinstruction has *not* completed, this information is apparently not available.

It is suggested that the application ensure that the RPL6CNVD is set to zeros before issuing the APPCCMD CONTROL=ALLOC or APPCCMD CONTROL=RCVFMH5, QUALIFY=DATAQUE macroinstruction. Early in the process of conversation allocation, VTAM specifies the RPL6CNVD with the assigned CONVID. Therefore, if the application wants to deallocate the conversation before allocation has completed, the application can check the RPL6CNVD for a nonzero value. If the RPL6CNVD contains a nonzero value, the application can use this information in the APPCCMD CONTROL=DEALLOC macroinstruction to successfully deallocate the pending conversation.

Use one of the following macroinstructions to deallocate a conversation in PENDING_ALLOCATE state:

- APPCCMD CONTROL=DEALLOC, QUALIFY=ABNDPROG
- APPCCMD CONTROL=DEALLOC, QUALIFY=ABNDSERV
- APPCCMD CONTROL=DEALLOC, QUALIFY=ABNDTIME
- APPCCMD CONTROL=DEALLOC, QUALIFY=ABNDUSER
- APPCCMD CONTROL=DEALLOCQ

For the APPCCMD CONTROL=RCVFMH5, QUALIFY=DATAQUE macroinstruction, the application can issue an abnormal termination request only after the FMH-5 has been received and VTAM updates RPL6CNVD.

For more information on the APPCCMD CONTROL=DEALLOC, refer to z/OS Communications Server: SNA Programmer's LU 6.2 Reference.

Rejecting a conversation pending deallocation for persistent sessions

After the failure of a VTAM application program that has enabled persistence, VTAM attempts to deactivate any active conversations. If deallocation of a conversation does not complete, for whatever reason, the conversation remains in a state of pending deallocation for persistent LU-LU sessions until it is brought down for another reason or until the session is deactivated. If the application program tries to use the conversation, VTAM sets a return code that indicates that the conversation identifier is not valid.

If session information is requested on the APPCCMD CONTROL=OPRCNTL, QUALIFY=RESTORE macroinstruction (LIST=ALL), the RESTORE structure indicates whether the *session* is pending deactivation for persistent LU-LU session support (SRESPNDA) or whether the *conversation* is pending deallocation for persistent LU-LU session support (SREPCONV). However, only the session identifier (SESSID) is passed back in the RESTORE structure. (See "Retrieving information for a mode and sessions to be restored" on page 147 for more information.)

After the mode is restored, the application program can issue APPCCMD CONTROL=REJECT, QUALIFY=SESSION for such a session to bring down the session and any existing conversation.

Chapter 9. Sending information

About this chapter

Application programs send normal and expedited information over conversations with the SEND, SENDRCV, PREPRCV, DEALLOC, DEALLOCQ and SENDEXPD varieties of the APPCCMD macroinstruction, according to the full-duplex or half-duplex capabilities of a session. Application programs exchange both data and conversation-related control information, such as confirmation requests and responses.

The VTAM program imposes restrictions on when an application program can send data or control information over a conversation, but it also handles many of the tasks involved in managing the flow of data handled by the application program for non-LU 6.2 sessions.

Both half-duplex and full-duplex conversations allocated to sessions negotiated to full-duplex capability have the ability to send and receive data in an expedited manner, bypassing normal, paced protocols. Expedited data is sent using APPCCMD CONTROL=SENDEXPD, QUALIFY=DATA macroinstruction.

There are some important concepts the application programmer must understand in order to send and receive information effectively and efficiently. The application programmer must fully understand the differences between *half-duplex-capable sessions* and *full-duplex-capable sessions*. The application programmer also needs to understand the capabilities of *half-duplex conversations* and *full-duplex conversations*. To review these concepts, see "Full-duplex and half-duplex protocols" on page 6.

This chapter contains information about VTAM's normal send services and interface. Better send performance is available to applications that use the interface for high performance data transfer (HPDT). For information about HPDT, see Chapter 11, "Sending and receiving data using high performance data transfer," on page 217. Before reading that chapter, however, the reader should be familiar with the information presented in this chapter.

Sending information on half-duplex conversations

This section explains the process and APPCCMDs involved in sending information on half-duplex conversations. For details on sending information on full-duplex conversations, see "Sending information on full-duplex conversations" on page 186.

Background of the SEND buffer

Systems Network Architecture transmits normal data through the network in a chain of request units. Before these units are transmitted, the partners agree to the maximum size of each of the elements. Because there is some fixed amount of work to handle each unit, regardless of its size, network efficiency can be improved if each unit contains the maximum amount of data.

The LU 6.2 architecture provides for a component in each implementation that:

- Accumulates data from the application
- Places the data in a SEND buffer behind any unsent data that remains from a previous operation
- Removes and prepares for transmission a full request unit size worth of data from the SEND buffer if the amount of data in the SEND buffer exceeds [2] the maximum request unit size
- Passes this request unit to another component for actual transmission

[2] The reason the amount of data must exceed the maximum unit size is also for efficiency. If the next macroinstruction causes the flow of normal control information, then that information may be included in the header of the maximum sized unit that was held in reserve. This design precludes the flow of a unit that contains only control information.

When a SEND macroinstruction completes, the amount of data that remains in the SEND buffer is less than or equal to a maximum unit size. A subsequent SEND macroinstruction from the local application will repeat this operation.

Certain macroinstructions cause all of the data in the SEND buffer to be transmitted. This can be either under the explicit control of the application by the use of a flush type of SEND macroinstruction or it may be implicit in that the flush is automatically performed by VTAM. For example, VTAM automatically flushes the SEND buffer, for half-duplex conversations, as part of the operation for the transition of the local transaction program from SEND to RECEIVE state. This automatic flushing ensures that it is in SEND state; it also has all the data that was transmitted by the local transaction program.

Use of the SEND buffer

The SEND buffer is used only to accumulate normal data.

Issuing an APPCCMD CONTROL=SEND macroinstruction to send data does not automatically cause data to flow through the network. Instead, VTAM stores the data in a buffer in VTAM storage until enough is sent to exceed the maximum SEND RU size that was specified in the session parameters used to establish the conversation's underlying session.

The SEND buffer has nothing to do with logical record size. If the buffer fills up before a logical record is completely stored in it, VTAM sends what is in the buffer and stores the remainder of the logical record in the buffer. VTAM maintains the record's integrity throughout the process.

In addition to logical records, VTAM stores function management headers that it uses for conversation control. FMH-7 headers created by the APPCCMD CONTROL=SEND, QUALIFY=ERROR macroinstruction are kept in the SEND buffer until enough data is accumulated from subsequent APPCCMD macroinstructions to exceed the maximum RU size, or the application program issues an APPCCMD forcing VTAM to transmit the contents of the buffer. For conversations allocated on half-duplex sessions, VTAM buffers the FMH-5 header pointed to by the APPCCMD CONTROL=ALLOC macroinstruction.

Flushing the buffer

Requesting VTAM to transmit buffered data is called flushing the SEND buffer. Many of the APPCCMD macroinstructions instruct VTAM to flush the SEND buffer. All macroinstructions with a QUALIFY value of FLUSH, DATAFLU, CONFIRM, or DATACON flush the buffer.

The APPCCMD CONTROL=SEND, QUALIFY=ERROR macroinstruction and the abnormal termination DEALLOC and DEALLOCQ APPCCMD macroinstructions also flush the buffer, but only if the local conversation is in a sending state.

The only macroinstructions that send normal information but do *not* flush the SEND buffer are:

- APPCCMD CONTROL=SEND, QUALIFY=CONFRMD
- APPCCMD CONTROL=SEND, QUALIFY=DATA (except when OPTCD=XBUFLST)
- APPCCMD CONTROL=SEND, QUALIFY=ERROR (except when issued from SEND state)

The buffering of data allows the application program to issue a macroinstruction or several macroinstructions before its partner receives anything. Consequently, the partner frequently is unable to detect errors or give other feedback immediately after an APPCCMD macroinstruction is used to send information. The application program can flush the buffer or use confirmation requests to ensure that its partner receives data quickly.

As an example, an application program might allocate a conversation, send data, and deallocate a conversation before its partner receives the FMH-5 signaling that the conversation is pending. This situation can adversely affect error reporting. Because VTAM has deallocated the conversation for the application program, error information cannot be reported back to the originating application program. In general, anytime a conversation uses the APPCCMD CONTROL=DEALLOC, QUALIFY=FLUSH macroinstruction to flush the buffer and end the conversation, error information can be lost. Consequently, application programs should use confirmation requests before deallocating the conversation if the potential loss of error information is unacceptable.

Example of flushing the buffer

In this example, an application program known as APPLA must flush the buffer. The conversation is identified by a conversation identifier stored in CONVERID. The application program first sends a portion of a logical record before flushing the buffer. This example assumes that the initial send does not cause the buffer to be flushed as a consequence of the maximum RU size being exceeded.

```
*
* FIRST PUT SOME DATA TO FLUSH IN THE BUFFER.
*
         MVC   MSGBUFF(2),=X'0018' * INITIALIZE LOGICAL RECORD HEADER
         MVC   MSGBUFF+2(22),=C'THIS IS A TEST MESSAGE' * MOVE IN DATA
         LA    10,X'18'            * PUT MESSAGE LENGTH IN REGISTER
*
         APPCCMD CONTROL=SEND,                                         X
               QUALIFY=DATA,                                           X
               RPL=RPLA,                                               X
               AAREA=RPLAX,                                            X
               ACB=APPLA,                                              X
               CONVID=CONVERID,                                        X
               AREA=MSGBUFF,                                           X
               RECLEN=(10),                                            X
               OPTCD=SYN
*
         LTR   15,15               * CHECK GENERAL RETURN CODE IN 15
         BNZ   BADGENRC            * HANDLE NONZERO RETURN CODE
         LTR   0,0                 * CHECK CONDITIONAL COMPLETION
         BNZ   BADCOND             * HANDLE NONZERO RETURN CODE
*
* RETURN CODES WERE GOOD. DATA IS IN THE BUFFER. NOW FLUSH IT.
*
         APPCCMD CONTROL=SEND,                                         X
               QUALIFY=FLUSH,                                          X
               RPL=RPLA,                                               X
               AAREA=RPLAX,                                            X
               ACB=APPLA,                                              X
               CONVID=CONVERID,                                        X
               OPTCD=SYN
*
         LTR   15,15               * CHECK GENERAL RETURN CODE IN 15
         BNZ   BADGENRC            * HANDLE NONZERO RETURN CODE
         LTR   0,0                 * CHECK CONDITIONAL COMPLETION
         BNZ   BADCOND             * HANDLE NONZERO RETURN CODE
*
* RETURN CODES WERE GOOD, THE BUFFER HAS BEEN FLUSHED.
*
                   .
                   .
                   .
CONVERID DS    XL4                * CONVERSATION ID
MSGBUFF  DS    XL255              * STORAGE FOR DATA TO SEND
RPLA     RPL AM=VTAM              * RPL STORAGE
RPLAX    ISTRPL6                  * RPL EXTENSION STORAGE
APPLA    ACB AM=VTAM,MACRF=LOGON,APPLID=APPLNAME * ACB STORAGE
```

In this example, two macroinstructions were used (one to send the data to the buffer and one to flush the buffer). The two could have been combined into one macroinstruction:

```
      APPCCMD CONTROL=SEND,                                          X
            QUALIFY=DATAFLU,                                         X
            RPL=RPLA,                                                X
            AAREA=RPLAX,                                             X
            ACB=APPLA,                                               X
            AREA=MSGBUFF,                                            X
            RECLEN=(10),                                             X
            CONVID=CONVERID,                                         X
            OPTCD=SYN
```

If the application program was to deallocate the conversation or change to RECEIVE state, the application program could have specified either APPCCMD CONTROL=DEALLOC|DEALLOCQ or APPCCMD CONTROL=PREPRCV instead of APPCCMD CONTROL=SEND.

Roles of sender and receiver

Application programs and their partners on half-duplex conversations regularly exchange the roles of sender and receiver according to rules enforced by VTAM. On half-duplex conversations, only one partner can send information while the other partner is receiving (except in certain error situations). When an application program is finished sending data, it can surrender its prerogative as the sender to allow its partner to send information. This type of protocol is referred to as half-duplex flip-flop protocol.

For the most part, the sender determines when its partner LU can send logical record data. The sending application program must surrender its prerogative to send data by notifying VTAM that it is ready to receive. This places the local conversation in RECEIVE state. VTAM then notifies the receiving side that it can now send. The receiving application program receives this notification as part of the parameters returned on an APPCCMD CONTROL=RECEIVE macroinstruction. The receiver's side of the conversation is then placed in SEND state.

In most cases, an application program must be acknowledged by VTAM as the sender before transmitting normal information, such as logical record data. The following APPCCMD macroinstructions can be issued only from a conversation in SEND state:

- APPCCMD CONTROL=SEND, QUALIFY=CONFIRM
- APPCCMD CONTROL=SEND, QUALIFY=DATA
- APPCCMD CONTROL=SEND, QUALIFY=DATACON
- APPCCMD CONTROL=SEND, QUALIFY=DATAFLU
- APPCCMD CONTROL=SEND, QUALIFY=FLUSH
- APPCCMD CONTROL=DEALLOC, QUALIFY=CONFIRM
- APPCCMD CONTROL=DEALLOC, QUALIFY=DATACON
- APPCCMD CONTROL=DEALLOC, QUALIFY=DATAFLU
- APPCCMD CONTROL=DEALLOC, QUALIFY=FLUSH
- All APPCCMD CONTROL=PREPRCV macroinstructions

Attempts to issue these macroinstructions without being in SEND state result in an error.

Application programs receiving information need not concern themselves with placing the conversation in SEND state to report an error or send some control information, such as a request for permission to send or a confirmation reply. A receiving application program can issue the following macroinstructions when the local conversation is *not* in SEND state:

- APPCCMD CONTROL=SEND, QUALIFY=CONFRMD
- APPCCMD CONTROL=SEND, QUALIFY=ERROR
- APPCCMD CONTROL=SEND, QUALIFY=RQSEND

When an APPCCMD CONTROL=SEND, QUALIFY=ERROR macroinstruction is issued while in a receiving state, the conversation is placed in SEND state. This situation is an exception to the rule that the sending side of the conversation determines when the partner LU can send data. The partner application's conversation state is changed to RECEIVE when notification of the error is reported on an APPCCMD macroinstruction.

For more information on how an application changes the conversation from SEND state to RECEIVE state, see “ Entering RECEIVE state” on page 203.

Entering SEND state

Application programs that successfully allocate a conversation are automatically designated the sender and enter SEND state. Assuming normal completion of the RCVFMH5, the conversation partner becomes the receiver and enters RECEIVE state. The receiving side can either request to send information, or wait until the partner relinquishes the role.

In general, the conversation is in SEND state after the successful completion of:

- An APPCCMD CONTROL=ALLOC macroinstruction
- An APPCCMD CONTROL=SENDFM5 macroinstruction
- An APPCCMD CONTROL=RECEIVE macroinstruction, and the confirm indicator in the what-received field is *off* and the send indicator in the what-received field is *on*
- An APPCCMD CONTROL=SEND, QUALIFY=CONFRMD macroinstruction after an APPCCMD CONTROL=RECEIVE macroinstruction in which both the confirm and send indicators in the what-received field were set *on*
- An APPCCMD CONTROL=SEND, QUALIFY=ERROR macroinstruction
- One of the APPCCMD CONTROL=SEND macroinstructions used to send information

The receiving side can request permission to send by issuing the APPCCMD CONTROL=SEND, QUALIFY=RQSEND macroinstruction. If the conversation partner accepts the request, it surrenders its right to send by preparing to receive information or receiving information. (In the case of VTAM application programs, this would mean issuing either the APPCCMD CONTROL=PREPRCV macroinstruction or the APPCCMD CONTROL=RECEIVE, QUALIFY=SPEC macroinstruction.)

Example of entering SEND state

In this example, an application program known as APPLA is receiving data and needs to place the conversation in SEND state normally (not because of an error condition). To do so, it uses the RQSEND type of APPCCMD. The conversation identifier is contained in register 8.

```
*
* FIRST ASK FOR PERMISSION TO SEND.
*
         APPCCMD CONTROL=SEND,                                         X
               QUALIFY=RQSEND,                                         X
               RPL=RPLA,                                               X
               AAREA=RPLAX,                                            X
               ACB=APPLA,                                              X
               CONVID=(8),                                             X
               OPTCD=SYN
*
         LTR   15,15              * CHECK GENERAL RETURN CODE IN 15
         BNZ   BADGENRC           * HANDLE NONZERO RETURN CODE
         LTR   0,0                * CHECK CONDITIONAL COMPLETION
         BNZ   BADCOND            * HANDLE NONZERO RETURN CODE
*
* RETURN CODES WERE GOOD, BUT APPLA STAYS IN RECEIVE STATE UNTIL
* BEING INFORMED ON A RECEIVE MACRO.
*
         APPCCMD CONTROL=RECEIVE,                                      X
               QUALIFY=SPEC,                                           X
               RPL=RPLA,                                               X
               AAREA=RPLAX,                                            X
               ACB=APPLA,                                              X
               CONVID=(8),                                             X
               AREA=RECAREA,                                           X
               AREALEN=255,                                            X
               FILL=LL,                                                X
               OPTCD=SYN
*
         LTR   15,15              * CHECK GENERAL RETURN CODE IN 15
         BNZ   BADGENRC           * HANDLE NONZERO RETURN CODE
         LTR   0,0                * CHECK CONDITIONAL COMPLETION
         BNZ   BADCOND            * HANDLE NONZERO RETURN CODE
*
* RETURN CODES WERE GOOD, NOW CHECK WHAT-RECEIVED INDICATOR FOR SEND.
*
         LA    9,RPLAX            * LOAD RPL EXTENSION ADDRESS
         USING ISTRPL6X,9         * ESTABLISH ADDRESSABILITY
         TM    RPL6RCV1,RPL6WSND  * CHECK SEND INDICATOR
*
* AT THIS POINT, IF THE INDICATOR BIT HAS BEEN SET APPLA KNOWS IT IS
* IN SEND STATE. IT MUST STILL HANDLE ANY DATA THAT WAS RECEIVED ON
* THE MACRO.
*
                  .
                  .
```

```
                    .
RECAREA  DS    XL255               * RECEIVE BUFFER
RPLA     RPL AM=VTAM               * RPL STORAGE
RPLAX    ISTRPL6                   * RPL EXTENSION STORAGE
APPLA    ACB AM=VTAM,MACRF=LOGON,APPLID=APPLNAME * ACB STORAGE
```

Sending normal information

Normal information includes normal data and normal indications. Normal information can be transmitted by the application program, and includes normal data such as logical records. Application programs cannot actually send normal indications. Normal indications are the result of macroinstructions or the result of information initiated by VTAM or intervening network components. See "Comparing normal information to expedited information" on page 60 for more information about normal and expedited information.

The following sections describe sending logical record data and indications as normal information.

Sending logical record data

Logical record data is usually of interest only to the transaction programs. It could, for example, consist of a bank balance inquiry and reply.

Logical record data is sent and the SEND buffer is flushed with the following APPCCMD macroinstructions:

- APPCCMD CONTROL=SEND, QUALIFY=DATACON
- APPCCMD CONTROL=SEND, QUALIFY=DATAFLU
- APPCCMD CONTROL=SENDRCV, QUALIFY=DATAFLU
- APPCCMD CONTROL=DEALLOC, QUALIFY=DATACON
- APPCCMD CONTROL=DEALLOC, QUALIFY=DATAFLU
- APPCCMD CONTROL=PREPRCV, QUALIFY=DATACON
- APPCCMD CONTROL=PREPRCV, QUALIFY=DATAFLU

The SEND buffer is also flushed with the following APPCCMD macroinstructions:

- APPCCMD CONTROL=SEND, QUALIFY=FLUSH
- APPCCMD CONTROL=DEALLOC, QUALIFY=CONFIRM
- APPCCMD CONTROL=DEALLOC, QUALIFY=FLUSH
- APPCCMD CONTROL=PREPRCV, QUALIFY=CONFIRM
- APPCCMD CONTROL=PREPRCV, QUALIFY=FLUSH

The DATAFLU and DATACON macroinstructions provide new data and, additionally, flush the SEND buffer. The DATACON macroinstruction also sends a confirmation request. The FLUSH macroinstruction does not provide new data to VTAM but causes VTAM to send buffered data to the partner LU.

When sending data, the application program sets the AREA field of the RPL to point to the conversation data and the RECLEN field to the length of the data being sent. (See "Buffer list requirements" on page 188 for details on using buffer lists to send data.)

Example of sending logical records

In this example, an application program known as APPLA sends two logical records using two macroinstructions. On the first macroinstruction, an entire logical record is sent, followed by the beginning of another. On the second macroinstruction, the rest of the logical record is sent. In addition, APPLA, at the same time, deallocates the conversation. The conversation identifier is stored in CONVERID.

```
*
* PUT FIRST LOGICAL RECORD AND PART OF SECOND IN THE SEND BUFFER
* MAINTAINED FOR THE CONVERSATION.  PUT ONLY 28 BYTES IN WITH THE
* CONTROL=SEND MACRO (COMPLETE 1ST RECORD AND FOUR BYTES OF
* THE SECOND RECORD).  START BY BUILDING RECORDS IN CONTIGUOUS STORAGE.
```

```
*
         MVC   MSGBUFF(2),=X'0018' * INITIALIZE 1ST HEADER
         MVC   MSGBUFF+2(22),=C'THIS IS THE 1ST RECORD'   * MOVE DATA
         MVC   MSGBUFF+24(2),=X'0019'                     * 2ND HEADER
         MVC   MSGBUFF+26(23),=C'THIS IS THE LAST RECORD' * MOVE DATA
         LA    10,MSGBUFF          * LOAD ADDRESS OF BUFFER AREA
*
         APPCCMD CONTROL=SEND,                                         X
               QUALIFY=DATA,                                           X
               RPL=RPLA,                                               X
               AAREA=RPLAX,                                            X
               ACB=APPLA,                                              X
               CONVID=CONVERID,                                        X
               AREA=(10),                                              X
               RECLEN=28,                                              X
               OPTCD=SYN
*
         LTR   15,15               * CHECK GENERAL RETURN CODE IN 15
         BNZ   BADGENRC            * HANDLE NONZERO RETURN CODE
         LTR   0,0                 * CHECK CONDITIONAL COMPLETION
         BNZ   BADCOND             * HANDLE NONZERO RETURN CODE
*
* RETURN CODES WERE GOOD. NOW FINISH SENDING SECOND RECORD AND
* IN ADDITION DEALLOCATE CONVERSATION.
*
         LA    10,MSGBUFF+28       * GET START OF REMAINING RECORD
*
         APPCCMD CONTROL=DEALLOC,                                      X
               QUALIFY=DATAFLU,                                        X
               RPL=RPLA,                                               X
               AAREA=RPLAX,                                            X
               ACB=APPLA,                                              X
               CONVID=CONVERID,                                        X
               AREA=(10),                                              X
               RECLEN=21,                                              X
               OPTCD=SYN
*
         LTR   15,15               * CHECK GENERAL RETURN CODE IN 15
         BNZ   BADGENRC            * HANDLE NONZERO RETURN CODE
         LTR   0,0                 * CHECK CONDITIONAL COMPLETION
         BNZ   BADCOND             * HANDLE NONZERO RETURN CODE
*
* RETURN CODES WERE GOOD, THE DATA HAS BEEN SENT AND THE CONVERSATION
* DEALLOCATED.
*
                  .
                  .
                  .
CONVERID DS    XL4                * CONVERSATION ID
MSGBUFF  DS    XL255              * STORAGE FOR DATA TO SEND
RPLA     RPL AM=VTAM              * RPL STORAGE
RPLAX    ISTRPL6                  * RPL EXTENSION STORAGE
APPLA    ACB AM=VTAM,MACRF=LOGON,APPLID=APPLNAME * ACB STORAGE
```

Getting better send and receive performance

Applications that perform frequent send and receive operations within a given conversation can combine these requests with the APPCCMD CONTROL=SENDRCV, QUALIFY=DATAFLU macroinstruction. Use of this macroinstruction reduces the send and receive pathlength and provides better performance for data transfers. For more information about this macroinstruction, refer to z/OS Communications Server: SNA Programmer's LU 6.2 Reference.

Sending confirmation requests

Application programs use confirmation requests and responses to synchronize processing with their conversation partners. Application programs send a confirmation request by specifying either QUALIFY=CONFIRM or QUALIFY=DATACON in the RPL. The list shows the APPCCMD macroinstructions that generate a confirmation request:

- APPCCMD CONTROL=DEALLOC, QUALIFY=CONFIRM
- APPCCMD CONTROL=DEALLOC, QUALIFY=DATACON
- APPCCMD CONTROL=PREPRCV, QUALIFY=CONFIRM
- APPCCMD CONTROL=PREPRCV, QUALIFY=DATACON

- APPCCMD CONTROL=SEND, QUALIFY=CONFIRM
- APPCCMD CONTROL=SEND, QUALIFY=DATACON

If the CONFIRM bit is set in the What-Received field in an APPCCMD CONTROL=RECEIVE macroinstruction, the application program must respond to the confirmation request before it can issue any other APPCCMD macroinstructions. Application programs should check for the CONFIRM setting before attempting any action based on the setting of the SEND or DEALLOCATE bits.

The application program can use confirmation requests for a variety of reasons. It can issue the macroinstruction after an allocation request to determine whether the allocation was successful before sending data, or it can issue the confirmation request as an acknowledgment of data sent to a conversation partner. Confirmation requests also cause VTAM to flush the SEND buffer.

The application program must ensure that a confirmation request is not issued on a conversation that was allocated with a synchronization level of "none." (The synchronization level is determined when the conversation is allocated by a field in the FMH-5.) If such a request is issued, a nonzero return code is received.

The application program does not have to supply any data to VTAM for a confirmation request. (On a QUALIFY=DATACON macroinstruction, however, the application program is sending data as well as a confirmation request, so data must be supplied.)

When the application program issues an APPCCMD macroinstruction that specifies a confirmation request, VTAM does not complete the macroinstruction until a response to the confirmation request is received from the partner application. The partner application can send a positive response to the confirmation request by issuing the APPCCMD CONTROL=SEND, QUALIFY=CONFRMD macroinstruction, which causes the application program's macroinstruction to complete with an RCPRI, RCSEC confirmation of OK. The partner application can send a negative response to the confirmation request by issuing APPCCMD CONTROL=SEND, QUALIFY=ERROR or one of the abnormal termination macroinstructions. VTAM interprets the FMH-7 received from the partner LU and completes the application program's macroinstruction with the appropriate RCPRI, RCSEC return codes.

If the application program receives a negative reply through a return code of USER_ERROR_CODE_RECEIVED__NEGATIVE_ RESPONSE (an RCPRI and RCSEC combination of X'005C' and X'0000', respectively), it is not immediately obvious whether the negative response was reported through the APPCCMD CONTROL=SEND, QUALIFY=ERROR macroinstruction or a DEALLOC macroinstruction. The application program must be able to determine this from the sense code returned in the SENSE field of the RPL. Alternatively, if the LOGRCV field indicates that error log data is being sent to the application program, the application program can receive the data. The APPCCMD CONTROL=RECEIVE macroinstruction that receives the error log data completes with a return code indicating the conversation has been deallocated, if the negative confirmation request was generated as part of a deallocation request.

The APPCCMD CONTROL=PREPRCV macroinstructions are somewhat different than the other macroinstructions. The application program can specify that the macroinstruction cannot complete execution until it has received data from the partner LU, instead of a positive response to the confirmation request. This is done by specifying LOCKS=LONG in the RPL.

The application program can be assured that data or control information is available to complete an APPCCMD CONTROL=RECEIVE, QUALIFY=SPEC|ISPEC macroinstruction when a LOCKS=LONG PREPRCV request completes. This allows the application program to avoid tying up buffer space waiting for an APPCCMD CONTROL=RECEIVE, QUALIFY=SPEC to complete, because this request cannot complete until something is available to receive on the conversation.

Example of a confirmation request

In this example, an application program known as APPLA requests a confirmation from a partner on a conversation whose identifier is in register 8. No data is included on the request.

```
*
         APPCCMD CONTROL=SEND,                                         X
```

```
               QUALIFY=CONFIRM,                                        X
               RPL=RPLA,                                               X
               AAREA=RPLAX,                                            X
               ACB=APPLA,                                              X
               CONVID=(8),                                             X
               OPTCD=SYN
*
         LTR   15,15              * CHECK GENERAL RETURN CODE IN 15
         BNZ   BADGENRC           * HANDLE NONZERO RETURN CODE
         LTR   0,0                * CHECK CONDITIONAL COMPLETION
         BNZ   BADCOND            * HANDLE NONZERO RETURN CODE
*
* RETURN CODES WERE ZERO. THE CONFIRMATION RESPONSE WAS POSITIVE.
*
                   .
                   .
                   .
RPLA     RPL AM=VTAM              * RPL STORAGE
RPLAX    ISTRPL6                  * RPL EXTENSION STORAGE
APPLA    ACB AM=VTAM,MACRF=LOGON,APPLID=APPLNAME * ACB STORAGE
```

Sending expedited information

Expedited data can be sent by an application program in any conversation state except PENDING_ALLOCATE, PENDING_DEALLOCATE or END_CONVERSATION. A request-to-send, which is expedited information, can be issued by an application program in any conversation state except:

- PENDING_DEALLOCATE
- PENDING_END_CONVERSATION_LOG
- PENDING_ALLOCATE
- END_CONVERSATION
- PENDING_RECEIVE_LOG

See “Comparing normal information to expedited information” on page 60 for more information about normal and expedited information.

The APPCCMD CONTROL=SENDEXPD, QUALIFY=DATA macroinstruction is used to send expedited data to a partner application over a conversation allocated on a full-duplex-capable session. The amount of expedited data specified by the application should be in the range of 1 - 86 bytes.

The APPCCMD CONTROL=SENDEXPD, QUALIFY=DATA macroinstruction causes expedited data to be sent to the partner. This macroinstruction is posted complete immediately without waiting for a response from the partner LU. If the macroinstruction is issued and the response to a previous SENDEXPD request has not been received, a nonzero return code is returned to the application. The response from the partner LU is not sent until the expedited data has been received by the partner application.

For more information, refer to z/OS Communications Server: SNA Programmer's LU 6.2 Reference.

The APPCCMD CONTROL=SEND, QUALIFY=RQSEND macroinstruction causes a request-to-send indicator to be sent to the partner. This macroinstruction is posted complete after a response has been received from the partner LU. If the macroinstruction is issued and the response to a previously issued APPCCMD CONTROL=SENDEXPD request has not been received, a nonzero return code is returned to the application. For more information, refer to z/OS Communications Server: SNA Programmer's LU 6.2 Reference.

When sending expedited data, ensure that the partner transaction program supports the receipt of expedited data. In some circumstances, the APPCCMD CONTROL=SENDEXPD, QUALIFY=DATA macroinstruction completes successfully even when expedited data does not reach the partner. This can occur when a half-duplex transaction program has allocated a conversation on a full-duplex session and the partner transaction program does not support the receipt of expedited data.

For example, assume there are two existing half-duplex transaction programs. If the LUs upon which the transaction programs operate are upgraded to support full-duplex, then the sessions between the half-duplex transaction programs may begin to have full-duplex capability. Now assume one of the transaction programs is upgraded to exploit the sending of expedited data, which is allowed on a half-

duplex conversation on a full-duplex session. If the other transaction program is not upgraded at the same time, a problem can occur.

The problem occurs when the upgraded transaction program sends expedited data to the partner. The first time the macroinstruction completes successfully, however, the downlevel partner will never receive the expedited data.

If the upgraded transaction program sends more expedited data to the partner, then that macroinstruction completes with an RCPRI, RCSEC of X'00A0', X'0005', REQUEST_NOT_ALLOWED—RSP_TO_PREVIOUS_SENDEXP_REQUEST_NOT_RECEIVED.

Sending information on full-duplex conversations

This section explains the process and APPCCMD macroinstructions involved in sending information on full-duplex conversations. For details on half-duplex conversations, see "Sending information on half-duplex conversations" on page 177.

Application programs can send information from the following states:

- SEND/RECEIVE
- SEND_ONLY
- PENDING_SEND/RECEIVE_LOG

Each of these states is described in the following sections.

The APPCCMD CONTROL=SENDRCV, QUALIFY=DATAFLU macroinstruction is not allowed on full-duplex conversations.

Because sending information involves the use of buffers, a discussion of buffers is included.

Use of the SEND buffer

The SEND buffer is used to accumulate only normal, not expedited, data.

Issuing an APPCCMD CONTROL=SEND macroinstruction to send data does not automatically cause data to flow through the network. Instead, VTAM stores the data in a buffer in VTAM storage until enough is sent to exceed the maximum SEND RU size that was specified in the session parameters used to establish the conversation's underlying session.

The SEND buffer has nothing to do with logical record size. If the buffer fills up before a logical record is completely stored in it, VTAM sends what is in the buffer and stores the remainder of the logical record in the buffer. VTAM maintains the record's integrity throughout the process.

FMH-5 headers pointed to by the APPCCMD CONTROL=ALLOC and APPCCMD CONTROL=SENDFM5 macroinstructions and FMH-7 headers created by the APPCCMD CONTROL=SEND, QUALIFY=ERROR macroinstruction are sent immediately to the partner transaction program.

Flushing the buffer

Requesting VTAM to transmit buffered data is called flushing the SEND buffer. Many of the APPCCMD macroinstructions instruct VTAM to flush the SEND buffer. All macroinstructions with a QUALIFY value of FLUSH or DATAFLU flush the buffer.

The APPCCMD CONTROL=SEND, QUALIFY=ERROR macroinstruction and the abnormal termination DEALLOC and DEALLOCQ APPCCMD macroinstructions also flush the buffer. The APPCCMD CONTROL=SEND, QUALIFY=ERROR macroinstruction flushes the buffer and sends the FMH-7 on full-duplex conversations.

The only macroinstruction that sends normal information but does not flush the SEND buffer is the APPCCMD CONTROL=SEND, QUALIFY=DATA macroinstruction.

The buffering of data allows the application program to issue a macroinstruction or several macroinstructions before its partner receives anything. Consequently, the conversation partner frequently

is unable to detect errors or give other feedback immediately after an APPCCMD macroinstruction is used to send information. The application program can flush the buffer to ensure that the partner receives the information in a timely manner.

Roles of sender and receiver

On full-duplex conversations, both partners can be in SEND/RECEIVE state and send and receive information concurrently. Unlike half-duplex conversations, no conversation partner has to request a change in state to send or receive information. Both conversations partners have the ability to send and receive information.

An exception is when a conversation partner normally deallocates a conversation. The conversation partner deallocating the conversation enters RECEIVE_ONLY state after initiating the deallocation, and the partner receiving the deallocation request enters SEND_ONLY state. At this point, the partner in RECEIVE_ONLY state can only receive information and the partner in SEND_ONLY state can only send information.

Entering SEND/RECEIVE state

Full-duplex conversations allow for both partners to send and receive information concurrently. The transaction program initiating the full-duplex conversation enters SEND/RECEIVE state after successful allocation. The partner transaction program receives the FMH-5 and enters SEND/RECEIVE state also. Either side of a full-duplex conversation in SEND/RECEIVE state can send information.

In general, the conversation is in SEND/RECEIVE state after successful completion of:

- An APPCCMD CONTROL=ALLOC macroinstruction
- An APPCCMD CONTROL=RCVFMH5 macroinstruction
- An APPCCMD CONTROL=SENDFM5 macroinstruction
- An APPCCMD CONTROL=RECEIVE, QUALIFY=SPEC|ISPEC macroinstruction and the log data indicator in the What-Received field is on, the data or data-complete indicator in the What-Received field is on, and the deallocate indicator in the What-Received field is off

Entering SEND_ONLY state

The conversation enters SEND_ONLY state when an APPCCMD CONTROL=RECEIVE macroinstruction completes with the deallocate indicator in the What-Received field on. This indicates that the partner application has issued an APPCCMD CONTROL=DEALLOC, QUALIFY=FLUSH|DATAFLU. In this state, the application can continue to send information until ready to deallocate the conversation.

In general, the conversation is in SEND_ONLY state after successful completion of an APPCCMD CONTROL=RECEIVE macroinstruction and the deallocate indicator in the What-Received field is on.

Entering PENDING_SEND/RECEIVE_LOG state

The local conversation enters PENDING_SEND/RECEIVE_LOG state after receiving notification that the partner application has issued an APPCCMD CONTROL=SEND, QUALIFY=ERROR macroinstruction. In general, the conversation is in PENDING_SEND/RECEIVE_LOG state after the completion of an APPCCMD CONTROL=RECEIVE macroinstruction whose RCPRI value indicates a nonterminating error and whose LOGRCV field indicates log data follows.

Sending normal information

Normal information includes normal data and normal indications. Normal information can be transmitted by the application program, and includes normal data such as logical records. Application programs cannot actually send normal indications. Normal indications are the result of macroinstructions or the result of information initiated by VTAM or intervening network components. See “Comparing normal information to expedited information” on page 60 for more information about normal and expedited information.

The following sections describe sending logical record data and indications as normal information.

Sending logical record data

Logical record data is usually of interest only to transaction programs. It could, for example, consist of a bank balance inquiry and reply.

Logical record data is sent with the following macroinstructions:

- APPCCMD CONTROL=SEND, QUALIFY=DATA
- APPCCMD CONTROL=SEND, QUALIFY=DATAFLU
- APPCCMD CONTROL=DEALLOC, QUALIFY=DATAFLU

The send buffer is always flushed with the following macroinstructions:

- APPCCMD CONTROL=SEND, QUALIFY=DATAFLU|FLUSH
- APPCCMD CONTROL=DEALLOC, QUALIFY=DATAFLU
- APPCCMD CONTROL=DEALLOC, QUALIFY=FLUSH

The APPCCMD CONTROL=SEND, QUALIFY=DATA macroinstruction provides new data. DATAFLU and FLUSH cause VTAM to send buffered data to the partner LU.

When sending logical record data, the application program sets the AREA field of the RPL to point to the conversation information and the RECLEN field to the length of data being sent. See "Buffer list requirements" on page 188 for details on using buffer lists to send data.

Sending expedited information

Expedited information can be sent by an application program in any conversation state except FDX_RESET and PENDING_ALLOCATE.

The APPCCMD CONTROL=SENDEXPD, QUALIFY=DATA macroinstruction is used to send expedited data to a partner LU. The amount of expedited data specified by the application should be in the range of 1 - 86 bytes.

This macroinstruction is posted complete immediately without waiting for a response from a partner LU. This response is not sent by the partner until the expedited data has been received by the partner application.

For more information, refer to z/OS Communications Server: SNA Programmer's LU 6.2 Reference.

Buffer list requirements

VTAM enables application programs to chain together data from discontiguous storage locations and send it as one piece of data. This is done by setting the OPTCD field in the RPL to include BUFFLST or XBUFLST, or by coding OPTCD=BUFFLST or OPTCD=XBUFLST on the SEND macroinstruction. The XBUFLST option specifies an *extended buffer list*. See Chapter 11, "Sending and receiving data using high performance data transfer," on page 217 for more information.

When the BUFFLST option is used, the AREA field in the RPL no longer points to the data to be sent. Instead, it points to an area of storage containing a buffer list. The list is made up of 16-byte entries, aligned on doubleword boundaries, which contain pointers to the actual data to be sent and the length of each data item. Entries in the list, which is in contiguous storage, are processed sequentially.

The first 8 bytes of each entry are reserved and should be set to 0. The next 4 bytes contain the pointer to the data and the last 4 bytes give the length of the data.

The buffer list option has no effect on the receiving side of the conversation. The use of the buffer list option is transparent to the receiving side.

Example of using a buffer list

Take as an example an application program sending the 27-byte character string:

```
0027THIS IS A BUFFLST EXAMPLE
```

Suppose the string was actually stored in four separate storage areas referenced as LENGTH, BEGIN, MIDDLE, and END and that the storage areas contained, respectively:

```
0027 THIS IS A BUFFLST EXAMPLE
```

The length of the data in LENGTH is 2 bytes, the length of the data in BEGIN is 15 bytes, MIDDLE is 7 bytes, and END is 3 bytes. Sending this string using the BUFFLST option requires a buffer list of four entries—one for each of the storage areas. The AREA field of the RPL points to a 64-byte area of storage containing the entries and the RECLEN field of the RPL has a value of 64. The buffer list would be:

Bytes 0–15
Buffer list entry for LENGTH

Bytes 16–31
Buffer list entry for BEGIN

Bytes 32–47
Buffer list entry for MIDDLE

Bytes 48–63
Buffer list entry for END

Table 33 on page 189 shows what the entries would contain.

Table 33. Sample buffer list contents

Entry	Bytes	Contents
Entry for LENGTH	0–7	Reserved
	8–11	Address of LENGTH area
	12–15	X'0000002'
Entry for BEGIN	16–23	Reserved
	24–27	Address of BEGIN area
	28–31	X'000000F'
Entry for MIDDLE	32–39	Reserved
	40–43	Address of MIDDLE area
	44–47	X'00000007'
Entry for END	48–55	Reserved
	56–59	Address of END area
	60–63	X'00000003'

The following code enables an application program known as APPLA to build and send the buffer list. The conversation identifier is in register 8. This example uses the DSECT ISTBLENT, which maps buffer list entries. For a description of ISTBLENT, refer to z/OS Communications Server: SNA Programming.

```
MVC   LENGTH(2),=X'001B'            * MOVE LENGTH INTO LENGTH FIELD
      MVC   BEGIN(15),=C'THIS IS A BUFFL' * MOVE DATA INTO BEGIN
      MVC   MIDDLE(7),=C'ST EXAM'         * MOVE DATA INTO MIDDLE
      MVC   END(3),=C'PLE'                * MODE DATA INTO END
      LA    10,LISTSTOR                   * GET BUFFER LIST ADDRESS
      USING ISTBLENT,10                   * ESTABLISH ADDRESSABILITY
      LA    7,X'0'                        * SET COUNTER TO 1ST TIME
LOOP  MVC   BLENT(8),=X'0000000000000000' * ZERO OUT RESERVED HEADER
```

```
ZERO     CL    7,=X'00000000'               * ZEROTH TIME THROUGH?
         BH    FIRST                        * GO TO 1ST ENTRY IF NOT
         LA    9,LENGTH                     * GET BEGIN ADDRESS
         ST    9,BLEAREA                    * STORE IN LIST ENTRY
         MVC   BLERLEN,=X'00000002'         * MOVE IN AREA LENGTH
         B     MOVEPTR                      * MOVE PTR TO SECOND ENTRY
FIRST    CL    7,=X'00000001'               * FIRST TIME THROUGH?
         BH    SECOND                       * GO TO NEXT ENTRY IF NOT
         LA    9,BEGIN                      * GET BEGIN ADDRESS
         ST    9,BLEAREA                    * STORE IN LIST ENTRY
         MVC   BLERLEN,=X'0000000F'         * MOVE IN AREA LENGTH
         B     MOVEPTR                      * MOVE PTR TO SECOND ENTRY
SECOND   CL    7,=X'00000002'               * SECOND TIME THROUGH?
         BH    THIRD                        * GO TO LAST ENTRY IF NOT
         LA    9,MIDDLE                     * GET MIDDLE ADDRESS
         ST    9,BLEAREA                    * STORE IN LIST ENTRY
         MVC   BLERLEN,=X'00000007'         * MOVE IN AREA LENGTH
         B     MOVEPTR                      * MOVE PTR TO LAST ENTRY
THIRD    LA    9,END                        * GET END ADDRESS
         ST    9,BLEAREA                    * STORE IN LIST ENTRY
         MVC   BLERLEN,=X'00000003'         * MOVE IN AREA LENGTH
         B     SENDMAC                      * DROP OUT OF LOOP
MOVEPTR  LA    10,16(10)                    * MOVE TO NEXT ENTRY
         AH    7,=X'0001'                   * INCREMENT COUNT
         B     LOOP                         * HANDLE NEXT ENTRY
*
SENDMAC  APPCCMD CONTROL=SEND,                                          X
               QUALIFY=DATAFLU,                                         X
               RPL=RPLA,                                                X
               ACB=APPLA,                                               X
               CONVID=(8),                                              X
               AREA=LISTSTOR,                                           X
               RECLEN=64,                                               X
               OPTCD=(SYN,BUFFLST)
*
         LTR   15,15               * CHECK GENERAL RETURN CODE IN 15
         BNZ   BADGENRC            * HANDLE NONZERO RETURN CODE
         LTR   0,0                 * CHECK CONDITIONAL COMPLETION
         BNZ   BADCOND             * HANDLE NONZERO RETURN CODE
*
* RETURN CODES WERE GOOD. THE BUFFER LIST HAS BEEN SENT.
*
                   .
                   .
                   .
LENGTH   DS    XL255               * STORAGE FOR LENGTH FIELD
BEGIN    DS    XL255               * STORAGE FOR DATA
MIDDLE   DS    XL255               * STORAGE FOR DATA
END      DS    XL255               * STORAGE FOR DATA
LISTSTOR DS    XL255               * STORAGE FOR BUFFER LIST
RPLA     RPL AM=VTAM               * RPL STORAGE
APPLA    ACB AM=VTAM,MACRF=LOGON,APPLID=APPLNAME * ACB STORAGE
```

BUFFLST differences for LU 6.2

The use of the BUFFLST option for LU 6.2 functions is somewhat different than for non-APPCCMD SEND macroinstructions. Application programs cannot supply a request header (RH) to be associated with the data. VTAM creates the RH using information in the RPL fields. In addition, application programs cannot use the LMPEO function in VTAM to control the splitting or segmenting of data within or across SNA request units. VTAM determines where RU boundaries fall within the data and when a current RU ends and a new RU begins. Application programs can, however, control data within RUs in an indirect way by using the APPCCMD CONTROL=SEND, QUALIFY=FLUSH macroinstruction to force VTAM to send data out. This is, however, independent of whether the application program uses the BUFFLST function.

Handling storage shortages

An APPCCMD that sends data can fail because of a temporary storage shortage while VTAM is attempting to send the data. This is indicated by a return code of X'98' in the RCPRI. The unsuccessful return code does not mean that no data was sent. Some of the data supplied by the application program might have been processed before the shortage occurred.

When this error is reported, VTAM reports to the application program how much data, if any, was processed. It does so in two RPL extension fields—RPL6STBF and RPL6STDS. RPL6STBF points to

the application-supplied area that is being processed. If no buffer list is being used on the send, it corresponds to the value supplied by the application program in the AREA field of the RPL. If a buffer list is being used, it points to the actual buffer being processed. Consequently, the application program must be able to take this address and examine its buffer list to determine how many entries have been processed. The RPL6STDS field gives the displacement into the buffer of the data that has been processed. If RPL6STDS is 0, no data has been processed in the buffer.

The application program can issue another send request to process the remaining data.

Send requests not using a buffer list

For send requests not using a buffer list (NBUFFLST), the proper address for the AREA field can be computed by adding RPL6STBF and RPL6STDS. The new RECLEN value can be determined by subtracting RPL6STDS from the RECLEN value.

RECLEN
Indicates the total size of the buffer.

RPLAREA
Indicates the starting point of the buffer.

RPL6STBF
Indicates the starting point of the data that was being sent.

RPL6STDS
Indicates the point at which the storage shortage occurred and, therefore, the starting point for the continuation of the send.

Use the following sequence:

1. RPLAREA = RPL6STBF + RPL6STDS
2. RECLEN = RECLEN - RPL6STDS

Send requests using a buffer list

If a buffer list (BUFFLST) is being used, the application program must modify the APPCCMD AREA field to point to the entry that was being processed at the time of the shortage. That entry might need to be modified so that the address field points to RPL6STBF plus RPL6STDS, with the length field in the entry modified to reflect the amount of data remaining.

RPLAREA
Indicates the beginning of the buffer list.

RECLEN
Indicates the size of the buffer list.

RPL6STBF
Indicates the address of the current buffer.

RPL6STDS
Indicates the displacement into the current buffer.

To continue processing the data:

1. Point RPLAREA to the entry that contains the current buffer address.

```
Save "old" RPLAREA value
Find buffer list entry where (BLEAREA = RPL6STBF)
RPLAREA = address of found buffer list entry
```

2. Decrement RECLEN by the number of entries preceding the entry containing the current buffer address.

```
RPLRLEN = RPLRLEN - (RPLAREA - saved
"old" RPLAREA value)
```

3. Update the "new" first entry in the updated buffer list so that the pointer to the buffer is equal to the current pointer plus RPL6STDS.

```
BLEAREA = BLEAREA + RPL6STDS
```

4. Update the length field in the first buffer list entry so that the length is equal to the current length minus RPL6STDS.

```
BLERLEN = BLERLEN - RPL6STDS
```

Use the APPCCMD CONTROL=SEND, QUALIFY=DATA macroinstruction with the AREA and RECLEN fields updated appropriately.

If the APPCCMD was one of the APPCCMD CONTROL=DEALLOC macroinstructions for abnormal terminations, use the APPCCMD CONTROL=DEALLOC, QUALIFY=DATAFLU macroinstruction with the RPL fields updated. Or deallocate the conversation with one of the abnormal deallocation macroinstructions or the APPCCMD CONTROL=REJECT, QUALIFY=CONV macroinstruction.

Note: A logical record length error can result if the application program issues an APPCCMD other than the macroinstructions suggested.

Chapter 10. Receiving information

About this chapter

This chapter discusses the tasks an application program must perform and the conversation states necessary to receive information on a conversation over the VTAM API. Replying to a confirmation request is also included for half-duplex conversations, because even though they actually involve sending information, confirmation replies always come from the receiving side of a conversation.

This chapter contains information about VTAM's normal receive services and interface. Better receive performance is available to applications that use the interface for high performance data transfer (HPDT). For information about HPDT, see Chapter 11, “Sending and receiving data using high performance data transfer,” on page 217. Before reading that chapter, however, the reader should be familiar with the information presented in this chapter.

Four APPCCMD macroinstruction variations can be used to receive normal information:

- APPCCMD CONTROL=RECEIVE, QUALIFY=ANY
- APPCCMD CONTROL=RECEIVE, QUALIFY=IANY
- APPCCMD CONTROL=RECEIVE, QUALIFY=SPEC
- APPCCMD CONTROL=RECEIVE, QUALIFY=ISPEC

Four APPCCMD macroinstruction variations can be used to receive expedited information:

- APPCCMD CONTROL=RCVEXPD, QUALIFY=SPEC
- APPCCMD CONTROL=RCVEXPD, QUALIFY=ISPEC
- APPCCMD CONTROL=RCVEXPD, QUALIFY=ANY
- APPCCMD CONTROL=RCVEXPD, QUALIFY=IANY

Each of these macroinstructions is described in the following sections.

Determining what is received

Application programs can receive several types of information. For a description of the types of information that can be exchanged, see “Comparing data to indications” on page 60.

To determine what is received by your application program, check the What-Received field in the RPL extension. This field describes the type of information received by the application program. The What-Received field in the RPL extension is referenced by the labels RPL6RCV1 and RPL6RCV2 in the ISTRPL6X DSECT. (For the complete layout of the ISTRPL6X DSECT, refer to z/OS Communications Server: SNA Programmer's LU 6.2 Reference.)

Note: The What-Received field is applicable only to the APPCCMD CONTROL=RECEIVE macroinstruction.

What-received field

The What-Received field should be examined only when the RCPRI field of the RPL extension indicates a return code of 0 on an APPCCMD CONTROL=RECEIVE. Otherwise, the What-Received field has no meaning.

VTAM maintains information and passes this information to the application program in the sequence in which it is received. VTAM passes to the application program any indicators for events only when the application must react or take action as noted by the indicator. For example, if the DEALLOCATE indicator is received along with other data, VTAM sets the DEALLOCATE indicator in the What-Received field only after passing all data to the application program. Some indicators are mutually exclusive, such as DATA, DATA_COMPLETE, and DATA_INCOMPLETE.

The 2-byte WHATRCV field has the format shown in Table 34 on page 194 (bit 0 is the most significant bit; it is the bit indicated by a value of X'8000').

Table 34. Description of bits in WHATRCV field

RPL6RCV1		RPL6RCV2	
Bit	**Meaning**	**Bit**	**Meaning**
0	DATA	0	PARTIAL_PS_HEADER
1	DATA_COMPLETE	1–7	Reserved
2	DATA_INCOMPLETE		
3	SEND		
4	CONFIRM		
5	DEALLOCATE		
6	LOG_DATA		
7	PS_HEADER		

The meanings of the bits are as follows:

DATA
Indicates that the application program received data independent of the data's logical record format. This value is returned to the application program only if the application program issued an APPCCMD CONTROL=RECEIVE, QUALIFY=SPEC|ISPEC, FILL=BUFF, or if the application program issued APPCCMD CONTROL=RECEIVE, QUALIFY=ANY|IANY and specified CONMODE=BUFFCA on a prior APPCCMD.

DATA_COMPLETE
Indicates that the application program received a complete logical record, or the last remaining portion of a record. This value is returned to the application program only if the application program issued APPCCMD CONTROL=RECEIVE, QUALIFY=SPEC|ISPEC, FILL=LL, or the application program issued APPCCMD CONTROL=RECEIVE, QUALIFY=ANY|IANY and specified CONMODE=LLCA on a prior APPCCMD.

DATA_INCOMPLETE
Indicates that the application program received less than a complete logical record. This situation occurs when the AREALEN value specified in the RPL is less than the length of the logical record, or when the macroinstruction completes before the entire logical record arrives. The application program issues one or more APPCCMD CONTROL=RECEIVE macroinstructions to receive the remainder of the data. This value is returned to the application program only if the application program issued APPCCMD CONTROL=RECEIVE, QUALIFY=SPEC|ISPEC, FILL=LL, or if the application program issued APPCCMD CONTROL=RECEIVE, QUALIFY=ANY|IANY and specified CONMODE=LLCA on a prior APPCCMD.

SEND
Indicates that the application program received an indication that the partner application has entered RECEIVE state, placing the local application program in SEND state. This situation never occurs on a full-duplex conversation.

CONFIRM
Indicates that the application program received information specifying that the partner LU has sent a confirmation request. The application program responds positively by issuing APPCCMD CONTROL=SEND, QUALIFY=CONFRMD. The application program can respond negatively by issuing APPCCMD CONTROL=SEND, QUALIFY=ERROR. This situation never occurs on a full-duplex conversation.

DEALLOCATE
Indicates that the conversation partner has deallocated the conversation. If the CONFIRM field in the what-received field is also set *on*, the partner transaction program has requested a conditional

deallocation of the half-duplex conversation, contingent upon the local application's reply; otherwise, the partner has unconditionally deallocated the conversation.

LOG_DATA
Indicates that the application program is receiving error log data from the conversation partner.

PS_HEADER
Indicates that the application program on a half-duplex conversation issued a receive and the receive data buffer is larger than the PS header length plus 2. PS headers consist of 2 bytes, which contain a constant value of X'0001', followed by a 1-byte field that contains the PS header length, followed by data. This situation never occurs on a full-duplex conversation. This bit is used only for sync point conversations.

PARTIAL_PS_HEADER
Indicates that the application program on a half-duplex-capable conversation received a data buffer smaller than the PS header length plus 2. This situation never occurs on a full-duplex conversation. This bit is used only for sync point conversations.

The CONFIRM, SEND, and DEALLOCATE indicators all indicate state changes. The application program must be able to properly process all valid combinations of these indicators. (For a list of valid combinations, see Table 35 on page 195.) The application program should check all of these bits after a RECEIVE. If sync point is supported, the application program should also check for the PS_HEADER and PARTIAL_PS_HEADER. If the application program had the FILL field in the RPL set to BUFF, only the DATA indicator needs to be checked. If FILL specified LL, either DATA_COMPLETE or DATA_INCOMPLETE can be set.

What-received indicators

Table 35 on page 195 shows the combinations of WHATRCV information that can be passed to the application program on one APPCCMD CONTROL=RECEIVE macroinstruction. The table also shows the valid conversation types for each combination.

Table 35. Valid combinations of what-received indicators

Returned Information	Valid Conversation Type	Meaning
DATA only	Full-duplex and Half-duplex	This value is returned when the application program is receiving data in terms of buffers instead of logical records. It indicates that data has been received. More data might remain in the logical record.
DATA and PS_HEADER	Half-duplex	These values are returned when the application program is receiving data in terms of buffers instead of logical records. It indicates that data and a complete PS header have been received. (The received data buffer is larger than or equal to any data preceding the PS header in the normal data queue plus the PS header.) The PS header is the last data in the buffer. The buffer might not be filled upon completion.
PS_HEADER	Half-duplex	This value indicates that a complete PS header has been received. (The received data buffer is larger than or equal to the PS header.) The PS header is the only data in the buffer. The buffer might not be filled upon completion.
DATA and PARTIAL_PS_HEADER	Half-duplex	These values are returned when the application program is receiving data in terms of buffers instead of logical records. It indicates that data and a portion of a PS header have been received. More data remains to be received to complete the PS header.

Table 35. Valid combinations of what-received indicators (continued)

Returned Information	Valid Conversation Type	Meaning
PARTIAL_PS_HEADER	Half-duplex	This value indicates that a portion of a PS header has been received. More data remains to be received to complete the PS header.
DATA_COMPLETE only	Full-duplex and Half-duplex	This value is returned when the application program is receiving data in terms of logical records, and it finishes receiving a logical record.
DATA_INCOMPLETE only	Full-duplex and Half-duplex	This value is returned when the application program is receiving data in terms of logical records, and a portion of a logical record is received. More data remains to be received to complete the logical record.
SEND only	Half-duplex	This value indicates that the partner LU has entered RECEIVE state, placing the local application program in SEND state. The local application program can now send conversation data.
SEND and PS_HEADER	Half-duplex	These values indicate that the partner LU has entered RECEIVE state, placing the local application program in SEND state. The local application program can now send conversation data. A complete PS header has also been received by the application program.
CONFIRM only	Half-duplex	This value indicates that the application program has received a confirmation request. The application program can respond positively to the confirmation request or report an error.
DEALLOCATE only	Full-duplex and Half-duplex	This value indicates that the partner LU has unconditionally deallocated the conversation.
DEALLOCATE and PS_HEADER	Half-duplex	These values indicate that the application has received a complete PS header and indicates that the partner LU has unconditionally deallocated the conversation.
DATA_INCOMPLETE and LOG_DATA	Full-duplex and Half-duplex	These values are returned when the application program is receiving data in terms of logical records, and a portion of an error log variable is received. More log data remains to be received to complete the error log variable, which is also a logical record.
CONFIRM and SEND	Half-duplex	These values indicate the partner LU will enter RECEIVE state after receiving a positive reply to a confirmation request. The application program can respond positively to the confirmation request or report an error.
CONFIRM and DEALLOCATE	Half-duplex	These values indicate that the partner LU will unconditionally deallocate the conversation after receiving a positive reply to a confirmation request. The application program is to respond either positively or negatively to the confirmation request.
DATA and SEND	Half-duplex	These values indicate that the application program is receiving data in terms of buffers rather than logical records, and has received data along with an indication that the partner LU has entered RECEIVE state. The received data should complete a logical record.

Table 35. Valid combinations of what-received indicators (continued)

Returned Information	Valid Conversation Type	Meaning
DATA, SEND, and PS_HEADER	Half-duplex	These values indicate that the application program is receiving data in terms of buffers rather than logical records, and an indication that the partner LU has entered RECEIVE state. The received data should complete a logical record, and the PS header is the last data in the buffer. The buffer might not be filled upon completion.
DATA and CONFIRM	Half-duplex	These values indicate that the application program is receiving data in terms of buffers rather than logical records, and has received data along with a confirmation request. The data must complete a logical record. The application program is to respond to the confirmation request, either positively or negatively.
DATA, CONFIRM, and PS_HEADER	Half-duplex	These values indicate that the application program is receiving data in terms of buffers rather than logical records, and a confirmation request. The data must complete a logical record, and the PS header is the last data in the buffer. The application program is to respond to the confirmation request, either positively or negatively.
CONFIRM and PS_HEADER	Half-duplex	These values indicate that the application program is receiving a complete PS header and a confirmation request. The PS header is the only data in the buffer. The application program is to respond to the confirmation request, either positively or negatively.
DATA and DEALLOCATE	Full-duplex and Half-duplex	These values indicate that the application program is receiving data in terms of buffers rather than logical records, and has received data along with an indication that the partner LU has unconditionally deallocated the conversation.
DATA, DEALLOCATE, and PS_HEADER	Half-duplex	These values indicate that the application program is receiving data in terms of buffers rather than logical records, has received a complete PS header, and an indication that the partner LU has unconditionally deallocated the conversation. The PS header is the last data in the buffer. The buffer might not be filled upon completion.
DEALLOCATE and PS_HEADER	Half-duplex	These values indicate that the application program has received a complete PS header and an indication that the partner LU has unconditionally deallocated the conversation. The PS header is the only data in the buffer. The buffer might not be filled upon completion.
DATA, CONFIRM, and SEND	Half-duplex	These values indicate that the application program is receiving data in terms of buffers rather than logical records. Indicates that the partner LU will enter RECEIVE state after receiving a positive reply to a confirmation request. The application program can respond positively to the confirmation request, or report an error.

Table 35. Valid combinations of what-received indicators (continued)

Returned Information	Valid Conversation Type	Meaning
DATA, CONFIRM, SEND, and PS_HEADER	Half-duplex	These values indicate that the application program is receiving data in terms of buffers rather than logical records, and has received a complete PS header, a confirmation request, and an indicator that the partner LU will enter RECEIVE state after receiving a positive reply to a confirmation request. The application program can respond positively to the confirmation request, or report an error. The PS header is the only data in the buffer. The buffer might not be filled upon completion.
CONFIRM, SEND, and PS_HEADER	Half-duplex	These values indicate that the application program has received a complete PS header a confirmation request, and an indicator that the partner LU will enter RECEIVE state after receiving a positive reply to a confirmation request. The application program can respond positively to the confirmation request, or report an error. The PS header is the only data in the buffer. The buffer might not be filled upon completion.
DATA, CONFIRM, and DEALLOCATE	Half-duplex	These values indicate that the application program is receiving data in terms of buffers rather than logical records. The application program has received data, along with a confirmation request and an indicator that the partner LU is seeking to deallocate the conversation. The application program is to respond to the confirmation request.
DATA, CONFIRM, DEALLOCATE, and PS_HEADER	Half-duplex	These values indicate that the application program is receiving data in terms of buffers rather than logical records. The application program has received data and a complete PS header. (The received data buffer is larger than or equal to any data preceding the PS header in the normal data queue plus the PS header.) The PS header is the last data in the buffer. The buffer might not be filled upon completion. The application program has also received a confirmation request and an indication that the partner LU will unconditionally deallocate the conversation after receiving a positive reply to the confirmation request. The application program is to respond to the confirmation request.
CONFIRM, DEALLOCATE, and PS_HEADER	Half-duplex	These values indicate that the application program is receiving a complete PS header. (The received data buffer is larger than or equal to the PS header.) The PS header is the only data in the buffer. The buffer might not be filled upon completion. The application program has also received a confirmation request and an indication that the partner LU will unconditionally deallocate the conversation after receiving a positive reply to the confirmation request. The application program is to respond to the confirmation request.
DATA and LOG_DATA	Full-duplex and Half-duplex	These values indicate that the application program is receiving data in terms of buffers rather than logical records. The application program is receiving only error log data.

Table 35. Valid combinations of what-received indicators (continued)

Returned Information	Valid Conversation Type	Meaning
DATA, LOG_DATA, and CONFIRM	Half-duplex	These values indicate that the application program is receiving data in terms of buffers rather than logical records. The application program has received data, an indication that the data received is error log data, and a confirmation request. The data must complete the error log variable. The application program is to respond to the confirmation request.
DATA, CONFIRM, SEND, and LOG_DATA	Half-duplex	These values indicate that the application program is receiving data in terms of buffers rather than logical records. They indicate that the application program has received data, a confirmation request, an indicator that the partner LU is seeking to enter RECEIVE state, and an indicator that the received data was error log data. The application program is to respond to the confirmation request.
DATA, CONFIRM, DEALLOCATE, and LOG_DATA	Half-duplex	These values indicate that the application program is receiving data in terms of buffers rather than logical records. The application program has received data, a confirmation request, an indicator that the partner LU is seeking to deallocate the conversation, and an indicator that the received data was error log data. The application program is to respond to the confirmation request.
DATA, SEND, and LOG_DATA	Half-duplex	These values indicate that the application program is receiving data in terms of buffers rather than logical records, and has received data, an indication that the partner LU has entered SEND state, and an indication that the received data was error log data. The received data should complete an error log variable.
DATA, DEALLOCATE, and LOG_DATA	Full-duplex and Half-duplex	These values indicate that the application program is receiving data in terms of buffers rather than logical records, and has received data, an indication that the partner LU has unconditionally deallocated the conversation, and an indication that the received data was error log data.
DATA_COMPLETE and SEND	Half-duplex	These values indicate that the application program is receiving data in terms of logical records. The application program has received all of a logical record and the partner LU has entered RECEIVE state.
DATA_COMPLETE and CONFIRM	Half-duplex	These values indicate that the application program is receiving data in terms of logical records. The application program has received a complete logical record, along with a confirmation request. The application program is to respond to the confirmation request.
DATA_COMPLETE and DEALLOCATE	Full-duplex and Half-duplex	These values indicate that the application program is receiving data in terms of logical records. It has finished receiving a logical record and has been notified that the partner LU has unconditionally deallocated the conversation.

Table 35. Valid combinations of what-received indicators (continued)

Returned Information	Valid Conversation Type	Meaning
DATA_COMPLETE, CONFIRM, and SEND	Half-duplex	These values indicate that the application program is receiving data in terms of logical records. It has finished receiving a logical record, and has received a confirmation request and an indicator that the partner LU wishes to enter RECEIVE state. The application program is to respond to the confirmation request.
DATA_COMPLETE, CONFIRM, and DEALLOCATE	Half-duplex	These values indicate that the application program is receiving data in terms of logical records. It has finished receiving a logical record, and has received a confirmation request along with an indicator that the partner LU will unconditionally deallocate the conversation. The application program is to respond to the confirmation request.
DATA_COMPLETE and LOG_DATA	Full-duplex and Half-duplex	These values indicate that the application program is receiving data in terms of logical records. The application program has finished receiving a logical record, along with an indication that the received data was error log data.
DATA_COMPLETE, LOG_DATA, and CONFIRM	Half-duplex	These values indicate that the application program is receiving data in terms of logical records. The application program has finished receiving a logical record, an indication that the data received was error log data, and a confirmation request. The data must complete the error log variable. The application program is to respond to the confirmation request.
DATA_COMPLETE, CONFIRM, SEND, and LOG_DATA	Half-duplex	These values indicate that the application program is receiving data in terms of logical records. They indicate that the application program has finished receiving a logical record, a confirmation request, an indicator that the partner LU is seeking to enter RECEIVE state, and an indicator that the received data was error log data. The application program is to respond to the confirmation request.
DATA_COMPLETE, CONFIRM, DEALLOCATE, and LOG_DATA	Half-duplex	These values indicate that the application program is receiving data in terms of logical records. The application program has finished receiving a logical record, a confirmation request, an indicator that the partner LU is seeking to deallocate the conversation, and an indicator that the received data was error log data. The application program is to respond to the confirmation request.
DATA_COMPLETE, SEND, and LOG_DATA	Half-duplex	This set of values indicates that the application program is receiving data in terms of logical records, and has finished receiving a logical record, an indication that the partner LU has entered SEND state, and an indication that the received data was error log data. The received data should complete an error log variable.
DATA_COMPLETE, DEALLOCATE, and LOG_DATA	Full-duplex and Half-duplex	These values indicate that the application program is receiving data in terms of logical records, and has finished receiving a logical record, an indication that the partner LU has unconditionally deallocated the conversation, and an indication that the received data was error log data.

For an APPCCMD CONTROL=RECEIVE specifying FILL=BUFF, VTAM normally waits for enough data to fill the application's buffer as specified by the contents of the AREA and AREALEN parameters. However, VTAM may complete the macroinstruction with less data when any of the following conditions occurs:

- A CONFIRM indication is in the What-Received field. The partner transaction program expects the local transaction to issue either positive or negative confirmation of the received data.
- A SEND indication is in the What-Received field. The partner transaction program expects the local transaction to send data, possibly a reply to the data just received request.
- A DEALLOC indication is in the What-Received field. The partner transaction program has initiated deallocation of the conversation.
- For full duplex conversations only, if the partner transaction program issues a flush type of send, an APPCCMD CONTROL=RECEIVE specifying FILL=BUFF completes with the flushed data.
- An FMH7 arrives in VTAM.

  Note: This will also result in a nonzero RCPRI, RCSEC return code.
- A conversation failure occurs.

  Note: This will also result in a nonzero RCPRI, RCSEC return code.
- Log data has arrived. This is done to isolate any prior data from the log data in the application's buffer. This specific situation is reported by VTAM by setting on the LOGRCV (RPL6RLOG) indicator in the RPL extension on the prior APPCCMD that reported the FMH-7.

When the log data itself is being received by using an APPCCMD CONTROL=RECEIVE specifying FILL=BUFF, the What-Received field indicates DATA as well as LOG_DATA (as opposed to LOGRCV(RPL6RLOG), which is not in the What-Received field).

Another way to view the above information is that VTAM will always insure that when APPCCMD CONTROL=RECEIVE FILL=BUFF completes, the contents of the application's buffer will contain one of the following items:

- Nothing
- Normal application data
- Normal application data followed by all or part of a PS Header
- All or part of a PS Header

  Note: No other data will follow a PS Header in the application's buffer.
- All or part of LOG_DATA

  Note: No other data will follow or precede the LOG_DATA in the application's buffer.

Regarding the What-Received field, also note that:

- The DATA indication in the What-Received field indicates that the receive macroinstruction specified FILL=BUFF.
- The DATA_COMPLETE and DATA_INCOMPLETE indications in the What-Received field indicate that the receive macroinstruction specified FILL=LL.
- The DATA indication together with the LOG_DATA in the What-Received field indicates that the receive macroinstruction specified FILL=BUFF.
- The DATA_INCOMPLETE or DATA_COMPLETE indications together with the LOG_DATA in the What-Received field indicate that the receive macroinstruction specified FILL=LL.
- The PS Header and PARTIAL_PS Header in the What-Received field does not indicate whether the receive macroinstruction specified FILL=BUFF or LL.

Checking the what-received field

In this example, an application program known as APPLA is receiving data on a conversation whose identifier is in register 8. The application program will check each possibility even though some indicators cannot be set at the same time, such as DATA and DATA_COMPLETE. (This was done for simplicity,

because the only negative consequences here of checking indicators that cannot be set are a few unnecessary instructions.)

```
*
         APPCCMD CONTROL=RECEIVE,                                      X
               QUALIFY=SPEC,                                           X
               RPL=RPLA,                                               X
               AAREA=RPLAX,                                            X
               ACB=APPLA,                                              X
               CONVID=(8),                                             X
               AREA=RECAREA,                                           X
               AREALEN=255,                                            X
               FILL=LL,                                                X
               OPTCD=SYN
*
         LTR   15,15               * CHECK GENERAL RETURN CODE IN 15
         BNZ   BADGENRC            * HANDLE NONZERO RETURN CODE
         LTR   0,0                 * CHECK CONDITIONAL COMPLETION
         BNZ   BADCOND             * HANDLE NONZERO RETURN CODE
*
* RETURN CODES WERE GOOD, NOW CHECK WHAT-RECEIVED INDICATORS.
* THE WHAT-RECEIVED MASK IS SAVED IN CASE ANY OF THE BAL SUBROUTINES
* NEED TO USE THE RPL EXTENSION FOR AN APPCCMD.
*
         LA    9,RPLAX             * LOAD RPL EXTENSION ADDRESS
         USING ISTRPL6X,9          * ESTABLISH ADDRESSABILITY
         MVC   MASK1,RPL6RCV1      * SAVE WHAT-RECEIVED MASK
         MVC   MASK2,RPL6RCV2      * SAVE WHAT-RECEIVED MASK
         TM    MASK1,RPL6WCFM      * CHECK CONFIRM INDICATOR
         BAL   14,CFMCODE          * GO AND HANDLE CONFIRMATION REQUEST
         TM    MASK1,RPL6WDAL      * CHECK DEALLOCATION INDICATOR
         BAL   14,DEALCODE         * GO AND HANDLE DEALLOCATION
         TM    MASK1,RPL6WSND      * CHECK SEND INDICATOR
         BAL   14,SENDCODE         * GO AND HANDLE BEING IN SEND STATE
         TM    MASK1,RPL6WDAT      * CHECK DATA INDICATOR
         BAL   14,DATACODE         * GO AND HANDLE RECEIVED DATA
         TM    MASK1,RPL6WDAC      * CHECK DATA COMPLETE INDICATOR
         BAL   14,DATCCODE         * GO AND HANDLE RECEIVED DATA
         TM    MASK1,RPL6WDAI      * CHECK DATA INCOMPLETE INDICATOR
         BAL   14,DATICODE         * GO AND HANDLE RECEIVED DATA
         TM    MASK1,RPL6WPSH      * CHECK PS HEADER INDICATOR
         BAL   14,PSHFCODE         * GO AND HANDLE RECEIVED DATA
         TM    MASK2,RPL6WPSI      * CHECK PARTIAL PS HEADER INDICATOR
         BAL   14,PSHPCODE         * GO AND HANDLE RECEIVED DATA
               .
               .
               .
MASK1    DS    XL1                 * SAVE AREA FOR WHAT-RECEIVED
MASK2    DS    XL1
RECAREA  DS    XL255               * RECEIVE BUFFER
RPLA     RPL AM=VTAM               * RPL STORAGE
RPLAX    ISTRPL6                   * RPL EXTENSION STORAGE
APPLA    ACB AM=VTAM,MACRF=LOGON,APPLID=APPLNAME * ACB STORAGE
```

Receiving information on half-duplex conversations

On half-duplex conversations, the conversation partner must be in RECEIVE state or enter RECEIVE state to receive information.

Roles of sender and receiver

Application programs and their conversation partners on half-duplex conversations regularly exchange the role of sender according to rules enforced by VTAM. When an application program is finished sending data, it can surrender its prerogative as the sender to allow its partner to send data. This type of protocol is referred to as half-duplex flip-flop protocol.

If the transaction program is in SEND state and wants to enter RECEIVE state, it must surrender its prerogative to send data by notifying VTAM that it is ready to receive. VTAM then notifies the partner that it can send data. A receiving application program receives this notification as part of the parameters returned on an APPCCMD CONTROL=RECEIVE macroinstruction.

Only the partner transaction program in SEND state can send information, but the transaction program in RECEIVE conversation state can send error indications, reply to confirmation requests, or request to

become the sender. (For details on conversation states, see Appendix A, “Conversation states,” on page 281.)

Entering RECEIVE state

VTAM allows only one side of a half-duplex conversation to receive information at any given time. When an application program receives information, the conversation must be in RECEIVE state. The conversation partner can receive error indications, confirmation responses, or requests from its partner to become the receiver.

The local transaction program enters SEND state after a successful allocation request is completed. The partner transaction program receives the FMH-5 and its side of the conversation enters RECEIVE state. After the conversation is started, the transaction program in SEND state can put the conversation in RECEIVE state when it is ready to receive data by issuing one of the following macroinstructions:

- APPCCMD CONTROL=PREPRCV
- APPCCMD CONTROL=RECEIVE, QUALIFY=SPEC

In general, the transaction program that has a conversation in SEND state can place the conversation in RECEIVE state at any time. The partner transaction program must examine the What-Received field or the CONSTATE field (RPL6CCST) on its RECEIVE requests to determine if the sending transaction program has placed this side of the conversation in RECEIVE state.

An exception exists to the rule that the sending transaction program determines when it places the conversation in RECEIVE state. If a receiving application program detects an error, it can issue an APPCCMD CONTROL=SEND, QUALIFY=ERROR macroinstruction. The original sending transaction program receives a return code indicating that an error was reported and that the application has been placed in RECEIVE state.

A special consideration exists when the application program wants to enter RECEIVE state and the application program is in SEND state or PEND_SEND state. Because the application is finishing its role as the sender, it must complete any logical records it has started to send. If the latest logical record being sent is not finished, the macroinstruction completes unsuccessfully with an error code of LOGICAL_RECORD_BOUNDARY_ERROR. The conversation remains in SEND state, at which time the application program has three choices:

- Send more data to complete the logical record.
- Use the APPCCMD CONTROL=SEND, QUALIFY=ERROR macroinstruction to indicate that an error occurred.
- Use one of the abnormal termination APPCCMD CONTROL=DEALLOC|DEALLOCQ macroinstructions or the APPCCMD CONTROL=REJECT macroinstruction to end the conversation.

The APPCCMD CONTROL=PREPRCV macroinstruction combines the functions of sending new data and a confirmation request. For example, if an application program is to finish sending a logical record, get confirmation, and begin receiving, it could issue an APPCCMD CONTROL=PREPRCV, QUALIFY=DATACON macroinstruction. The application program cannot request confirmation if it places the conversation in RECEIVE state using the APPCCMD CONTROL=RECEIVE macroinstruction.

For a general discussion of sending data, see Chapter 9, “Sending information,” on page 177.

Receiving normal information

The following macroinstructions can be used to receive information through normal, paced protocols:

- APPCCMD CONTROL=RECEIVE, QUALIFY=ANY
- APPCCMD CONTROL=RECEIVE, QUALIFY=IANY
- APPCCMD CONTROL=RECEIVE, QUALIFY=SPEC
- APPCCMD CONTROL=RECEIVE, QUALIFY=ISPEC

Using APPCCMD CONTROL=RECEIVE, QUALIFY=SPEC|ISPEC

The APPCCMD CONTROL=RECEIVE, QUALIFY=SPEC|ISPEC macroinstructions receive normal information from a specific conversation. When this macroinstruction is issued, a conversation identifier must be specified, and only incoming information for that conversation can be received.

The QUALIFY=SPEC macroinstruction causes VTAM to copy the data from the conversation specified by the CONVID parameter to the area specified by the AREA parameter. The conversation can be in continue-any or continue-specific mode. The AREALEN parameter specifies the length of the data area. If no data is ready to be received on the conversation, VTAM queues the macroinstruction until the data arrives.

The QUALIFY=ISPEC macroinstruction receives normal information that is immediately available from a specified conversation. The conversation can be in continue-any or continue-specific mode. When this macroinstruction is issued, VTAM copies all normal information that is immediately available into the supplied data area or control block specified by the AREA parameter. VTAM does not wait for additional information before completing this macroinstruction. VTAM issues an RCPRI, RCSEC combination of X'0000', X'0008', NO_INFORMATION_IMMEDIATELY_AVAILABLE.

If the application program is managing several independent conversations simultaneously, these macroinstructions would have to be issued for each conversation in order to receive information for that conversation.

Using APPCCMD CONTROL=RECEIVE, QUALIFY=ANY|IANY

The APPCCMD CONTROL=RECEIVE, QUALIFY=ANY|IANY macroinstructions receive normal information that is available from any conversation that is in continue-any mode. The macroinstruction completes with the first conversation that is in continue-any mode and receives that data.

When data is ready to be received on a continue-any mode conversation, VTAM copies the data into the data area that is specified on the AREA parameter and completes the macroinstruction. The conversation identifier for the conversation used to complete the macroinstruction is placed in the CONVID field.

The QUALIFY=IANY macroinstruction receives normal information that is immediately available from any conversation that is in continue-any mode. VTAM does not wait for other data before completing this macroinstruction. If there are no conversations in continue-any mode with normal information available, the macroinstruction is completed with an RCPRI, RCSEC combination of X'0000', X'0008', NO_INFORMATION_IMMEDIATELY_AVAILABLE.

When these macroinstructions are issued, VTAM copies data that is available into the supplied data area or control block that is specified by the AREA parameter. VTAM also returns the identification of the conversation that satisfied the macroinstruction in the CONVID parameter.

The APPCCMD CONTROL=RECEIVE, QUALIFY=ANY|IANY macroinstructions allow an application program to monitor several conversations simultaneously with a single macroinstruction.

Typically, an application program uses conversations in continue-any mode to determine which of several conversations has received a transaction to process. However, the application program does not process the transaction while the conversation remains in continue-any mode because the macroinstruction might receive data on another conversation in the middle of a transaction, or the data may become out of order.

To avoid this problem, an application program can change the conversation from continue-any mode when data is received for that conversation. This ensures that the application program can process an entire transaction at one time. The application program specifies a continuation mode of continue specific on the macroinstruction. When data on a continue-any conversation is met by the QUALIFY=ANY|IANY macroinstruction, the conversation's identifier is returned to the application program in RPL6CNVD, and the conversation is put into continue-specific mode. The application program can then issue APPCCMD CONTROL=RECEIVE, QUALIFY=SPEC|ISPEC to finish processing the transaction.

Responding to confirmation requests

Confirmation requests are allowed on half-duplex conversations. Full-duplex conversations do not support confirmation requests.

When receiving information on a conversation, application programs must be prepared to respond to confirmation requests from the partner transaction program. Only the receiving end of a conversation can receive a confirmation request.

The receipt of a confirmation request is indicated by the setting of the CONFIRM bit in the What-Received field of the RPL extension. If the bit is set on, the application program must respond to the confirmation request before issuing any other APPCCMD macroinstructions on this conversation. The application program can simultaneously receive data and other indicators in addition to the confirmation request. This data does not change the application program's need to respond to the confirmation request.

Positive response

The APPCCMD CONTROL=SEND, QUALIFY=CONFRMD macroinstruction is the only way to respond positively to a confirmation request. This macroinstruction is one of the few SEND macroinstructions that can be issued when the local conversation state is *not* SEND. (For details on the syntax and operands for the macroinstruction, see Table 9 on page 69.) Application programs can respond negatively to a confirmation request by reporting an error.

Negative response

The application program has a number of choices for responding negatively to a confirmation request. If the error condition that caused the negative response is not severe enough to end the conversation, the application program uses APPCCMD CONTROL=SEND, QUALIFY=ERROR. If the error is severe enough to terminate the conversation, the application program uses one of the abnormal deallocation macroinstructions:

- APPCCMD CONTROL=DEALLOC|DEALLOCQ, QUALIFY=ABNDPROG
- APPCCMD CONTROL=DEALLOC|DEALLOCQ, QUALIFY=ABNDSERV
- APPCCMD CONTROL=DEALLOC|DEALLOCQ, QUALIFY=ABNDTIME
- APPCCMD CONTROL=DEALLOC|DEALLOCQ, QUALIFY=ABNDUSER

In addition, if the error is severe enough to terminate the session, the application program can issue APPCCMD CONTROL=REJECT. (For details on the syntax and operands for these macroinstructions, refer to their individual descriptions in z/OS Communications Server: SNA Programmer's LU 6.2 Reference.)

Example of confirmation responses

In this example, an application program known as APPLA is receiving data over a conversation identified by the ID stored in the CONVERID storage area. APPLA issues a RECEIVE and branches to a subroutine that handles the received data. The subroutine sets a flag byte referenced by the STATUS label to indicate errors. APPLA always responds positively to the confirmation request if the flag byte is set to 0.

```
         L     9,CONVERID        * LOAD SAVED CONVERSATION ID
*
         APPCCMD CONTROL=RECEIVE,                                      X
               QUALIFY=SPEC,                                           X
               RPL=RPLA,                                               X
               AAREA=RPLAX,                                            X
               ACB=APPLA,                                              X
               CONVID=(9),                                             X
               OPTCD=SYN,                                              X
               AREA=RECAREA,                                           X
               AREALEN=255,                                            X
               FILL=LL,                                                X
               CONMODE=CS
*
         LTR   15,15             * CHECK GENERAL RETURN CODE IN 15
         BNZ   BADGENRC          * HANDLE NONZERO RETURN CODE
         LTR   0,0               * CHECK CONDITIONAL COMPLETION
         BNZ   BADCOND           * HANDLE NONZERO RETURN CODE
         MVC   SAVEMASK,RPL6RCV1 * SAVE WHAT-RECEIVED MASK
         BAL   14,CHKREC         * CHECK RECEIVED DATA
         TM    SAVEMASK,RPL6WCFM * CONFIRMATION REQUEST INCLUDED?
         BNO   DEALCHCK          * IF NOT, CHECK DEALLOCATION INDICATOR
         CLI   STATUS,X'00'      * IS STATUS OK?
         BNE   NEGRESP           * IF NOT ISSUE NEGATIVE RESPONSE
```

```
*
* STATUS BYTE WAS OK, SO RESPOND POSITIVELY TO CONFIRMATION REQUEST.
*
         APPCCMD CONTROL=SEND,                                         X
               QUALIFY=CONFRMD,                                        X
               RPL=RPLA,                                               X
               AAREA=RPLAX,                                            X
               ACB=APPLA,                                              X
               CONVID=CONVERID,                                        X
               CONMODE=CS,                                             X
               OPTCD=SYN
*
         LTR   15,15             * CHECK GENERAL RETURN CODE IN 15
         BNZ   BADGENRC          * HANDLE NONZERO RETURN CODE
         LTR   0,0               * CHECK CONDITIONAL COMPLETION
         BNZ   BADCOND           * HANDLE NONZERO RETURN CODE
*
* AT THIS POINT THE CONFIRMATION RESPONSE HAS BEEN MADE AND THE
* RETURN CODES FROM IT WERE GOOD.
*
*
                  .
                  .
                  .
NEGRESP  APPCCMD CONTROL=SEND,                                         X
               QUALIFY=ERROR,                                          X
               TYPE=PROGRAM,                                           X
               RPL=RPLA,                                               X
               AAREA=RPLAX,                                            X
               ACB=APPLA,                                              X
               CONVID=CONVERID,                                        X
               RECLEN=0,                                               X
               OPTCD=SYN
*
         LTR   15,15             * CHECK GENERAL RETURN CODE IN 15
         BNZ   BADGENRC          * HANDLE NONZERO RETURN CODE
         LTR   0,0               * CHECK CONDITIONAL COMPLETION
         BNZ   BADCOND           * HANDLE NONZERO RETURN CODE
*
* AT THIS POINT THE NEGATIVE CONFIRMATION RESPONSE HAS BEEN MADE AND
* THE RETURN CODES FROM IT WERE GOOD.
*
                  .
                  .
                  .
CONVERID DS    XL4               * CONVERSATION ID
STATUS   DC    X'00'             * RECEIVED DATA OK FLAG
SAVEMASK DS    XL1               * SAVE SPACE FOR WHAT-RECEIVED MASK
RECAREA  DS    XL255             * RECEIVE STORAGE AREA
RPLA     RPL AM=VTAM             * RPL STORAGE
RPLAX    ISTRPL6                 * RPL EXTENSION STORAGE
APPLA    ACB AM=VTAM,MACRF=LOGON,APPLID=APPLNAME * ACB STORAGE
```

Receiving expedited information

When a partner sends a request to send indication, for example, by using an APPCCMD CONTROL=SEND, CONTROL=RQSEND macroinstruction, the indication may be reported to the local application in one of two ways:

- On the completion of any APPCCMD macroinstruction that allows the SIGRCV (RPL6RSIG) and SIGDATA (RPL6SGNL) indications
- Only on the completion of expedited macroinstructions APPCCMD CONTROL=SENDEXPD and APPCCMD CONTROL=RCVEXPD

The application determines the ways of receiving the request to send information when the conversation begins. By specifying RTSRTRN=BOTH on the APPCCMD CONTROL=ALLOC or APPCCMD CONTROL=RCVFMH5, both ways are enabled. By specifying RTSRTRN=EXPD on the APPCCMD CONTROL=ALLOC or APPCCMD CONTROL=RCVFMH5, the request to send indication may be received only on an expedited macroinstruction, that is, either APPCCMD CONTROL=SENDEXPD or APPCCMD CONTROL=RCVEXPD.

Note: An APPCCMD CONTROL=RCVEXPD macroinstruction completes successfully if the only expedited information available is a request to send indication.

The APPCCMD CONTROL=RCVEXPD macroinstruction will always receive expedited *data* sent from the conversation partner, for example, by using the APPCCMD CONTROL=SENDEXPD macroinstruction.

Expedited information can be received by a transaction program in any conversation state except the following information:

- PENDING_DEALLOCATE
- PENDING_ALLOCATE
- END_CONVERSATION

The following macroinstructions can be used to receive expedited information:

- APPCCMD CONTROL=RCVEXPD, QUALIFY=ANY
- APPCCMD CONTROL=RCVEXPD, QUALIFY=IANY
- APPCCMD CONTROL=RCVEXPD, QUALIFY=SPEC
- APPCCMD CONTROL=RCVEXPD, QUALIFY=ISPEC

Using APPCCMD CONTROL=RCVEXPD, QUALIFY=SPEC|ISPEC

The APPCCMD CONTROL=RCVEXPD, QUALIFY=SPEC|ISPEC receives expedited information from a specified conversation. The QUALIFY=SPEC macroinstruction causes the transaction program to wait for expedited information to arrive to satisfy the macroinstruction request. If expedited information is immediately available, the application receives it without waiting.

The QUALIFY=ISPEC macroinstruction does not wait for expedited information to arrive to satisfy the macroinstruction request. If expedited information is not available, an RCPRI, RCSEC combination of X'0000', X'0008', NO_IMMEDIATELY_AVAILABLE_INFORMATION is returned to the application.

If the length of the area specified by the application is not sufficient to receive all the expedited data available, an RCPRI, RCSEC combination of X'002C', X'0008', PARAMETER_ERROR—SUPPLIED_LENGTH_INSUFFICIENT, is returned to the application.

Using APPCCMD CONTROL=RCVEXPD, QUALIFY=ANY|IANY

The APPCCMD CONTROL=RCVEXPD, QUALIFY=ANY|IANY macroinstructions receive expedited information that is available from any conversation whose expedited information mode is continue-any.

The QUALIFY=ANY macroinstruction waits for expedited information to arrive on a conversation in continue-any mode to satisfy the macroinstruction request. If expedited data is available, then the application must receive the entire amount of expedited data available.

The QUALIFY=IANY macroinstruction does not wait for expedited information to arrive on a conversation to satisfy the macroinstruction request, but immediately receives any expedited information available. If expedited information is not available on any conversation in continue-any mode, an RCPRI, RCSEC combination of (X'0000',X'0008') NO_IMMEDIATELY_AVAILABLE_INFORMATION is returned to the application.

If the length of the area specified by the application is not sufficient to receive all the expedited data available, an RCPRI, RCSEC combination of X'002C', X'0008', PARAMETER_ERROR—SUPPLIED_LENGTH_INSUFFICIENT, is returned to the application.

Receiving information on full-duplex conversations

On full-duplex conversations, there are five valid conversation states for receiving information:

- SEND/RECEIVE
- RECEIVE-ONLY
- PENDING_SEND/RECEIVE_LOG
- PENDING_RECEIVE-ONLY_LOG
- PENDING_RESET_LOG

Each of these states are described in the following sections.

Roles of sender and receiver

For full-duplex conversations, both conversations partners can send and receive information simultaneously. The partners do not have to exchange roles as sender and receiver to send or receive information.

Entering SEND/RECEIVE state

The transaction program initiating the full-duplex conversation sends an allocate request to the partner transaction program. The local conversation then enters SEND/RECEIVE state. The partner receives the FMH-5 and also enters SEND/RECEIVE state. Either side of a full-duplex conversation in SEND/RECEIVE state can receive information.

If the partner transaction program does not support full-duplex, the session capability is negotiated to half-duplex. A conversation can still be allocated, but only a half-duplex conversation is allowed on a half-duplex session.

Entering RECEIVE_ONLY state

A transaction program enters RECEIVE_ONLY state when it initiates normal conversation deallocation by issuing an APPCCMD CONTROL=DEALLOC, QUALIFY=FLUSH|DATAFLU. In this state the transaction program can continue to receive information until the partner transaction program completes normal conversation deallocation.

Entering PENDING_SEND/RECEIVE_LOG state

The transaction program enters PENDING_SEND/RECEIVE_LOG state when error log data must be received before normal receiving of information can continue. This can occur when the partner has issued an APPCCMD CONTROL=SEND, QUALIFY=ERROR and supplied error log data. The conversation state was SEND/RECEIVE before the error condition was detected.

Entering PENDING_RECEIVE-ONLY_LOG state

The transaction program enters PENDING_RECEIVE-ONLY_LOG state when error log data must be received before normal receiving of information can continue. This can occur when the partner has issued an APPCCMD CONTROL=SEND, QUALIFY=ERROR and supplied error log data. The conversation state was RECEIVE-ONLY before the error condition was detected.

Entering PENDING_RESET_LOG state

The local conversation enters PENDING_RESET_LOG state when error log data must be received before the conversation is ended. This can occur when the partner has issued an abnormal deallocation request and supplied error log data.

Receiving normal information

The following macroinstructions can be used to receive information through normal, paced protocols:

- APPCCMD CONTROL=RECEIVE, QUALIFY=ANY
- APPCCMD CONTROL=RECEIVE, QUALIFY=IANY
- APPCCMD CONTROL=RECEIVE, QUALIFY=SPEC
- APPCCMD CONTROL=RECEIVE, QUALIFY=ISPEC

Using APPCCMD CONTROL=RECEIVE, QUALIFY=SPEC|ISPEC

The APPCCMD CONTROL=RECEIVE, QUALIFY=SPEC|ISPEC macroinstructions receive normal information from a specific conversation. When this macroinstruction is issued, a conversation identifier must be specified, and incoming information from only the specified conversation can be received.

The QUALIFY=SPEC version causes VTAM to copy the data from the conversation specified by the CONVID parameter to the area specified by the AREA parameter. The conversation can be in continue-any or continue-specific mode. The AREALEN parameter specifies the length of the data area. If no data is ready to be received on the conversation, VTAM queues the macroinstruction until the data arrives.

The QUALIFY=ISPEC version receives normal information that is immediately available from a specified conversation. The conversation can be in continue-any or continue-specific mode. When the QUALIFY=ISPEC macroinstruction is issued, VTAM copies all normal information that is immediately available into the supplied data area or control block specified by the AREA parameter. VTAM does not wait for additional information before returning a response to this macroinstruction. If normal information is not available, VTAM completes the macroinstruction with an RCPRI, RCSEC combination of X'0000', X'0008', NO_INFORMATION_IMMEDIATELY_AVAILABLE.

If the application program is managing several independent conversations simultaneously, these macroinstructions would have to be issued for each conversation in order to receive information from that conversation.

Using APPCCMD CONTROL=RECEIVE, QUALIFY=ANY|IANY

The APPCCMD CONTROL=RECEIVE, QUALIFY=ANY|IANY macroinstructions receive normal information that is available from any conversation that is in continue-any mode.

When data is ready to be received on a continue-any mode conversation, VTAM copies the data into the data area that is specified on the AREA parameter and if QUALIFY=IANY is specified, completes the macroinstruction. If QUALIFY=ANY is specified, VTAM may wait for additional information to be received before completing the macroinstruction. The conversation identifier for the conversation used to complete the macroinstruction is placed in the CONVID field.

The QUALIFY=IANY version receives normal information that is immediately available from a conversation that is in continue-any mode. VTAM does not wait for additional information before completing this macroinstruction. If there are no conversations in continue-any mode with information available, VTAM issues an RCPRI, RCSEC combination of X'0000', X'0008', NO_INFORMATION_IMMEDIATELY_AVAILABLE.

The APPCCMD CONTROL=RECEIVE, QUALIFY=ANY|IANY macroinstructions allow an application program to monitor several conversations simultaneously with a single macroinstruction.

Typically, an application program uses conversations in continue-any mode to determine which of several conversations has received a transaction to process. However, the application program does not process the transaction while the conversation remains in continue-any mode because the macroinstruction might receive data on another conversation in the middle of a transaction and data may get out of order.

To avoid this problem, an application program can change the conversation from continue-any mode when data is received from that conversation. This ensures that the application program can process an entire transaction at one time. The application program specifies a continuation mode of continue specific on the macroinstruction. When data on a continue-any conversation is met by the QUALIFY=ANY|IANY macroinstruction which also specified CONMODE=CS, the conversation is put into continue-specific mode. The application program can then issue APPCCMD CONTROL=RECEIVE, QUALIFY=SPEC|ISPEC to finish processing the transaction.

Receiving expedited data

Expedited data can be received by a transaction program in any conversation state except FDX_RESET.

The following macroinstructions can be used to receive expedited information:

- APPCCMD CONTROL=RCVEXPD, QUALIFY=ANY

- APPCCMD CONTROL=RCVEXPD, QUALIFY=IANY
- APPCCMD CONTROL=RCVEXPD, QUALIFY=SPEC
- APPCCMD CONTROL=RCVEXPD, QUALIFY=ISPEC

Using APPCCMD CONTROL=RCVEXPD, QUALIFY=SPEC|ISPEC

The APPCCMD CONTROL=RCVEXPD, QUALIFY=SPEC|ISPEC macroinstruction receives expedited information from a specified conversation.

The QUALIFY=SPEC macroinstruction may wait for expedited information to arrive to satisfy the macroinstruction request. If expedited information is immediately available, the application receives it without waiting.

The QUALIFY=ISPEC macroinstruction does not wait for expedited information to arrive to satisfy the macroinstruction request. If expedited data is not available, an RCPRI, RCSEC combination of X'0000', X'0008' NO_IMMEDIATELY_AVAILABLE_INFORMATION is returned to the application.

If the length of the area specified by the application is not sufficient to receive all the expedited data available, an RCPRI, RCSEC combination of X'002C', X'0008', PARAMETER_ERROR_SUPPLIED_LENGTH_INSUFFICIENT is returned to the application.

Using APPCCMD CONTROL=RCVEXPD, QUALIFY=ANY|IANY

The APPCCMD CONTROL=RCVEXPD, QUALIFY=ANY|IANY macroinstructions receive expedited information that is available from any conversation whose expedited information mode is continue-any.

The QUALIFY=ANY macroinstruction can wait for expedited information to arrive on a conversation in continue-any mode to satisfy the macroinstruction request. The application must receive the entire amount of expedited data available.

The QUALIFY=IANY macroinstruction does not wait for expedited information to arrive on a conversation to satisfy the macroinstruction request. If there are no conversations in continue-any mode with expedited information available, the macroinstruction is completed with an RCPRI, RCSEC of X'0000', X'0008', NO_IMMEDIATELY_AVAILABLE_INFORMATION.

If the length of the area specified by the application is not sufficient to receive all the expedited data available, an RCPRI, RCSEC combination of X'002C', X'0008' PARAMETER_ERROR_SUPPLIED_LENGTH_INSUFFICIENT is returned to the application.

Specifying how information is received

Application programs can control how certain information is received and processed by specifying parameters on the APPCCMD CONTROL=RECEIVE macroinstruction. Application programs can specify the following options:

FILL
Determines whether the application program receives information in terms of logical records or in terms of defined buffer boundaries. This option is described in “Logical records versus buffers” on page 211.

CONMODE
Specifies whether normal information is received in terms of buffers or logical record format, and whether a specific-type or any-type macroinstruction is required to receive the information. You may also specify whether the mode remains unchanged upon completion of the macroinstruction. This option is described in “Continuation modes for receiving normal information” on page 212.

CONXMOD
Specifies the mode for receiving expedited information. The mode can be specified so that expedited information can be received by either a specific or any-type macroinstruction, only a specific-type macroinstruction, or you may specify that the mode remain unchanged upon completion of the macroinstruction. This option in described in “Continuation modes for receiving expedited information” on page 215.

Logical records versus buffers

Application programs can control whether VTAM respects logical record boundaries when fulfilling a RECEIVE macroinstruction. This is done by means of the FILL keyword operand on the APPCCMD CONTROL=RECEIVE, QUALIFY=SPEC and QUALIFY=ISPEC macroinstructions. For the APPCCMD CONTROL=RECEIVE, QUALIFY=ANY and QUALIFY=IANY macroinstructions, the choice is determined on an earlier macroinstruction by means of the CONMODE parameter. (This is discussed in "Continuation modes for receiving normal information" on page 212.)

The FILL keyword specifies whether the application program is to receive data in terms of the logical-record format of the data. This keyword, which corresponds to the FILL=LL|BUFFER parameter described in LU 6.2 architecture, is labeled RPL6FILL in the RPL extension.

The following values are valid for this parameter:

FILL=BUFF
Specifies that the application program is to receive data independently of its logical-record format, up to the length specified by the AREALEN field of the RPL. This corresponds to FILL=BUFFER on the RECEIVE_AND_WAIT verb described in LU 6.2 architecture. Note that for APPCCMD CONTROL=RECEIVE, QUALIFY=ISPEC, the data received may be less than the length specified on AREALEN.

FILL=LL
Specifies that the application program is to receive one logical record, or whatever portion of the logical record is available, up to the length specified by the AREALEN field of the RPL. This corresponds to FILL=LL on the RECEIVE_AND_WAIT verb described in LU 6.2 architecture. Note that for APPCCMD CONTROL=RECEIVE, QUALIFY=ISPEC, the data received may be less than the length specified by the AREALEN field and not complete a logical record.

By specifying FILL=LL on the APPCCMD CONTROL=RECEIVE macroinstruction, the application program can receive a logical record even if the total amount of data is less than the AREALEN specified on the RPL for the RECEIVE. If FILL=BUFF is specified, the macroinstruction is usually not completed until enough data is received to fill the application's entire buffer. In such cases, it is entirely up to the application program to detect the boundaries of logical records. When FILL=LL is specified, data in excess of a logical record is stored by VTAM and is available to use in fulfilling the next RECEIVE. Table 36 on page 211 illustrates the use of FILL and its impact on how much data is received.

Table 36. Examples of the FILL parameter

Example	Macroinstruction	Results
A 120-byte logical record arrives for the application program, and VTAM receives it. The application program has a 100-byte buffer.	APPCCMD CONTROL=RECEIVE, QUALIFY=SPEC, AREA=buffer address, AREALEN=100, FILL=LL	The application program receives the full 100 bytes that its buffer holds. VTAM continues to store the remaining 20 bytes of the logical record. The DATA_INCOMPLETE indicator is set on in the What-Received field in the RPL extension.
APPCCMD CONTROL=RECEIVE, QUALIFY=SPEC, AREA=buffer address, AREALEN=100, FILL=BUFF	One hundred bytes are put in the application's RECEIVE buffer, and VTAM continues to store the remainder. The DATA indicator in the What-Received field is set on.	APPCCMD CONTROL=RECEIVE QUALIFY=ISPEC AREA=buffer address, AREALEN=100, FILL=LL
The application program receives the full 100 bytes that its buffer holds. VTAM continues to store the remaining 20 bytes of the logical record. The DATA_INCOMPLETE indicator is set on in the What-Received field in the RPL extension.	APPCCMD CONTROL=RECEIVE, QUALIFY=ISPEC, AREA=buffer address, AREALEN=100, FILL=BUFF	One hundred bytes are put in the application's RECEIVE buffer, and VTAM continues to store the remainder. The DATA indicator in the What-Received field is set on.

Table 36. Examples of the FILL parameter (continued)

Example	Macroinstruction	Results
The program issues RECEIVE after the 20 remaining bytes from the first example have been stored by VTAM, and no other data has been received by VTAM for the application program.	APPCCMD CONTROL=RECEIVE, QUALIFY=SPEC, AREA=buffer address, AREALEN=100, FILL=LL	The 20 bytes are received by the application program, and the DATA_COMPLETE indicator is turned on in the What-Received field of the RPL extension. RECLEN holds a value of X'14'.
APPCCMD CONTROL=RECEIVE, QUALIFY=SPEC, AREA=buffer address, AREALEN=100, FILL=BUFF	The macroinstruction does not receive the 20 bytes of data yet, unless this is the last logical record the partner application program is sending prior to a confirmation request, deallocation request, or attempt to enter RECEIVE state. VTAM normally waits until another 80 bytes of data arrive to complete the RECEIVE. When the macroinstruction does complete, the DATA bit in the What-Received field will be set on.	APPCCMD CONTROL=RECEIVE QUALIFY=ISPEC AREA=buffer address, AREALEN=100, FILL=LL
The 20 bytes are received by the application program, and the DATA_COMPLETE indicator is turned on in the What-Received field of the RPL extension. RECLEN holds a value of X'14'.	APPCCMD CONTROL=RECEIVE, QUALIFY=ISPEC, AREA=buffer address, AREALEN=100, FILL=BUFF	The 20 bytes are received by the application program, and the DATA indicator is turned on in the What-Received field of the RPL extension. RECLEN holds a value of X'14'.
VTAM receives 100 bytes for the application program—the first 50 finishing a logical record and the next 50 making up a complete logical record. The application program has specified a buffer of 100 bytes.	APPCCMD CONTROL=RECEIVE, QUALIFY=SPEC, AREA=buffer address, AREALEN=100, FILL=LL	The application program receives the first 50 bytes and the other 50 continue to be stored by VTAM. The DATA_COMPLETE indicator bit is set on.
APPCCMD CONTROL=RECEIVE, QUALIFY=SPEC, AREA=buffer address, AREALEN=100, FILL=BUFF	The application program receives the entire 100 bytes. The DATA indicator bit is set on.	APPCCMD CONTROL=RECEIVE QUALIFY=ISPEC AREA=buffer address, AREALEN=100, FILL=LL
The application program receives the first 50 bytes and the other 50 continue to be stored by VTAM. The DATA_COMPLETE indicator bit is set on.	APPCCMD CONTROL=RECEIVE, QUALIFY=ISPEC, AREA=buffer address, AREALEN=100, FILL=BUFF	The application program receives the entire 100 bytes. The DATA indicator bit is set on.

Continuation modes for receiving normal information

At any given time, a conversation can be in one of three continuation modes:

- Buffer-continue-any
- Continue-specific
- Logical-record-continue-any

Each of these modes can be specified on any APPCCMD that allows the CONMODE parameter.

CONMODE=BUFFCA
Specifies that the conversation is to be placed in buffer-continue-any mode. It indicates that normal information may be received by either APPCCMD CONTROL=RECEIVE, SPEC|ISPEC or APPCCMD CONTROL=RECEIVE, QUALIFY=ANY|IANY and that the application program is to receive data independently of the logical-record format of the data.

CONMODE=CS
Specifies that the conversation is to be placed in continue-specific mode. It indicates that only APPCCMD CONTROL=RECEIVE, QUALIFY=SPEC|ISPEC can be used to receive normal information on this conversation. When the application issues APPCCMD CONTROL=RECEIVE, QUALIFY=SPEC or APPCCMD CONTROL=RECEIVE, QUALIFY=ISPEC, it must indicate whether the data is to be received in terms of the logical record format of the data or independently of the logical record format of the data.

CONMODE=LLCA
Specifies that the conversation is to be placed in logical-record-continue-any mode. It indicates that normal information may be received by either APPCCMD CONTROL=RECEIVE, SPEC|ISPEC or APPCCMD CONTROL=RECEIVE, QUALIFY=ANY|IANY and that the application program is to receive information in terms of the logical-record format of the data.

CONMODE=SAME
Specifies that the continuation mode of the conversation is to remain unchanged.

The continuation mode of a conversation can be set on the CONMODE parameter (RPL6CMOD) of several macroinstructions. See “Keywords and returned parameters” on page 72 to determine which macroinstructions support this parameter. The APPCCMD CONTROL=RESETRCV macroinstruction can also be used to change the continuation mode of a conversation. For more information about this macroinstruction, refer to z/OS Communications Server: SNA Programmer's LU 6.2 Reference.

Example of using any-mode RECEIVEs

In this example, an application program known as APPLA is receiving data with an any-mode RECEIVE. After it receives the data, it moves the data into a buffer contained in a control block used by the application program to represent a conversation. (The control block is one in a chain of conversation blocks. The application program uses the returned conversation identifier to find the proper block.) The application program then issues another any-mode RECEIVE.

```
*
         APPCCMD CONTROL=RECEIVE,                                      X
               QUALIFY=ANY,                                            X
               RPL=RPLA,                                               X
               AAREA=RPLAX,                                            X
               ACB=APPLA,                                              X
               AREA=RECAREA,                                           X
               AREALEN=255,                                            X
               CONMODE=CS,                                             X
               OPTCD=SYN
*
         LTR   15,15               * CHECK GENERAL RETURN CODE IN 15
         BNZ   BADGENRC            * HANDLE NONZERO RETURN CODE
         LTR   0,0                 * CHECK CONDITIONAL COMPLETION
         BNZ   BADCOND             * HANDLE NONZERO RETURN CODE
*
* RETURN CODES WERE GOOD. FIND THE CONVERSATION BLOCK AND MOVE DATA.
*
         LA    10,RPLA             * LOAD RPL ADDRESS
         USING IFGRPL,10           * ESTABLISH ADDRESSABILITY
         LA    9,RPLAX             * LOAD RPL EXTENSION ADDRESS
         USING ISTRPL6X,9          * ESTABLISH ADDRESSABILITY
         L     8,BLKCHAIN          * GET START OF BLOCK CHAIN
         USING BLKMAP,8            * ESTABLISH ADDRESSABILITY
SEARCHLP LTR   8,8                 * IF CHAIN ZERO, BLOCK NOT FOUND
         BZ    ERROR               * HANDLE ERROR IF SO
         CLC   RPL6CNVD,BLKCNVD    * RIGHT CONVERSATION BLOCK?
         BE    STOREBUF            * IF SO, TRANSFER DATA
         L     8,BLKNEXT           * GET NEXT IN CHAIN
         B     SEARCHLP            * GO AND CHECK NEW POINTER VALUE
STOREBUF MVC   BLKBUF(255),RECAREA * MOVE DATA FROM RECEIVE BUFFER
         MVC   BLKBUFLN,RPLRLEN    * STORE LENGTH OF RECEIVED DATA
               •
```

```
               .
               .
BLKCHAIN DS    A                  * POINTER TO CHAIN OF BLOCKS
RECAREA  DS    XL255              * RECEIVE BUFFER
RPLA     RPL AM=VTAM               * RPL STORAGE
RPLAX    ISTRPL6                   * RPL EXTENSION STORAGE
APPLA    ACB AM=VTAM,MACRF=LOGON,APPLID=APPLNAME * ACB STORAGE
               .
               .
               .
BLKMAP   DSECT
BLKCNVD  DS    XL4                * CONVERSATION ID
BLKNEXT  DS    A                  * NEXT IN CHAIN
BLKBUFLN DS    XL4                * LENGTH OF DATA IN BUFFER
BLKBUF   DS    XL255              * CONVERSATION BUFFER
```

Once the application program has finished receiving data on the conversation, the conversation can be returned to continue-any state by using the CONMODE operand or by using the RESETRCV macroinstruction.

This example could be coded as follows:

```
*
RECLOOP APPCCMD CONTROL=RECEIVE,                                      X
                QUALIFY=SPEC,                                         X
                RPL=RPLA,                                             X
                AAREA=RPLAX,                                          X
                ACB=APPLA,                                            X
                CONVID=(8),                                           X
                AREA=RECAREA,                                         X
                AREALEN=255,                                          X
                FILL=LL,                                              X
                CONMODE=CS,                                           X
                OPTCD=SYN
*
       LTR      15,15               * CHECK GENERAL RETURN CODE IN 15
       BNZ      BADGENRC            * HANDLE NONZERO RETURN CODE
       LTR      0,0                 * CHECK CONDITIONAL COMPLETION
       BNZ      BADCOND             * HANDLE NONZERO RETURN CODE
*
* RETURN CODES WERE GOOD, CHECK WHAT-RECEIVED INDICATOR.
*
       LA       9,RPLAX             * LOAD RPL EXTENSION ADDRESS
       USING    ISTRPL6X,9          * ESTABLISH ADDRESSABILITY
       BAL      14,PROCDATA         * BRANCH TO HANDLE RECEIVED DATA
       TM       RPL6RCV1,RPL6WDAC   * CHECK DATA COMPLETE INDICATOR
       BNO      RECLOOP             * IF DATA LEFT, DO ANOTHER RECEIVE
*
* DATA ALL RECEIVED.  RESET CONTINUATION MODE
*
  APPCCMD CONTROL=RESETRCV,
          RPL=RPLA,                                                   X
          AAREA=RPLAX,                                                X
          ACB=APPLA,                                                  X
          CONVID=(8),                                                 X
          CONMODE=BUFFCA,                                             X
          OPTCD=SYN
*
  LTR      15,15                 * CHECK GENERAL RETURN CODE IN 15
  BNZ      BADGENRC              * HANDLE NONZERO RETURN CODE
  LTR      0,0                   * CHECK CONDITIONAL COMPLETION
  BNZBAD   COND                  * HANDLE NONZERO RETURN CODE

*
* RETURN CODES WERE GOOD, CONTINUATION MODE RESET.
*
                    .
                    .
                    .
  RECAREA DS     XL255            * RECEIVE BUFFER
  RPLA    RPL AM=VTAM             * RPL STORAGE
  RPLAX   ISTRPL6                 * RPL EXTENSION STORAGE
  APPLA   ACB AM=VTAM,MACRF=LOGON,APPLID=APPLNAME * ACB STORAGE
```

Continuation modes for receiving expedited information

The CONXMOD parameter specifies the mode for receiving expedited information upon completion of the APPCCMD issued. The continuation mode for receiving expedited information can be set on the CONXMOD parameter of several macroinstructions. See “Keywords and returned parameters” on page 72 to determine which macroinstructions support this parameter. The APPCCMD CONTROL=RESETRCV macroinstruction can also be used to change the continuation mode of a conversation. For more information about this macroinstruction, refer to z/OS Communications Server: SNA Programmer's LU 6.2 Reference.

The following values are valid for this parameter:

CONXMOD=CA
This specifies that the mode for expedited information is to be put in such a state that expedited information can be received by either a specific type or an any-type macroinstruction, for example, APPCCMD CONTROL=RCVEXPD, QUALIFY=SPEC or ISPEC, or APPCCMD CONTROL=RCVEXPD, QUALIFY=ANY or IANY.

CONXMOD=CS
This specifies that the mode for expedited information is to be put in such a state that expedited information may be received by a only specific type macroinstruction, for example, APPCCMD CONTROL=RCVEXPD, QUALIFY=SPEC or ISPEC.

CONXMOD=SAME
This specifies that the conversation mode for expedited information is to remain unchanged at the completion of this macroinstruction.

Error handling

A general discussion of error reporting is found in Chapter 14, “Handling errors,” on page 263. Some error-handling considerations are particularly pertinent to receiving data.

When an application program receiving data over a conversation reports an error through an APPCCMD macroinstruction, it no longer receives data but is put in SEND state. Data remaining to be received is purged.

The purged information includes:

- Confirmation requests
- Conditional deallocation notices
- Conversation data

In addition, return codes on a macroinstruction that cause data to be purged are affected. Allocation error return codes, abnormal deallocation return codes, and program error and LU services error return codes are reported as a DEALLOCATE_NORMAL return code.

Purging does not occur when a negative response is reported to a confirmation request. If a confirmation request is received, all data sent up to that point has been received as well.

Chapter 11. Sending and receiving data using high performance data transfer

About this chapter

High performance data transfer (HPDT) optimizes the performance of large message transfers (sending and receiving operations) for VTAM LU 6.2 applications. No application change is required to obtain HPDT services. When an LU 6.2 application issues an APPCCMD send or receive request, HPDT services are invoked automatically if the session is bound over a connection that supports channel I/O operations using buffers in the communications storage manager (CSM). These connections are explained in the z/OS Communications Server: SNA Network Implementation Guide. HPDT services are also available for data transfers between two LU 6.2 applications that reside on the same host. HPDT increases total system throughput by reducing the requirements on system resources (CPU, memory bus, effective pathlength, and cache) and thereby increasing the total number of streams that can be supported by a CPU with given MIPS (one million instructions per second) capacity, memory bus bandwidth, and configuration limits.

This chapter describes an HPDT interface that applications can use for obtaining performance benefits even greater than when HPDT services are available for normal APPCCMD send and receive requests. The HPDT interface includes an API for using CSM (IVTCSM macroinstruction) and an extension to the APPCCMD macroinstruction (OPTCD=XBUFLST). Optimal performance is achieved by using the HPDT interface to eliminate the data copy at the APPCCMD interface.

The role of CSM and the IVTCSM macroinstruction

CSM provides a means for authorized host applications to share data with other CSM users without having to copy the data. (A CSM user can be any system-authorized application program or product.) CSM is also used by VTAM on behalf of any VTAM LU 6.2 application, authorized or not, to transfer data between CSM buffers and the multipath channel with fewer data moves. Applications can create buffer pools in CSM and load data in buffers, which are represented by buffer tokens. Ownership of these buffers can be changed to allow other CSM users to assume responsibility for storage return to CSM. This allows an application and VTAM to exchange ownership of a single piece of commonly addressable storage so that there is no need to move the data at the APPCCMD API. VTAM uses the same CSM storage area for channel I/O. CSM storage is data space, extended common service area (ECSA) storage, or high virtual common (HVCOMM) storage above the 2 GB bar.

CSM provides an API (IVTCSM) that enables applications to request CSM services, such as registering to use a pool of buffers, getting or freeing a number of buffers from a pool, or changing the ownership of a buffer. For more information about the CSM requests available to the application, refer to z/OS Communications Server: CSM Guide.

For more complete information about CSM, refer to z/OS Communications Server: CSM Guide.

Applications that use the HPDT interface

Because performance improvements are targeted at sending and receiving large data objects, applications that typically require high-speed data transfer are most suited to use HPDT. For example, in a host file server environment, certain applications may serve a smaller number of users concurrently storing and retrieving data than in a traditional online transaction processing environment. For each access, bulk data transfer is required. This is especially true for multimedia servers where vast amounts of bits must be transmitted for each minute of video stream. HPDT was designed to optimize performance for applications requiring efficient, high-speed transfer of large amounts of data. Specific characteristics about applications determine the ideal situations for using HPDT. The following items are some examples of operations in which the HPDT interface is ideal:

- Host file server

- Transmitting multimedia, video, or images
- Backing up and recovering large databases
- Retrieving archived data

Overall, applications that perform frequent transfer of small messages are not highly suited for using the HPDT interface.

Using the APPCCMD macroinstruction for HPDT requests

The application program sends or receives data in CSM buffers by issuing an APPCCMD send or receive macroinstruction with OPTCD=XBUFLST specified. The XBUFLST option indicates that the application is sending or receiving data using an *extended buffer list*. The extended buffer list points to data that resides in CSM buffers. See "Macroinstructions used by HPDT applications" on page 219 for information about macroinstructions that send or receive data using HPDT.

The APPCCMD side of the HPDT interface also includes vectors that applications can use to determine whether a particular session supports HPDT services or to inform VTAM about how the application intends to receive HPDT data. See "Verifying the session's capabilities" on page 221 and "Passing HPDT receive requirements to VTAM" on page 229 for more information about vector lists available to applications that use the HPDT interfaces.

The use of HPDT services on one end of a conversation is transparent to the partner LU. If an application specifies XBUFLST when the HPDT service is not available, VTAM executes the request using normal APPCCMD services.

Designing programs to use HPDT

This section describes considerations for writing new, or modifying existing, application programs to implement HPDT. The following concepts and tasks are described:

Design considerations for HPDT applications

Each HPDT send or receive request is processed independently. An application is, therefore, free to mix HPDT and non-HPDT requests. Valid request combinations include, but are not limited to, the following items:

- HPDT send with no HPDT receive
- HPDT receive with no HPDT send
- Both HPDT send and receive
- HPDT send and non-HPDT send
- HPDT receive and non-HPDT receive

The capability of mixing types of send and receive requests allows application programmers to make decisions on the use of HPDT based on the length of the data being sent or the anticipated length of the data being received. Also, an application that typically moves large packets in one direction can be written to implement HPDT only for the direction of the large packet flow and use the normal data path for the direction of the smaller flow.

When mixing HPDT requests with non-HPDT requests, be aware that after the first HPDT send has been issued on a conversation, logical record checking is not performed for all send operations for the duration of that conversation.

HPDT services are not available when VTAM cryptography or VTAM compression are used on any send or receive request. HPDT also requires that the session route is capable of performing data transfers directly to or from CSM buffers. Refer to z/OS Communications Server: SNA Network Implementation Guide for a description of these route characteristics. The application can inspect the session information vector to determine whether HPDT services are available for a session.

Macroinstructions used by HPDT applications

Applications request HPDT services by specifying OPTCD=XBUFLST on the following macroinstructions.

- **For sending data**
  - APPCCMD CONTROL=SEND,QUALIFY=DATA|DATACON|DATAFLU
  - APPCCMD CONTROL=DEALLOC,QUALIFY=DATACON|DATAFLU
  - APPCCMD CONTROL=PREPRCV,QUALIFY=DATACON|DATAFLU
  - APPCCMD CONTROL=SENDRCV,QUALIFY=DATAFLU
- **For receiving data**
  - APPCCMD CONTROL=RECEIVE,QUALIFY=SPEC|ISPEC
  - APPCCMD CONTROL=RECEIVE,QUALIFY=ANY|IANY

The XBUFLST keyword specifies an extended buffer list that points to data residing in CSM storage. See “Using the extended buffer list (XBUFLST)” on page 221 for more information.

Applications using HPDT must also access CSM services using the IVTCSM macroinstruction. The following types of requests are available to the application:

- Register (REQUEST=CREATE_POOL) or deregister (REQUEST=DELETE_POOL) to use a storage pool of buffers residing in ECSA or in a data space
- Get buffers of a pool (REQUEST=GET_BUFFER)
- Return buffers to a storage pool (REQUEST=FREE_BUFFER)
- Transfer ownership of buffers of a pool (REQUEST=CHANGE_OWNER)
- Copy data between buffers of any type (REQUEST=COPY_DATA)
- Create a logical image of a buffer (REQUEST=ASSIGN_BUFFER)
- Change the pageable state of a buffer (REQUEST=FIX_BUFFER and REQUEST=PAGE_BUFFER)
- Provide the address of CSM resource statistics information (REQUEST=RESOURCE_STATS)
- Provide the address of information required to dump CSM data space (REQUEST=DUMP_INFO)

For a complete description of all CSM macroinstructions, refer to z/OS Communications Server: CSM Guide. “Application responsibilities for using HPDT” on page 220 contains general guidelines for authorized applications using CSM.

Application authorization

An application must be authorized to use the following interfaces for HPDT services:

APPCCMD with OPTCD=XBUFLST
Like all other APPCCMD requests, HPDT requests under SRB control are always considered authorized. Requests under a TCB, however, must specify BRANCH=YES. VTAM prevents a TCB mode requester that has not identified itself as authorized from using HPDT services.

IVTCSM macroinstructions
All IVTCSM requests must be from an authorized application program.

For more information about authorization, refer to z/OS Communications Server: SNA Programming.

Application responsibilities for using HPDT

As system-authorized applications, all HPDT users are expected to be written so that they handle CSM storage in a responsible manner. Therefore, the application design should adhere to the following guidelines for requesting CSM services:

- Data in CSM storage should be modified only by the original requester of the buffers. The original requester is considered to be the application that obtained the storage using the IVTCSM REQUEST=GET_BUFFER macroinstruction. All other applications are considered to be *borrowers* of the buffers and must treat the data as read-only. There are possible exceptions to this rule. See "Data delivery considerations" on page 233 for more details.
- All programs directly referencing CSM storage must do so in the proper storage key. All CSM storage is allocated in key 6.

  The IVTCSM REQUEST=COPY_DATA macroinstruction allows data to be copied into or out of CSM storage. The authorized invoker can be in any key. Use of this service may reduce the impact to the application due to storage key mismatches when CSM storage must be accessed.
- An application must not reference or use CSM storage after invoking an APPCCMD send operation except under circumstances explained in "Send macroinstruction completion considerations" on page 225.
- For receive processing, unless an application is passing the storage to another process, the application is obligated to return CSM storage received over the APPCCMD API after it has completed processing the data.
- Applications using HPDT have additional responsibilities for handling confidential text in CSM storage. See "Confidential text considerations" on page 237 for more information.
- Applications must ensure that residual data in unused data areas (which MPC may send as pad characters to maintain IDAW boundary alignment) is cleared prior to transmission. See "MPC pad character considerations" on page 237 for more information.
- In general, applications should use data space instead of ECSA to ensure that more ECSA virtual storage is available to HPDT applications on the host. ECSA should be used only if there are special application requirements.
- For send requests, applications should use the storage type indicated by the session-information vector. See "Verifying the session's capabilities" on page 221 for more information.
- The application using the HPDT interface should document its use of CSM so that the installation can tune its use of CSM storage. This information should be available in application installation documentation so that necessary changes to the limits can be made prior to application installation. CSM storage limits and buffer pool related values are defined in the CSM parmlib member, IVTPRM00, as described in z/OS Communications Server: New Function Summary.

How support for HPDT is communicated between the application and VTAM

Support for HPDT is exchanged between an application and VTAM when the application opens its ACB.

VTAM informs an application about its level of support for HPDT on the LU-6.2-support-function-list vector, which is located in the access-method-support vector list. The application must examine this vector list to determine if the VTAM with which it has opened its ACB supports HPDT. For more information about using vectors and vector lists, see "Access-method-support vector list" on page 23.

Determining VTAM's support for HPDT can also be performed at assembly time by referencing the &ISTGA56 variable on the ISTGAPPC macroinstruction. Refer to z/OS Communications Server: SNA Programmer's LU 6.2 Reference for more information.

Applications inform VTAM about their intent to use the HPDT interface on the application-capabilities vector. For more information about this vector, see "Vector lists supplying information to VTAM" on page 21.

Verifying the session's capabilities

VTAM performs normal send and receive processing when an HPDT request (OPTCD=XBUFLST) is received for a session when HPDT services are not available. Under these circumstances, an application program can be written to invoke normal send or receive requests and avoid the steps required to invoke CSM. The application should check the session-information vector in the VTAM-APPCCMD vector list. This vector contains information about the session that the application can use to optimize performance for sending and receiving operations. The following fields in the session-information vector can be examined by the application:

APC19NOF
If this bit setting is on, the route used by the session does not support HPDT services. The application should either not use the HPDT interface or use a CSM data space that is marked *eligible to be pagefreed by CSM* (BLXEN_PAGEELIG flag is set).

If this bit setting is off, the application should examine the other indicators to make a determination as to what type of CSM storage should be used.

APC19SMB
If this bit setting is on, the session route uses a small maximum RU size and the performance gain for using HPDT is negligible. The application should either not use the HPDT interface or request 4 KB CSM buffers for application data.

APC19PGP
If this bit setting is on, the session is using a connection that does not require fixed buffers. No additional performance can be gained by using CSM fixed storage. The application should use storage marked eligible to be paged.

APC19FXP
If this bit setting is on, maximum performance benefits are realized by providing the data to VTAM in CSM fixed storage.

APC19RUO
This field indicates the maximum outbound RU size.

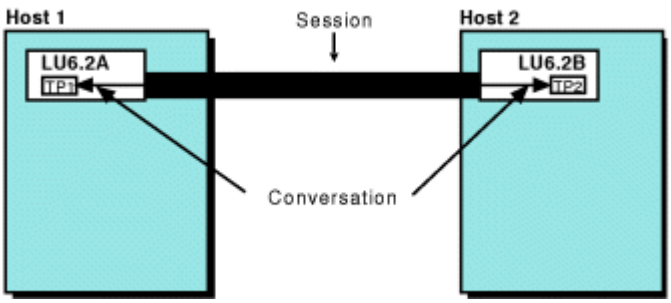
APC19RUI
This field indicates the maximum inbound RU size.

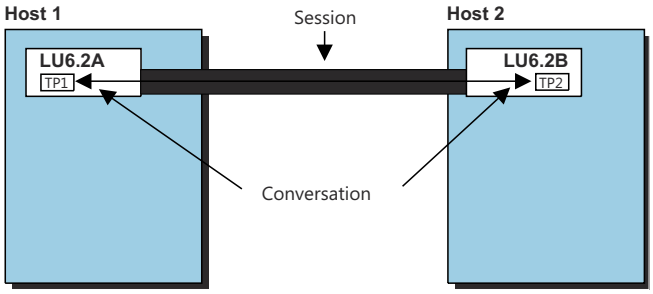
Using the extended buffer list (XBUFLST)

To send data using the HPDT interface, an application passes CSM storage to VTAM on an APPCCMD macroinstruction using the *extended* buffer list (XBUFLST). Likewise, VTAM uses the extended buffer list to pass data to an application that receives data using the HPDT interface. The HPDT interface supports all CSM storage pools, including 64-bit backed.

The extended buffer list enables application programs to use multiple CSM buffers and send them as one piece of data. This type of send is requested by the application by specifying OPTCD=XBUFLST on the APPCCMD request. The use of the XBUFLST option is transparent to the conversation partner. When the XBUFLST option is used, the AREA field points to an area of storage containing the extended buffer list. The list is made up of 48-byte entries that contain pointers to buffers in CSM.

The format of each extended buffer list entry is as follows.





Byte (hex)
Contents

00
0

01
Indicates the buffer source.

X'80'
ECSA

X'40'
Data space

02
Indicates the state of the buffers.

X'80'
Fixed

X'40'
Pageable

X'20'
Eligible to be page freed

03
0

04 - 0F
CSM token.

10 - 13
CSM data space ALET.

14 - 17
Address of data.

18 - 1B
Length of data.

1C - 1F
Length of data accepted by VTAM for a send request; the application must ensure that this field is set to 0 before issuing an HPDT send request.

20
VTAM and application flags. Bit 1 indicates whether VTAM has accepted ownership of the CSM buffer for a send request.

21 - 2F
0

The extended buffer list entry is mapped by the ISTBLXEN DSECT. For a complete layout, refer to z/OS Communications Server: SNA Programmer's LU 6.2 Reference.

Sending data using HPDT

This section explains how applications can be written to send data using the HPDT interface. The instructions in this section should be performed by an application after it has determined HPDT support as explained in "How support for HPDT is communicated between the application and VTAM" on page 220 and "Verifying the session's capabilities" on page 221. Some of these steps use the IVTCSM macroinstruction. For more information about the IVTCSM macroinstruction, refer to z/OS Communications Server: CSM Guide.

SEND processing using the HPDT interface

The following process explains how an application uses the HPDT interface to send data as shown in Figure 27 on page 224.

1. The application registers itself as a user of a CSM pool of a specific size and type by issuing the IVTCSM REQUEST=CREATE_POOL macroinstruction.
2. The application obtains buffers from where the data will be sent using the IVTCSM REQUEST=GET_BUFFER macroinstruction. The BUFLIST parameter is used to indicate the address where CSM will build a list of buffer entries. Each buffer entry is mapped by the IVTBUFL DSECT, which is 28 bytes long.
3. The application loads the data into the buffers. Using the CSM buffer list entries, the application builds an extended buffer list (48 bytes long). Each extended buffer list entry is mapped by the ISTBLXEN DSECT.
4. The application invokes VTAM to send the data by specifying OPTCD=XBUFLST on one of the following APPCCMD macroinstructions:
   - APPCCMD CONTROL=SEND,QUALIFY=DATA|DATACON|DATAFLU
   - APPCCMD CONTROL=DEALLOC,QUALIFY=DATACON|DATAFLU
   - APPCCMD CONTROL=PREPRCV,QUALIFY=DATACON|DATAFLU
   - APPCCMD CONTROL=SENDRCV,QUALIFY=DATAFLU
5. VTAM accepts data in CSM buffers from the application. VTAM issues the IVTCSM REQUEST=CHANGE_OWNER macroinstruction to accept responsibility for returning the buffers.
6. After the send is posted, the application examines the RPLXSRV field in the RPL to determine if VTAM accepted all of the buffers supplied by the application. The application is responsible for freeing any buffers that are not accepted by VTAM.
7. VTAM performs the DLC I/O directly from the application-supplied CSM buffers.
8. After VTAM has completed the transmission of the data, VTAM returns all storage to CSM. If the application has not previously requested the return of the buffers, CSM returns the storage directly to the CSM storage pools where they are available for reuse by other CSM requesters. If the application has previously requested the return of the buffers (using the FREERTN parameter on the GET_BUFFER request), CSM returns the storage to the application by scheduling an application exit routine.
9. The application removes its registration as a user of the buffer pool at application termination using the IVTCSM REQUEST=DELETE_POOL macroinstruction.

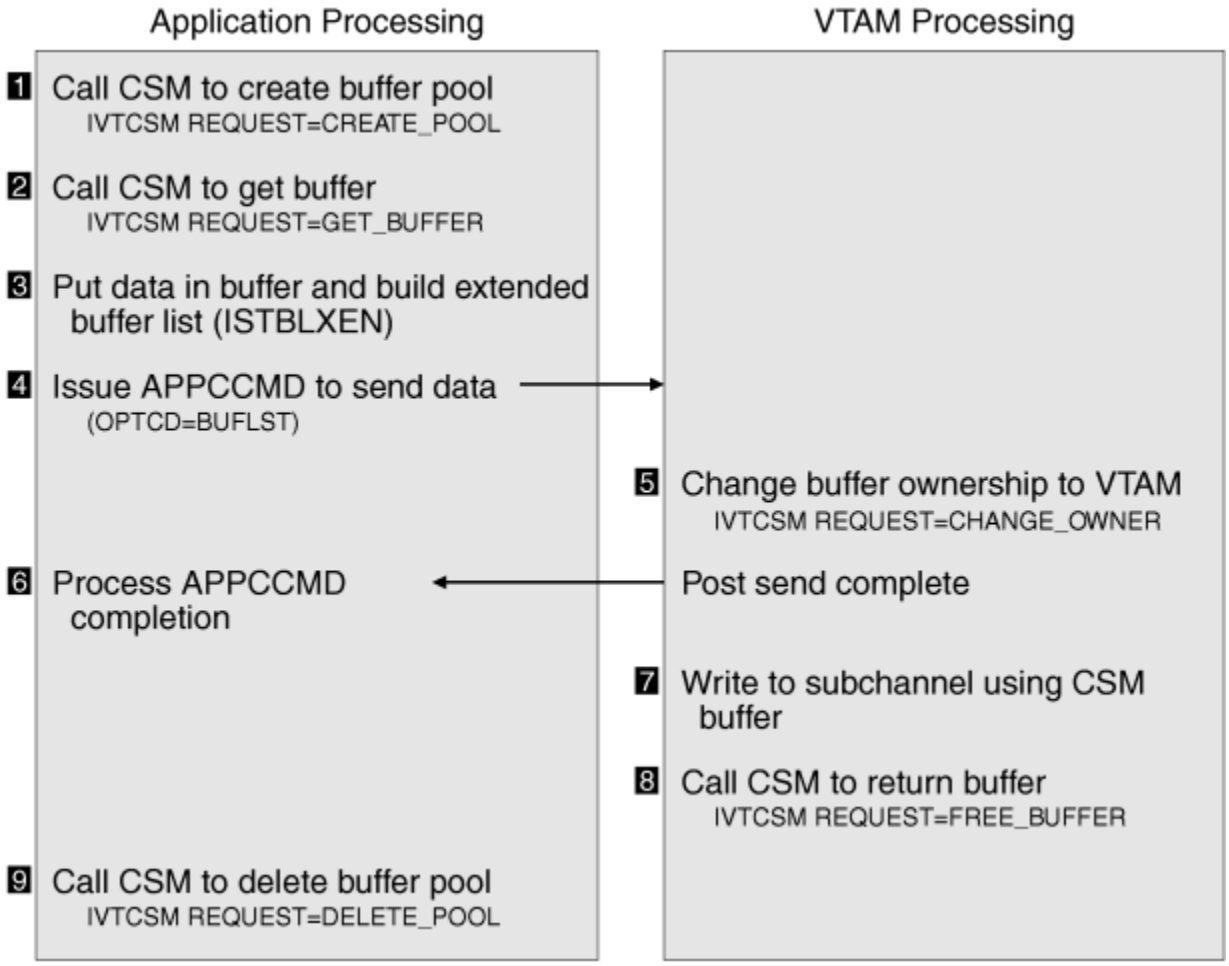


Figure 27. SEND processing using CSM buffers

Note that CSM data must not be accessed by the application after the APPCCMD API is crossed until either:

- The application's buffer return exit is scheduled with CSM buffers.

  This exit is specified when the CSM macroinstruction is issued to allocate CSM storage (IVTCSM REQUEST=GET_BUFFER). For more information, refer to z/OS Communications Server: CSM Guide
- Certain error conditions result in the send being unsuccessful.

  In these error cases, VTAM may not have accepted responsibility for the CSM storage. For these situations, the application must not handle the CSM storage until the APPCCMD function is completed.

Creating and sending multiple pieces of data in one CSM buffer

Applications using the HPDT interface must supply a buffer list that uniquely identifies each CSM buffer entry passed in the buffer list. If an application uses one CSM buffer to hold multiple pieces of data to be sent using separate buffer list entries, the application must use the IVTCSM REQUEST=ASSIGN_BUFFER macroinstruction to acquire a token to uniquely identify the buffer for each entry in the buffer list. The data address and length of data passed in each buffer list entry with the unique buffer token would represent the manner in which the application oriented the data in the buffer. VTAM frees each piece of data using the unique token provided.

For example, the IVTCSM REQUEST=ASSIGN_BUFFER macroinstruction could be used to create multiple images of a buffer in which small pieces of data that relate to larger discontiguous buffers, such as headers, are packed. The VTAM buffer list would be used to structure the ordering of the small pieces with the larger discontiguous buffers.

How VTAM processes an HPDT send request

Because HPDT is targeted for use by large packet applications that are moving large amounts of data, VTAM's processing of HPDT send requests differ from the processing of normal APPCCMD send requests. The following describes these differences.

- All HPDT send requests require that the data is flushed. If an APPCCMD CONTROL=SEND, QUALIFY=DATA macroinstruction is issued with OPTCD=XBUFLST specified, the data is flushed.
- Logical record length checking for HPDT send requests is bypassed. For normal send processing, VTAM performs logical record checking. Because the users of HPDT are authorized, they are expected to be correctly written and to have provided the data correctly. Bypassing logical record length checking saves pathlength for HPDT applications.

  Note: Logical record checking is turned off for the duration of a conversation for all sends, including non-HPDT sends, once the first HPDT send is issued.

Send macroinstruction completion considerations

This section explains possible outcomes of an HPDT send request and the associated rules for storage handling after CSM storage has been given to VTAM. See "Handling of temporary storage shortages during send" on page 227 for information regarding changes to the handling of temporary storage shortages during a send operation. The application examines RPLXSRV in the RPL to determine if the HPDT send request was successful.

Completion of a successful send

If the application does not request that the CSM storage be returned to its buffer return exit routine, the application must not access the CSM storage after the send macroinstruction has been accepted. VTAM returns the storage to the buffer pool after the send process is complete.

If the application requests that the CSM storage be returned (by using the FREERTN parameter on the GET_BUFFER request), the application must not access the CSM storage until the exit is given control. The scheduling and subsequent execution of this exit is asynchronous to the completion of the APPCCMD macroinstruction. The storage may be marked as *eligible to be pagefreed by CSM* upon return to the application's buffer return exit routine. Eligible to be pagefreed is a status maintained by CSM. The actual system state of a buffer with this status can be either fixed or pageable. The application is responsible for using CSM services to ensure the buffer is fixed if subsequent use of the storage requires that the buffer be fixed.

Prior to normal application termination, the application should wait for all buffers to be returned to the exit. For abnormal termination, CSM ensures that all CSM storage is returned to CSM.

Refer to z/OS Communications Server: CSM Guide for a discussion of the buffer return exit routine and the CSM method of managing real storage.

Completion of an unsuccessful send

Certain macroinstruction error completions occur prior to VTAM actually accepting the CSM storage or the send request. For these cases, the CSM storage is still considered owned by the application. VTAM does not invoke CSM to return the buffers. The application's buffer return exit routine, if specified, is not scheduled. The application is responsible for the eventual return of the storage to CSM using the IVTCSM REQUEST=FREE_BUFFER macroinstruction.

If VTAM indicates that it has accepted any of the CSM storage, that CSM storage is considered owned by VTAM. In this case, the handling of the CSM storage is the same as the case where the send is successful. VTAM eventually invokes CSM to return the buffers. The application's buffer return exit routine, if specified, is scheduled.

VTAM indicates whether it has accepted a CSM buffer so that the application can determine whether it is responsible for freeing the buffer. The application is responsible for checking these indicators. This indication is provided by two different methods, depending on the nature of the failure:

- If the error is in the category where only a register return code is provided to the application, the application is responsible for returning the CSM buffers. Typically, this type of error is returned when an RPL is not considered valid for setting status. For example, VTAM may find the RPL to already be in use.

  These errors, with accompanying return codes, are:

  – Logic error due to not valid RPL

    General Return Code=4
    Recovery Action Return Code=X'18'

  – Logic error due to not valid RPL extension

    General Return Code=4
    Recovery Action Return Code=X'1C'

  – Logic error due to RPL in wrong state

    General Return Code=4
    Recovery Action Return Code=X'20'

  For more information on these errors, see Chapter 14, “Handling errors,” on page 263.
- If an RPL is available to VTAM, VTAM sets the RPLXSRV flag if all of the buffers are accepted from the application. If this flag is not set, it is possible that some of the buffers have been accepted and further analysis of each buffer list entry is required.

Each buffer list entry contains a flag, BLXEN_OWNACC, which indicates whether the buffer in the list has been accepted by VTAM. If the buffer is not accepted, it is possible that VTAM has accepted part of the data. A count field in each list entry, BLXEN_RLENA, can be used by the application to determine whether the buffer in the list has been partially or completely accepted, or completely unaccepted. VTAM sets the count field as data is accepted. The application is responsible for ensuring that the initial value of the count field is 0. If the count field is equal to the application-supplied length field, VTAM has accepted all of the data. If the two fields are not equal, the application is responsible for freeing the CSM storage. The value for BLXEN_RLENA, when added to the start of the storage area, provides the address of the data area that the application can validly access. Any data prior to the address could be in use in a channel program and should not be accessed.

The following list describes the application's responsibilities for freeing storage depending on how much data in the buffer list entry is accepted.

- **None of the data in the buffer list entry is accepted (BLXEN_OWNACC = B'0' & BLXEN_RLENA = 0)**

  The application is responsible for freeing the CSM storage. All of the storage is available for reuse.
- **All of the data in the buffer list entry is accepted (BLXEN_OWNACC = B'1')**

  VTAM is responsible for freeing the CSM storage. None of the storage can be reused by the application until its buffer return exit routine, if specified, is executed.
- **Part of the data in the buffer list entry is accepted (BLXEN_OWNACC = B'0' & BLXEN_RLEN ¬= BLXEN_RLENA)**

  The application is responsible for freeing the CSM storage using the original CSM token. VTAM is also responsible for freeing the portion of the CSM storage it has accepted. VTAM represents the accepted storage with the CSM tokens it obtained using IVTCSM REQUEST=ASSIGN_BUFFER. Only the storage not accepted by VTAM can be reused by the application. This reusable storage address is calculated by adding the accepted count, BLXEN_RLENA, to the original storage address. There is no way to reuse the portion of the CSM storage accepted by VTAM without using the IVTCSM REQUEST=FREE_BUFFER macroinstruction. If the application is using a buffer return exit routine, it is not scheduled until all CSM tokens associated with the storage are freed, including the original token, using the IVTCSM REQUEST=FREE_BUFFER macroinstruction. If a CSM buffer is partially accepted by VTAM and partially owned by the application, there are at least two CSM tokens associated with the CSM storage. Each token must be returned to CSM regardless of which address space is the CSM owner before the buffer return exit routine is scheduled and the entire buffer is available for reuse. If a CSM buffer return exit is not specified, the entire CSM buffer can never be recovered by the application under the original token

because issuance of the IVTCSM REQUEST=FREE_BUFFER returns the storage directly to the CSM pool rather than to the application.

Note: In the previous discussion, there are conditions where the storage is considered reusable. This includes alterations of contents only when the sending application is the originator of the storage. All applications except the original requester must treat any CSM storage as *read-only*. See "Application responsibilities for using HPDT" on page 220 for more information.

Storage not accepted by VTAM could potentially be marked *eligible to be pagefreed by CSM* upon send completion. Eligible to be pagefreed is a status maintained by CSM. The actual system state of a buffer with this status can be either fixed or pageable. The application should examine ISTBLXEN to determine the pageable state of the buffers returned by CSM. If the BLXEN_PAGEELIG flag is set on, then the storage is marked as eligible to be pagefreed by CSM. The application is responsible for fixing the buffers (IVTCSM REQUEST=FIX_BUFFER macroinstruction) if subsequent use of the storage requires fixed buffers.

Handling of temporary storage shortages during send

After an HPDT send request has been issued, VTAM can run out of required VTAM internal storage (RCPRI = X'0098') during the processing of the application's extended buffer list. If the application is to restart the send to transmit any data that was not successfully sent, the application must first determine where in the data stream that the previous send was stopped. There are two methods available to the application to send the remaining data.

- The first method uses the accepted count field, BLXEN_RLENA, that is located in each list entry and described in "Completion of an unsuccessful send" on page 225. This method is usable regardless of the reason for the send failure. The application must adjust the value of the AREA parameter to the first entry that was not completely processed. The total length value specified for the RECLEN parameter must be reduced accordingly. The address in the buffer list entry must be adjusted to the sum of the original AREA value plus the accepted count value, BLXEN_RLENA. The length in the buffer list entry must be reduced accordingly. The accepted count field in the buffer list entry must be cleared.
- The second method uses the RPL6STBF and RPL6STDS indications in the RPL extension. This method, described in "Handling storage shortages" on page 190, can be used only in the case of temporary storage shortages.

TPEND exit considerations when sends are pending

The TPEND exit can be driven with several different return codes. For applications using HPDT, there may be CSM cleanup activity required after the TPEND exit is driven because pending RPL requests can be canceled before being completed but after some data has been transferred. At this point, some or all of the data may not have been transferred to VTAM. Any such data is still the responsibility of the application and should be freed by the application. In general, the RPL and associated buffer list for all such canceled requests must be examined for non-transferred storage. Depending on the reason for scheduling of the TPEND exit, the application has varying required actions:

- Return Code 0

  A HALT command without CANCEL or QUICK was issued. In this case, all outstanding RPL requests are completed normally. There are no unique considerations for this return code.

- Return Code 4

  A HALT NET, QUICK or VARY NET, INACT command for the application was issued. In this case, any pending RPL requests are completed with an error return. It is possible that all storage in the application buffer list was not accepted by VTAM before the pending request was canceled. Therefore, the application is responsible for freeing storage referenced by such RPLs as described in "Completion of an unsuccessful send" on page 225.

- Return Code 8

  A HALT NET, CANCEL command was issued or VTAM abnormally terminated. In this case, any pending RPL requests are not completed. It is possible that all storage in the application buffer list was not accepted by VTAM before processing on the pending request was canceled. Therefore, the application is

responsible for freeing storage referenced by such RPLs as described in “Completion of an unsuccessful send” on page 225. After the application issues CLOSE ACB from the application mainline, the application should locate any incomplete RPLs for cleanup. Note that after CLOSE ACB completion, all VTAM activity has already been quiesced with respect to processes that may be manipulating these RPLs. Therefore, the contents of RPLs and associated buffer lists are stable and can be safely examined.

It is possible that the RPLXSRV flag may not be set when all data is actually serviced. This condition could occur if the request is terminated before the CRPL is copied back into the RPL. In this case, the application checks all of the buffer list entries for the BLXEN_OWNACC flag to determine if cleanup action is required.

- Return Code 12

  This situation occurs when an application is enabled for persistent sessions. This return code occurs when an alternate application issues OPEN ACB to take over sessions owned by this application.

  In this case, any pending RPL requests are completed with an error return. It is possible that all storage in the application buffer list was not accepted by VTAM before the pending request was canceled. Therefore, the application is responsible for freeing storage referenced by such RPLs as described in “Completion of an unsuccessful send” on page 225.

Receiving data using HPDT

This section explains how applications can be written to receive data using the HPDT interface. Applications request an HPDT receive by issuing the APPCCMD CONTROL=RECEIVE macroinstruction specifying OPTCD=XBUFLST and freeing the buffers after processing the data. The following APPCCMD macroinstructions can be used to receive data using the HPDT interface:

- APPCCMD CONTROL=RECEIVE,QUALIFY=SPEC|ISPEC
- APPCCMD CONTROL=RECEIVE,QUALIFY=ANY|IANY

Applications using the HPDT interface to receive data are also responsible for performing the following actions:

- Informing VTAM of the application's intent to use the HPDT interface.

  See “How support for HPDT is communicated between the application and VTAM” on page 220 for more information.
- Specifying storage type to be used for inbound data on the XBUFLST-receive vector.
- Specifying receive completion criteria and amount of data to be received.

When XBUFLST is specified on a receive request, the application provides a storage area pointed to by the AREA input parameter. The length of the area is specified by the AREALEN parameter. VTAM builds the receive buffer list in this storage area. The buffer list points to addresses of data that reside in CSM buffers. The RECLEN value in the RPL that is returned after the APPCCMD completes indicates the length of this buffer list that VTAM builds. The data in each CSM buffer received is described in each buffer list entry. The length of the data received is the sum of the data in all buffer list entries.

RECEIVE processing using HPDT

This section summarizes the process for receiving normal application data using HPDT (see Figure 28 on page 229).

1. The application builds an XBUFLST-receive vector as described in “Passing HPDT receive requirements to VTAM” on page 229.
2. The application issues an APPCCMD CONTROL=RECEIVE. The AREA parameter specifies an area for VTAM to build a buffer list that points to data in CSM storage.
3. After data is read from subchannel to CSM storage, VTAM transfers ownership of buffers to the application and builds the extended buffer list in the application's area.
4. When the receive is posted complete, the application examines the extended buffer list in the address indicated by the AREA parameter. Each buffer list entry contains information used to address the data.

The buffer list entries are mapped as shown in "Using the extended buffer list (XBUFLST)" on page 221.

5. The application returns all storage to CSM after the application has finished processing the data. See "Receive macroinstruction completion considerations" on page 230 for complete information about the applications options.

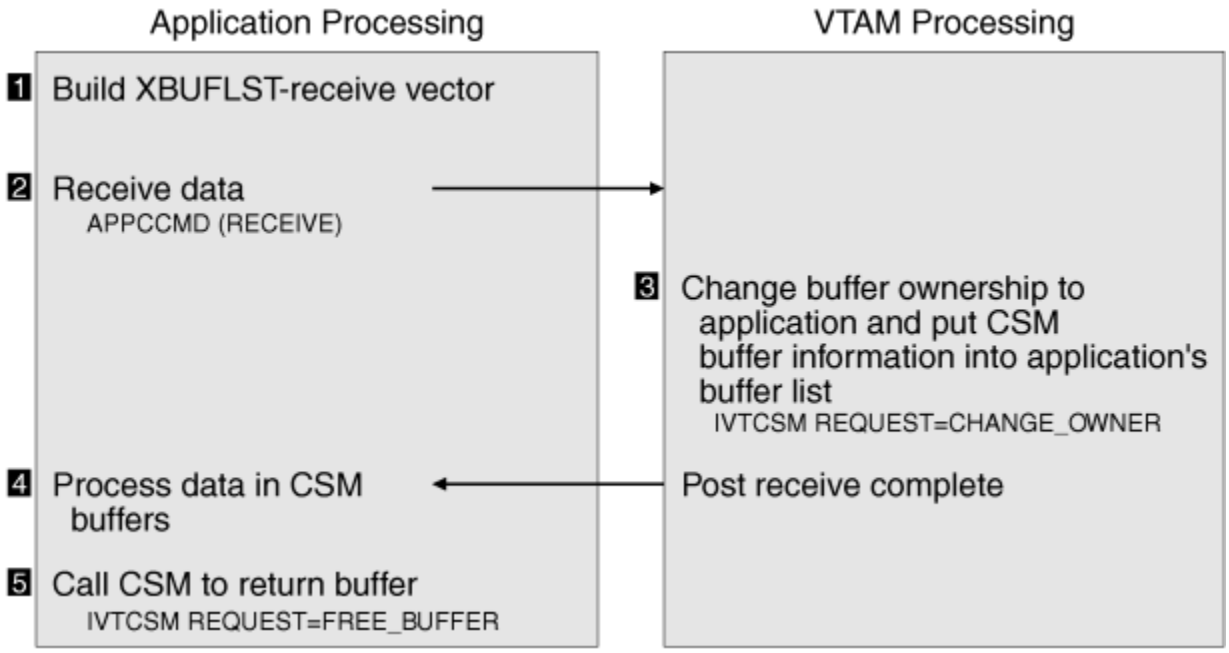


Figure 28. RECEIVE processing using CSM buffers

Passing HPDT receive requirements to VTAM

The XBUFLST-receive vector provides information to VTAM about the area in which the application will receive CSM data. It is supplied to VTAM when the APPCCMD CONTROL=RECEIVE macroinstruction is issued with the VTROUTA=*address* and VTROUTL=*length* parameters specified. The XBUFLST-receive vector includes the following fields:

Storage Type (APC82SFL)
The application can indicate its preference for the type of CSM storage that should be used for inbound data. The application must specify at least one of these storage types.

APC82ECS X'80'
Indicates that buffers in ECSA should be used for inbound data.

APC82CDS X'40'
Indicates that buffers in a data space should be used for inbound data.

A mismatch between the storage type specified on the XBUFLST-receive vector and the storage type used for channel I/O (specified by the STORAGE parameter in the TRL major node) results in a data move. For best receive performance, the application should set both storage type flags on to indicate that either storage type is acceptable.

Buffer Length (APC82XBL)
This field specifies how much data VTAM should accumulate before completing a receive when the receive is operating in FILL=BUFF mode. When operating in FILL=LL mode, the buffer length parameter is ignored. The receive completion criteria uses this buffer length value the same as it would a real buffer length when in FILL=BUFF mode and XBUFLST is not specified.

When either QUALIFY=ANY or QUALIFY=IANY is specified, the FILL parameter is not specified. Data is received in the current mode of the receiving conversation, which is determined by either the CONMODE parameter of the previous APPCCMD macroinstruction processed or by the APPCCMD CONTROL=RESETRCV. If FILL=BUFF mode is in effect and if XBUFLST is specified, a buffer length

value must be specified. For more information about conversations in FILL=BUFF mode, see “Continuation modes for receiving normal information” on page 212.

For more information about how to use the buffer length parameter, see “Controlling the amount of data received” on page 231.

Maximum Data (APC82MXD)
This is an optional field that specifies the maximum amount of data to be received. VTAM determines the amount of data to pass to the application based on one of the following settings, depending on the setting that is reached first:

- One logical record, when FILL=LL
- The size of the application's buffer area
- The maximum data value

If maximum data is not specified, VTAM provides as much data as possible to the application.

This value must be specified as a fullword (maximum data value or binary 0's).

CSM Task Association TCB (APC82TSK)
This is an optional parameter that specifies the TCB to be used by VTAM for performing CSM task association for storage being passed to the application by way of the current APPCCMD. If this parameter is not specified, the storage is associated only with the application's address space. Applications executing in cross-memory mode should use this parameter to indicate the ACB task so that VTAM can associate the CSM buffers to the appropriate task.

This value must be specified as a fullword (TCB address or binary zeros).

Receive macroinstruction completion considerations

The use of the HPDT interface does not alter the point in the VTAM receive process where the APPCCMD is considered complete. However, the use of CSM storage for application data requires some special actions and considerations that are explained in this section.

Considerations for the successful completion of a receive

If the application requires that the CSM storage be pagefixed, the application must check the BLXEN_TYPE flag in the buffer list entry, ISTBLXEN. If the buffer is pageable, the application can use the IVTCSM REQUEST=FIX_BUFFER macroinstruction to ensure that the storage is pagefixed.

If the CSM storage received is from a data space, the application can gain addressability to the storage by using the ALET that is returned in the buffer list. The application can use the IVTCSM REQUEST=COPY_DATA macroinstruction if it needs to copy the data.

If the application wishes to perform I/O directly from the data space storage, it must build the I/O structures in accordance with the ESAME mode of the machine. When the machine is in ESAME mode, 64-bit I/O structures must be used. When the machine is not in ESAME mode, 31-bit I/O structures must be used.

After processing the data, the application has the following options:

- Return the storage to CSM using the IVTCSM REQUEST=FREE_BUFFER macroinstruction.

  Note: If a logical image of a buffer has been created on an ASSIGN_BUFFER request, the buffer may not be immediately returned to the pool. CSM determines when all users sharing a CSM buffer have called CSM to return the storage. CSM maintains a use count and when all users, including the original requester, have returned the storage, the buffer is actually returned to the pool. CSM performs this function without explicit application involvement.
- Use the received data area as the storage area for a subsequent APPCCMD send operation.

  In this case, CSM storage return responsibilities are transferred to a different process. The data must not be altered because the received data area is considered a read-only copy. The original requester of

the CSM storage may have a CSM free routine outstanding and is entitled to the return of the storage with the data unaltered.

Controlling the amount of data received

The buffer length parameter (APC82XBL) in the XBUFLST-receive vector specifies how much data VTAM should accumulate before completing a receive when the receive is operating in FILL=BUFF mode. When operating in FILL=LL mode, the buffer length parameter is ignored. It is possible for a FILL=BUFF mode receive to complete with less data than the APC82XBL value and data remaining enqueued for the conversation.

The receive completion criteria uses the buffer length indicated in the XBUFLST-receive vector the same as it would on the AREALEN parameter when in FILL=BUFF mode and XBUFLST is not specified. That criteria is one of the following items:

- VTAM has internally accumulated the buffer length amount of data.
- A PS Header is received.
- A non-data indication is received as follows:
  - A SEND indication is received.
  - A CONFIRM indication is received.
  - A DEALLOCATE indication is received.
  - An ERROR has been detected.

Once the above receive completion criteria is met, the amount of data received could vary as follows:

- The data amount received is greater than or equal to the buffer length value.

  This situation occurs if the application-supplied buffer list area is large enough to reference the entire amount of data.
- The data amount received is less than the buffer length value.

  This situation occurs if:

  - The application buffer list is not large enough to reference the entire amount of data.

    In this case the buffer list area is filled.
  - VTAM encountered a resource constraint that prevented it from delivering all of the data.

    In this case the buffer list area may or may not be filled.
  - One of the non-data events that is indicated in the WHATRCV field occurred.

    In this case the buffer list area may or may not be filled. The WHATRCV field will be appropriately set.

If the application received less than buffer length, it must check the WHATRCV field for a non-data event. If a non-data event is not indicated, that means that more data is being held by VTAM. The application then should issue a new receive for the remainder of the data. A non-data event could have occurred, but it will not be indicated in the WHATRCV field until all enqueued data is received. If the application wants to immediately receive any data remainder, the buffer length parameter used on the subsequent receive should be reduced to reflect the difference of the previous buffer length value and the previous amount received. This ensures that the posting criteria is met in the event that a non-data event has not occurred. Note that the APPCCMD TESTSTAT macro can be used to determine the data amount enqueued for the conversation.

Considerations for the unsuccessful completion of a receive

In general, the application should check for the presence of data received. VTAM could have already built some buffer list entries prior to completing an APPCCMD due to an error. This situation should occur only for severe errors where no further communications on the conversation are possible. After the CSM information is placed in the application buffer list, VTAM has passed responsibility for that storage to the application. In this situation, the application is responsible for freeing this storage using CSM services. Any processing of the data prior to storage release can be done based on application requirements. The

application can determine the length of the buffer list built by examining the RECLEN parameter in the RPL.

The application should check to see if the error occurred because no CSM storage could be obtained. In this case, no data is passed to the application. This status is indicated by the RCPRI,RCSEC value of X'0070',X'0000', TEMPORARY_STORAGE_SHORTAGE_ OR_RESOURCE_SHORTAGE. If that status is present, the application could take one of the following actions:

- Reissue the APPCCMD several times as a temporary retry recovery action.
- Issue a non-HPDT receive so that the data can be copied into application private storage.
- Explicitly deallocate the conversation using APPCCMD services.

TPEND exit considerations when receives are pending

The TPEND exit can be driven with several different return codes. For applications using HPDT there may be CSM cleanup activity required after the TPEND exit is driven because pending RPL requests can be canceled before operation completion, but after some data has been transferred to the application. Depending on the reason for the TPEND exit drive, the application has varying required actions.

- Return Code 0

  A HALT command without CANCEL or QUICK was issued. In this case, all outstanding RPL requests are completed normally. There are no unique considerations for this return code.

- Return Code 4

  A HALT NET, QUICK or VARY NET, INACT command for the application was issued. In this case, any pending RPL requests are completed with an error return. It is possible that the application buffer list was completed with data before the pending request was canceled. Note that responsibility for the CSM storage is also passed to the application. Therefore, the application is responsible for freeing storage referenced by such RPLs. Even though the RPL completed with an error, the RECLEN= parameter should be examined for a nonzero value in order to determine if any storage was passed to the application prior to RPL request cancellation. Any storage found should be freed.

- Return Code 8

  A HALT NET, CANCEL command was issued or VTAM abnormally terminated.

  In this case, any pending RPL requests are not completed. It is possible that the application buffer list was completed with data before processing on the pending request was canceled. The responsibility for any CSM storage referenced by the buffer list was passed to the application. Therefore, the application is responsible for freeing storage referenced by such RPLs. After the application issues CLOSE ACB from the application mainline as is normal procedure after TPEND return code 8, the application should locate any incomplete RPLs. Even though the RPLs completed with an error, the RECLEN parameter should be examined for a nonzero value in order to determine if any storage was passed to the application prior to RPL request cancellation. Any storage found by such a search should be freed. Note that after CLOSE ACB completion, all VTAM activity has been quiesced with respect to processes that may be manipulating these RPLs. Therefore, the contents of RPLs and associated buffer lists are stable and can be safely examined.

  Note: For non-HPDT receives, RECLEN would not be set because VTAM updates the RECLEN value on an internal copy of the RPL rather than the application's RPL. RECLEN is updated only at request completion time. For HPDT receive requests, RECLEN is kept current in the application's RPL.

- Return Code 12

  This situation occurs when an application is enabled for persistent sessions. This return code occurs when an alternate application issues Open ACB in order to take over sessions owned by this application.

  In this case, any pending RPL requests are completed with an error return. It is possible that the application buffer list was completed with data before the pending request was canceled. Note that responsibility for the CSM storage is also passed to the application. Therefore, the application is responsible for freeing storage referenced by such RPLs. Even though the RPL completed with an error,

the RECLEN= parameter should be examined for a nonzero value in order to determine if any storage was passed to the application prior to RPL request cancellation. Any storage found should be freed.

Data delivery considerations

As VTAM specifies the application supplied buffer list, VTAM passes responsibility to the application for ownership of the storage. This ownership responsibility entails either ultimately freeing the storage or passing the storage (along with ownership) to another process. If the original requester of the storage specified a free routine at storage allocation time, that original requester is entitled to the return of storage without modification. Therefore, the receiving application should consider this storage as read-only storage, and should not modify the contents. Note that the storage allocation source is unknown to the receiver. Therefore, the application cannot safely make assumptions as to whether a storage return is to be performed, and consequently whether the read-only requirement exists.

It is possible that applications, if written as a cooperative set of processes, could determine that it is acceptable to modify the data if the original application does not require the original data returned unmodified. The caution here is that applications written in this manner must be able to guarantee that the original requester of the storage is one of the cooperative applications. If it is possible that the originating conversation partner can be on another node, VTAM is the requester of the storage. In this case the cooperative set of applications cannot meet the guarantee that the allocator of the storage be one of the cooperative applications.

Storage passed to the application on an HPDT receive could be copied by VTAM. Data that is not copied is most likely in the same storage as it was when VTAM first obtained the data. Therefore, the contiguous size of storage is based on how the data was obtained by VTAM. If the source was a same-host application, the contiguous size is based on the smaller of sending application buffer size and MAXRU size for the session. If the source was a DLC, the size of each contiguous piece could be based on the amount received from the network based on any number of segmentation schemes. Different arrival rates of the data segments can very likely result in the various segments being received into different CSM buffers. Data can be segmented into different sizes based on these factors. Regardless of the source, VTAM passes the data to the application in as large a contiguous piece as possible without moving the data.

If the storage passed to the application is copied, VTAM obtains a new CSM buffer in order to copy the data. The size of the CSM storage may vary.

If CSM storage is constrained when VTAM attempts to obtain new storage, it is possible that a request for CSM storage could be made that cannot be satisfied. In this case, a return indication is passed by way of the RPL (RCPRI,RCSEC of X'0070',X'0000') to indicate that CSM storage was unavailable. If this situation occurs, the application can take several possible actions.

- Reissue the APPCCMD several times as a temporary retry recovery action.
- Issue a non-HPDT receive so that the data can be copied into application private storage.
- Explicitly deallocate the conversation using APPCCMD services.

Note VTAM gives this return indication only if no storage can be obtained. If some amount of storage can be obtained, this storage is passed to the application with no error return code. This treatment allows VTAM to continue to move data through the system, even when constraint conditions exist. For FILL=LL processing, this method results in a partial LL being received. For FILL=BUFF processing, this method could result in an amount of data being passed to application that is less than the receive buffer size [3].

Using the SENDRCV macroinstruction for HPDT

The APPCCMD CONTROL=SENDRCV macroinstruction allows both a send and a receive on a single API request. Use of CONTROL=SENDRCV reduces the path length on half-duplex conversations that experience frequent changes in the direction of data flow. This function is supported for

[3] Because the application specifies a buffer list rather than an actual receive buffer, the receive buffer size is specified by the application by way of parameter.

applications using both HPDT and non-HPDT interfaces. This section describes considerations for using CONTROL=SENDRCV for HPDT requests.

The HPDT interface can be used only for the sending function of the APPCCMD CONTROL=SENDRCV macroinstruction. The application can send data directly from CSM storage using the XBUFLST option. However, data received is always copied into application storage.

When the application specifies OPTCD=XBUFLST on the CONTROL=SENDRCV request, all entries in the buffer list except the last specify the address and length of data to be sent. The data to be sent resides in CSM buffers. VTAM does not track logical records supplied by the application. Like OPTCD=BUFFLST, the last entry specifies the address and length of an area in which data is to be received. When this macroinstruction completes, another field in this last entry contains the number of bytes placed in this receive buffer by VTAM. This receive buffer is not a CSM buffer.

Figure 29 on page 234 shows the difference between the normal buffer list and the extended buffer list used on an APPCCMD CONTROL=SENDRCV macroinstruction.

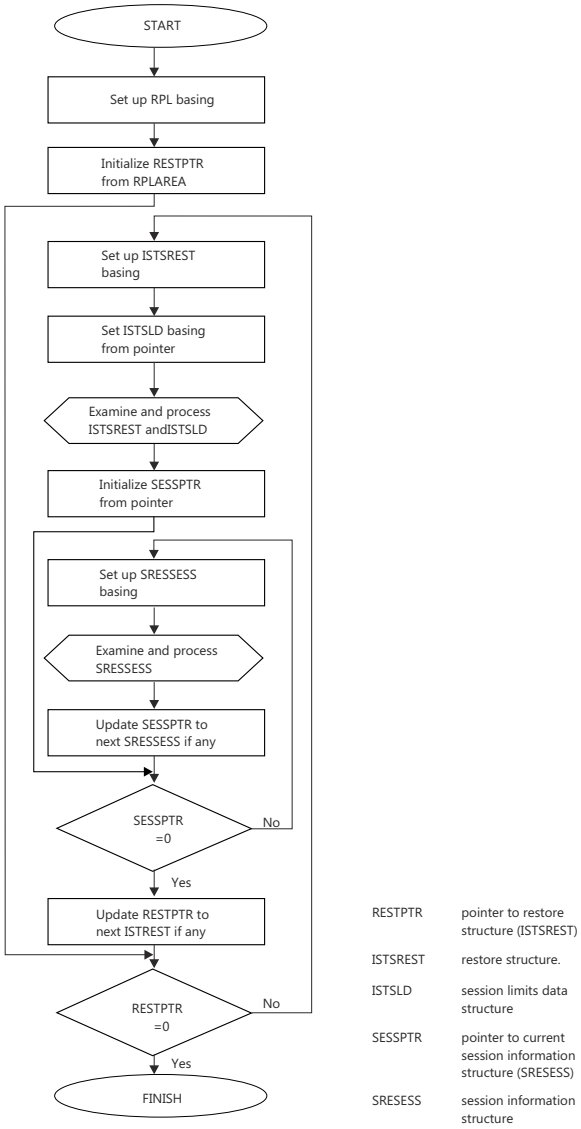


Figure 29. Comparison of BUFFLST and XBUFLST entries associated With APPCCMD CONTROL = SENDRCV (part 1 of 2)

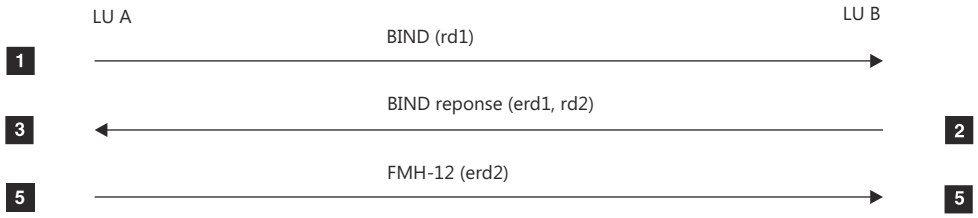


Figure 30. Comparison of BUFFLST and XBUFLST entries associated with APPCCMD CONTROL = SENDRCV (part 2 of 2)

Note that the receive portion of the extended buffer list is the same as entries in the buffer list built by the BUFFLST parameter. There can be multiple send XBUFLST entries. There is one receive BUFFLST entry.

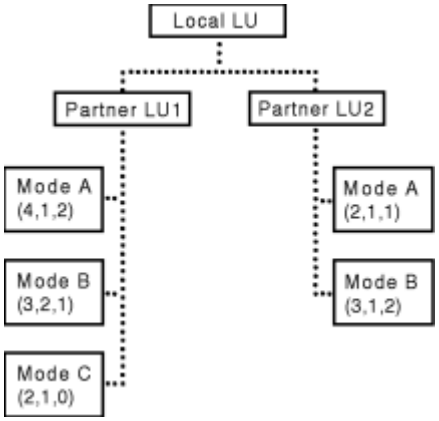
The send buffers are mapped by the ISTBLXEN DSECT and the receive buffer is mapped by the ISTBLENT DSECT. The layout of these DSECTs is shown in the z/OS Communications Server: SNA Programmer's LU 6.2 Reference.

For more information about this macroinstruction, refer to z/OS Communications Server: SNA Programmer's LU 6.2 Reference.

APPCCMD application requirements to ensure CSM storage recovery

This section describes the responsibilities of applications using the HPDT interface in order to ensure CSM storage recovery during error or abnormal termination. The basic support provided by CSM for the recovery of storage after abnormal terminations is described in z/OS Communications Server: CSM Guide. A brief summary of the types of recovery termination manager (RTM) support provided by CSM is as follows:

- Memory termination
- Jobstep task termination if not a started task (for example, batch)
- Optional task level termination
  - Abnormal termination only
  - Specified on IVTCSM requests that support the TASKID parameter



General APPCCMD storage ownership requirement

All storage provided by the application on the APPCCMD request must be associated with the ACB task or a task higher in the TCB hierarchy. This includes CSM buffers as well as private area storage that is mapped by the RPL and XBUFLST.

For applications running in cross-memory mode, prior to issuing an HPDT request, the application's primary address space must ensure that CSM storage is associated with the primary ASID and task of

the ACB. For an HPDT send, the application can use CSM services to change ownership of the buffers to the ACB task. For an HPDT receive, the application should pass the ACB task identifier to VTAM on the APC82TSK field in the XBUFLST-receive vector so VTAM can perform the ownership change (VTAM uses the ASID of the primary address).

Application responsibilities when the RPL is posted complete with an error

In the case where the RPL is actually posted, the application is responsible for determining which storage units are still the responsibility of the application. The application must process the XBUFLST to locate any storage for which VTAM did not accept ownership (for a send request), or that was already transferred to the application (for a receive request). Details of this process are described in "Send macroinstruction completion considerations" on page 225 and "Receive macroinstruction completion considerations" on page 230.

Application responsibilities when the RPL is not posted complete

There are several cases where the RPL is not posted complete after the APPCCMD is issued. These cases and the associated application responsibilities are:

- TPEND return code 8 (VTAM abnormal termination or a HALT NET, CANCEL command)

  The application must locate all pending RPLs and then must process XBUFLST to locate any storage for which VTAM did not accept ownership (for a send request) or which was already transferred to application (for a receive request). Details of this process are described in "TPEND exit considerations when sends are pending" on page 227 and "TPEND exit considerations when receives are pending" on page 232.

- Application executing in initiator-attached task that terminates

  CSM memory termination or job step task RTM locates and releases any CSM owned by the terminating application.

- Subtask of initiator-attached task abends

  If the application requires unprocessed CSM storage in pending RPLs to be automatically freed, the application must task associate the CSM as follows:

  – For TCB mode RPL requests on a send macroinstruction, use CSM Services to ensure that CSM is associated with the ACB task.
  – For TCB mode RPL requests on a receive macroinstruction, specify to VTAM the application task that is the ACB task so that VTAM can ensure that CSM is associated with that task.
  – For SRB mode RPL requests, associate CSM storage to the same TCB that the SRB is associated with.

  If the application does not require unprocessed CSM storage in pending RPLs to be automatically freed, task association is not used. RPLs could be active until the abending task has completed task termination. The application must wait for the RPL request to quiesce. Therefore, the application must locate and process incomplete RPLs at the attaching task level after task term completes.

Cross-memory considerations when the RPL is not posted complete

This section documents special responsibilities for the application running in cross-memory mode.

- An application executing in its home address space, or HASN, terminates and a request is pending under its primary ASID.

  VTAM drives LOSTERM exit with RPL. The application is responsible for doing incomplete RPL processing of CSM.

- If the application is executing in the initiator-attached task and that task terminates, CSM memory termination or jobstep TCB RTM locates and releases any CSM owned by the terminating application.
- Subtask of initiator-attached task in Primary ASID abends.

If the application requires unprocessed CSM storage in pending RPLs to be automatically freed, the application must task associate the CSM as follows:

- For TCB mode RPL requests on a send macroinstruction, use CSM services to ensure that CSM is associated with the ACB task.
- For TCB mode RPL requests on a receive macroinstruction, specify to VTAM the application task that is the ACB task so that VTAM can ensure that CSM is associated with that task.
- For SRB mode RPL requests, associate CSM storage to the same TCB that the SRB is associated with.

If the application does not require unprocessed CSM storage in pending RPLs to be automatically freed, task association is not used. RPLs could be active until the abending task has completed task termination. The application must wait for the RPL request to quiesce. Therefore, the application must locate and process incomplete RPLs at the attaching task level after task term completes.

MPC pad character considerations

MPC sends pad bytes in order to maintain 2KB IDAW alignment. The pad bytes are the bytes in CSM storage that are surrounding the session data that is being transmitted to the partner LU. Although the pad bytes are not sent to the end point destination, the pad bytes are transmitted potentially over an unsecured channel resource. Therefore, steps must be taken in order to ensure that any residual data in the CSM storage from the previous use of the CSM storage area are not transmitted over a potentially unsecured channel facility.

Both the application and VTAM share the responsibility for ensuring that residual data is not sent as MPC pad characters. In this discussion, the term *data image creator* denotes the software entity that originally allocated the CSM storage and placed data into the storage. The data image creator is responsible for ensuring that MPC pad characters are cleared. If an application is the data image creator and is sending data using the XBUFLST option, the application is responsible for clearing certain residual data in CSM storage as follows:

- For any data passed in a CSM buffer on an APPCCMD that does not end on a 2K boundary, the area after the data up to the next 2K boundary must be cleared (unless the data is known to be non-sensitive).
- If the data start is not 2K boundary-aligned, any residual data between the previous 2KB boundary and the data start must be cleared (unless the data is known to be nonsensitive).

An application resending data from CSM storage that was previously received on an APPCCMD macroinstruction should not clear residual data because it is not the data image creator. In this case, the application should copy any 2K portion of data that does not start or end on a 2K boundary, clearing pad areas as previously described.

Note: While data sent using the HPDT interface can be aligned on any storage address boundary, data aligned starting on a 2 KB boundary and sent in 2 KB increments minimizes the transmission of MPC pad characters, and therefore increases effective throughput on the channel.

Confidential text considerations

Confidential text refers to the following information:

- Unencrypted data that is ultimately encrypted by VTAM prior to transmission.
- Data carried on a session that an application has indicated should be considered confidential on the CONFTXT=YES parameter of the APPCCMD CONTROL=OPRCNTL,QUALIFY=ACTSESS macroinstruction.

On APPCCMD requests that do not use the HPDT interface, confidential text is treated specially by VTAM as follows:

- Confidential text data is not traced in the VTAM buffer trace.
- Any storage that VTAM allocates to hold confidential text is cleared after use during storage deallocation.

With HPDT, confidential text can reside in CSM storage, which can be allocated and deallocated by the application. The general rules for determining where the responsibility lies to clear CSM storage are as follows:

- If VTAM both allocates and deallocates CSM storage, VTAM ensures that the storage is cleared.
- If the application either allocates or deallocates the CSM storage, the application is responsible for clearing the storage as described in "Application data clear responsibilities" on page 238.

Application data clear responsibilities

It is the responsibility of the application to determine if data is in the class that should be cleared when freed. Knowledge of the confidential text and encryption options in effect for a session is available to applications by examining the RPLTCRYP and RPL6CFTX fields in the RPL. To clear CSM storage, the application specifies the CLEAR parameter on an IVTCSM request.

- For an HPDT send operation, the application specifies the CLEAR parameter when it obtains the buffers on the IVTCSM REQUEST=GET_BUFFER macroinstruction.
- For an HPDT receive operation, the application specifies the CLEAR parameter when it frees the buffers on the IVTCSM REQUEST=FREE_BUFFER macroinstruction.

The storage is not cleared until it is returned to the pool.

Chapter 12. Using exit routines

About this chapter

This chapter discusses the use of exit routines for LU 6.2 application programs. The complete description of the attention (ATTN) exit routine is found here, as are LU 6.2 considerations for exits that are not specific to LU 6.2.

The complete description of the exits not specific to LU 6.2, and the use of RPL exits, is found in z/OS Communications Server: SNA Programming. A description of how exit routines work under authorized path and a description of the differences between an exit routine running under a task control block (TCB) and a service request block (SRB) can also be found in z/OS Communications Server: SNA Programming.

LU 6.2 application programs use exit routines the same way that other application programs use them. The application program can supply VTAM with a list of special-purpose exit routines (EXLST exit routines) that are scheduled by the VTAM program to handle external events, such as the receipt of a session-initiation request. In addition, application programs can supply the address of an exit routine in the RPL, which VTAM schedules when the requested RPL operation completes. In all of these cases, VTAM interrupts the mainline program to allow the exit routine to execute.

As part of its LU 6.2 support, VTAM handles many external events that EXLST routines handle for other LU types or for application-implemented LU 6.2s. Consequently, not all EXLST exit routines are meaningful for VTAM LU 6.2 support. In addition, the scope of other EXLST exit routines is much more narrow. As far as LU 6.2 support is concerned, for example, VTAM schedules the SCIP exit routine on receipt only of a BIND request. All other session control requests for supported LU 6.2 sessions are handled by VTAM. The EXLST exit routines that VTAM schedules for its LU 6.2 application programs are:

- ATTN
- LERAD
- LOGON
- LOSTERM
- RELREQ
- SCIP
- SYNAD
- TPEND

The use of all these routines is optional. However, use of the ATTN, LERAD, SYNAD, and TPEND exit routines is strongly recommended. The other exit routines need to be used for LU 6.2 application programs only if the application program requires the function provided by the exit routines. Only the ATTN exit routine is unique to LU 6.2 application programs. Only the effect of exit routines on LU 6.2 application programs is discussed here. Requirements for the ATTN exit are the same as for any other EXLST exit in regards to authorized path and processing under a TCB or SRB.

Using the ATTN exit

The ATTN exit is the only EXLST exit routine unique to LU 6.2 sessions. The ATTN exit routine enables the application program to handle LU 6.2 events such as receiving notification of VTAM's negotiation of a CNOS request from a partner LU or from an operator-issued CNOS command. VTAM schedules the ATTN exit routine when any of the following events occurs:

- VTAM receives an FMH-5 for the application program.
- VTAM processes a CNOS request for the application program.
- The last LU 6.2 session, or all LU 6.2 sessions, with another LU using a given mode name group is lost.

When the application program receives control, it can determine which event has occurred by examining word 4 of the parameter list pointed to by register 1. The word contains a 4-byte string (FMH5, CNOS, or LOSS) describing the event.

Most of the information passed to the exit routine is contained in a read-only RPL and RPL extension residing in VTAM storage. The application program cannot write to this storage, but it can examine the fields in the control blocks. The address of the RPL is word 5 of the parameter list in register 1. Any other control blocks pointed to by fields in the RPL or RPL extension, such as a session limits control block, should also be treated as read-only.

Parameter list

Table 37 on page 240 shows the register contents upon entry to the ATTN exit routine.

Table 37. ATTN exit: register contents upon entry

Register	Contents
0, 2—13	Unpredictable
1	Address of parameter list • Word 1: address of the ACB for the application program • Words 2,3: reserved • Word 4: name of the event for which the exit routine is being driven • Word 5: address of read-only RPL • Word 6: reserved • Word 7: address of the network-identifier parameter list
14	VTAM address branched to when ATTN exit routine completes processing
15	Address of ATTN exit routine
Note: Register 13 does not contain the caller's save area. When the exit routine receives control, no save area is in register 13. If a user wants to issue a macroinstruction from an exit routine, a save area must be provided in register 13.	

VTAM provides a network identifier parameter list to the exit routine. Word 7 of the exit parameter list points to this list.

Table 38 on page 240 shows the contents of the list when passed to the ATTN exit routine.

Table 38. Contents of the network identifier parameter list at the ATTN exit

Bytes	Length	Contents
0 Bit 0=0 Bit 0=1 Bit 1=0 Bit 1=1 Bits 2—6 Bit 7		Flags VTAM application program is primary LU. VTAM application program is secondary LU. Primary LU is contention winner. Secondary LU is contention winner. Reserved. Reserved.
1—7	7	Reserved
8—15	8	FMH-5 and LOSS: Network identifier of the primary LU (padded with blanks, if necessary) CNOS: Network identifier of the application program

Table 38. Contents of the network identifier parameter list at the ATTN exit (continued)		
Bytes	**Length**	**Contents**
16—23	8	FMH-5 or LOSS: LU network name of the primary LU (padded with blanks, if necessary) CNOS: LU network name of application program
24—31	8	FMH-5 or LOSS: Network identifier of the secondary LU (padded with blanks, if necessary) CNOS: Network identifier of the partner LU
32—39	8	FMH-5 or LOSS: LU network name of the secondary LU (padded with blanks, if necessary) CNOS: LU network name of the partner LU

FMH-5 function

VTAM schedules the ATTN exit routine and sets "FMH5" in word 4 of the parameter list upon receiving an FMH-5 for the application program.

The application program must issue APPCCMD CONTROL=RCVFMH5 to receive the FMH-5. It need not do so in the exit routine, but at some point the application program must receive the FMH-5. More information on receiving the FMH-5 is found in “Responding to an FMH-5” on page 158.

The read-only copy of the RPL contains the length of the FMH-5 (RPL6MH5L) that has been received, the partner LU's name (RPL6LU), and the mode name (RPL6MODE) used to support the session. It also contains an indication of the security acceptance level (RPL6SECL) that is supported and whether the partner LU accepts FMH-5s with security subfields.

For more information about the parameter list that is pointed to in register 1 of the ATTN exit and for an explanation of entry procedures, refer to z/OS Communications Server: SNA Programming.

The application program can rely on ATTN alone or in conjunction with the RPL extension feedback field FMH5RCV (RPL6FMH5 in the ISTRPL6X DSECT) to receive notification of the receipt of an FMH-5. The application program issues APPCCMD CONTROL=RCVFMH5 to receive an FMH-5. In addition, the application program can maintain an internal timer and periodically issue APPCCMD CONTROL=RCVFMH5.

The ATTN exit routine has some advantages over the RPL extension field for FMH-5 notification. At times, the application program might have no conversations in progress, or it might have no APPCCMD macroinstructions outstanding. In those cases, FMH5RCV cannot notify the application program of the receipt of the FMH-5. If an ATTN exit is not provided, the application program should periodically issue APPCCMD CONTROL=RCVFMH5 to ensure that no FMH-5s are waiting.

The ATTN exit routine normally is driven only once for each FMH-5. With persistent LU-LU sessions it might be driven once for each FMH-5 for each application instance. If the application program lacks resources to start a conversation at that time, the application program can rely on RPL feedback to indicate an FMH-5 remains to be received, or maintain its own record that an FMH-5 is outstanding.

An additional difference in the two forms of notification is the reported length of the FMH-5 to be received. The length reported in the RPL extension in the ATTN exit will be the length of the FMH-5 for which the exit was driven. If a queue of FMH-5s has formed, this will not be the next one to be received. The length reported in the RPL extension feedback will be the length of the next FMH-5 to be received.

One way to avoid any problems with length is to reserve 255 bytes of storage for the FMH-5. This accommodates any possible FMH-5. The application program can check the RECLEN field of the RPL when the RCVFMH5 request completes to determine the actual size of the FMH-5 that was received.

CNOS function

VTAM schedules the ATTN exit routine with "CNOS" in word 4 of the parameter list when a CNOS request is received for the application program and is processed by VTAM. In other words, a negotiation occurs. A negotiation cannot take place for a single session. However, an ATTN(LOSS) exit can be returned for a single session that is deactivated.

The RPLAREA field in the read-only RPL points to a CNOS session limits data structure that contains the new session limits and the application program's security acceptance level. The application program can use the ISTSLCNS DSECT to map the storage. For details on the CNOS session limits data structure, see Table 17 on page 99.

For more information about the parameter list that is pointed to in register 1 of the ATTN exit and for an explanation of entry procedures, refer to z/OS Communications Server: SNA Programming.

The read-only RPL and RPL extension also contain the following fields:

RPL6LU
LU name of the partner LU

RPL6MODE
Mode name for which the session limits apply

RPLRLEN
Length of the session limits data structure

RPL6VAIA
Address of an area containing APPCCMD vector list information if any is available

When the VTAM operator issues a MODIFY CNOS command, the ATTN(CNOS) exit is used to notify both application programs in a parallel session environment. The ATTN(CNOS) exit also is used to notify only the local application program in a single session environment. When the VTAM operator issues a MODIFY DEFINE, the ATTN(CNOS) exit is used to notify the specified application program. The application program can use this information to manage CNOS requirements more effectively.

This exit will also be scheduled when VTAM issues an internal MODIFY CNOS (see “Synchronizing end points after session activation failure” on page 164 for more information).

Vectors provided for the ATTN(CNOS)

When VTAM schedules the ATTN(CNOS), VTAM may provide information about the session in the application's VTAM-APPCCMD vector list if the application has provided the address (VTRINA) and length (VTRINL) of a vector list area for the session. The following vectors may be available for the application to examine on the ATTN(CNOS) exit:

- Partner's DCE capabilities vector (X'12')
- Name change vector (X'18')
- Partner's application capabilities vector (X'1A')

For more information about vectors in the VTAM-APPCCMD vector list, see “Vector lists used during APPCCMD processing” on page 25.

LOSS function

VTAM schedules the ATTN exit with LOSS in word 4 of the parameter list to inform the application program that a session has been deactivated. The ATTN(LOSS) exit is driven every time a session is deactivated or only when the last session of a mode name group is deactivated, depending on the ATNLOSS parameter on the application program's APPL definition statement.

- If ATNLOSS=LAST, VTAM schedules the ATTN(LOSS) exit only when the last session on a mode name group is deactivated. This is the default.
- If ATNLOSS=ALL, VTAM schedules the ATTN(LOSS) exit every time a session is deactivated.

However, if the application program is a communications network management application (indicated by AUTH=CNM on the APPL definition statement), VTAM schedules the ATTN(LOSS) exit every time a session is deactivated.

The exit is scheduled even if sessions are deactivated in an orderly fashion. The exit routine is scheduled, for example, when sessions are deactivated as a result of setting the session limits to 0. The exit is scheduled only when currently active sessions are deactivated. Session activation failures do not cause the exit to be scheduled, with two exceptions. VTAM schedules the exit if the session activation fails:

- If the LU is single-session capable and has AUTOSES coded on its APPL definition statement.
- To report an RPL6LAST value of B'11'

When the exit routine is entered, the read-only copy of the RPL and RPL extension contains the LU name (RPL6LU) of the partner and the mode name (RPL6MODE) of the session. Table 39 on page 243 provides a list of the types of information provided by VTAM and the RPL6X field that contains the information.

Table 39. Information provided in the RPL6X field

Information Provided	RPL6X Field
Session ID	RPL6SSID
Session ID length	RPL6SIDL
Session completion sense code	RPL6SNSI
Session deactivation reason code	RPL6DERC
Session deactivation type	RPL6DETP
Whether the deactivated session is: • Just another session — B'00' • The last session to this LU for this particular modename — B'01' • The last session to this LU for all non-control modenames (SNASVCMG, CPSVCMG, and CPSVRMGR are control modes) — B'10' • The last session to this LU for all modenames — B'11'	RPL6LAST

To perform the safest application termination, complete the following steps:

1. Issue an APPCCMD CONTROL=OPRCNTL, QUALIFY=DEFINE macroinstruction to set session limits to 0. This macroinstruction prevents a CNOS from a partner LU from starting a session successfully.
2. Issue an APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS to set session limits to 0 for all non-control mode modenames (for example, SLCALL=X'10').
3. When RPL6LAST indicates that all non-control mode sessions have ended, issue an APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS to set SNASVCMG session limits to 0.
4. When RPL6LAST indicates that all sessions have ended, use the CLOSE macroinstruction to close the ACB.

An orderly session deactivation is indicated by:

- A sense code of 0
- A reason code of 0 (normal)
- A deactivation type of either X'01' or X'02'

Other combinations of these parameters indicate an abnormal deactivation of the session. For example, an APPCCMD CONTROL=REJECT, QUALIFY=SESSION macroinstruction can be issued to deactivate a session with either X'0F' or X'FE '. For a description of this macroinstruction, refer to z/OS Communications Server: SNA Programmer's LU 6.2 Reference.

For detailed information on deactivation types, refer to the information about UNBIND in *SNA Formats*.

Table 40 on page 244 shows the RPL extension fields that are returned for the ATTN exit. Chapter 5, "Coding the APPCCMD macroinstruction," on page 67 has information about RPL and RPL extension fields and their DSECT labels, as well as information about APPCCMD keywords.

Table 40. Fields returned in the RPL6X for the ATTN exit

RPL6X Fields	ATTN Exit (FMH5)	ATTN Exit (CNOS)	ATTN Exit (LOSS)
RPL6CBID	X	X	X
RPL6RPL	X	X	X
RPL6MH5L	X		
RPL6RMH5	X		
RPL6LU	X	X	X
RPL6MODE	X	X	X
RPL6CLSA		X	
RPL6AVFA		X	
RPL6PV		X	
RPL6SSID			X
RPL6SIDL			X
RPL6SNSI			X
RPL6DETP			X
RPL6DERC			X
RPL6LAST			X

Using other EXLST exit routines

Although the ATTN exit routine is the chief exit routine for LU 6.2 functions, other EXLST exit routines can also be useful to the application program. The LERAD, SYNAD, and TPEND exit routines, in particular, are highly recommended for handling error situations that arise. Because VTAM handles many errors for the application program, the scope of the LERAD and SYNAD exit routines, particularly SYNAD, is much narrower for LU 6.2 support.

EXLST is a declarative macroinstruction. The following example shows how this macroinstruction defines the exit routines to be used:

```
EXLST1   EXLST AM=VTAM,LOGON=LOGON1,SYNAD=SYNAD1,LERAD=LERAD1,        *
               TPEND=TPEND1,ATTN=ATTN1,SCIP=SCIP1
```

The LU 6.2 requirements for EXLST exit routines other than the ATTN exit are discussed here. For a full discussion of the EXLST macroinstruction and these exit routines, refer to z/OS Communications Server: SNA Programming.

SYNAD

This exit routine is scheduled by VTAM only for environment errors, such as VTAM being inactive or the application program closing its ACB. These situations are marked by a general return code of X'4' and a recovery action return code of X'10'. In such situations, the FDB2 field in the RPL contains a specific error return code that provides further information on the error. Several return codes have been defined for LU 6.2 support. Refer to z/OS Communications Server: SNA Programming for other FDB2 return codes.

LERAD

For non-LU 6.2 sessions, VTAM schedules the LERAD exit routine for errors causing a recovery action return code of X'14' or X'18'. VTAM schedules the LERAD for these return codes, and, in addition, defines several unique recovery action return codes for LU 6.2 support. For cases of a recovery action return code of X'14', the FDB2 field contains specific error return codes, some of which are also unique to LU 6.2 sessions.

The recovery action return codes applicable to LU 6.2 support are:

X'14'
Indicates a general logic error. In the case of LU 6.2 support, the exit is scheduled when an application program attempts to issue a non-APPCCMD macroinstruction for an LU 6.2 session and that macroinstruction is not allowed on an LU 6.2 session. Most of these macroinstructions are the instructions used to establish sessions, such as OPNDST.

X'18'
Indicates that VTAM cannot complete the request because the RPL address specified on the APPCCMD macroinstruction does not point to a valid RPL or an APPCCMD was issued asynchronously and no exit routine was provided. This return code also applies to non-LU 6.2 macroinstructions.

X'1C'
Indicates that the RPL extension is not a valid data area.

X'20'
Indicates that APPCCMD CONTROL=CHECK was issued specifying an inactive RPL, or another APPCCMD macroinstruction was issued specifying an already active RPL.

X'24'
Indicates that APPCCMD CONTROL=CHECK was issued against an RPL that contained a non-APPCCMD request type, or indicates that a non-APPCCMD CHECK macroinstruction was issued against an RPL that contained an APPCCMD request type.

TPEND

The TPEND exit routine is entered for one of the following reasons:

- The VTAM operator issues a HALT command.
- VTAM is halting itself in an orderly fashion because of an internal problem.
- VTAM is being abnormally terminated.
- The operator issues a VARY NET,INACT command for the application program.
- A VTAM application program with persistence enabled is being taken over.

The reason is indicated by a reason code that is passed in the exit routine parameter list. If a nonzero code is returned to an application with outstanding HPDT requests, the application may be responsible for freeing some or all of its CSM buffers. See “TPEND exit considerations when sends are pending” on page 227 for more information.

Reason Code
Meaning

0
A HALT command without the QUICK or CANCEL operand is indicated by reason code 0. In this case, the application program can continue communication on existing conversations, but the application program should end those communications in an orderly fashion as soon as it can. No new conversations or sessions can be established.

LU 6.2 application programs should end all conversations and then use APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS macroinstructions to set all session limits with all partner LUs to 0. (A CNOS issued for limits other than 0 is rejected.) Draining of allocation requests should not be allowed. If draining is currently in effect, turn the DRAINL bit off in the session limits data structure specified on the CNOS request. The application program can issue the CNOS request in the TPEND

exit routine. To do so, the application program must have some way of obtaining the LU names of its partner LUs.

Setting the session limits to 0 also drives the ATTN exit routine when sessions are deactivated as a result of the CNOS. The application program should wait for notification of the termination of its last session through the ATTN exit routine and then issue a CLOSE macroinstruction. The ATTN exit routine does not interrupt the TPEND exit routine. It cannot receive control until the TPEND finishes.

The CLOSE macroinstruction cannot be issued from either the TPEND or ATTN exit routine. The application program must be designed so that the mainline program issues the CLOSE. The exit routines must have a way to set a flag or to post an ECB that tells the mainline program to close the ACB. This might require the ATTN exit routine to know that VTAM is shutting down. It might be necessary for the TPEND exit routine to take some action that the ATTN exit routine can check to determine the circumstances under which it was scheduled.

4

Reason code 4 indicates that a HALT NET,QUICK command was issued or that an operator issued a VARY NET,INACT command for the application program. (Refer to z/OS Communications Server: SNA Operation for details on these operator commands.) In this case, pending RPL-based operations are canceled. The application program must not wait for the pending APPCCMDs to be completed or try to terminate conversations on an orderly basis. The TPEND should return to the mainline program where a CLOSE can be issued. A halt with a reason code of 4 still allows VTAM to continue dispatching exit routines for the application program, such as SYNAD. The TPEND should, therefore, set a flag or post an ECB to indicate to other exit routines that a shutdown is in progress.

8

When TPEND is scheduled with a reason code of 8, indicating a HALT CANCEL or abnormal VTAM termination, the exit should immediately pass control back to VTAM. When VTAM returns control to the mainline program, CLOSE should be issued. No other exit routines are scheduled, so TPEND only needs to set an indicator that the mainline program can recognize telling it to do a CLOSE.

12

Reason code 12 indicates that an application program has issued an OPEN ACB for the same ACB that the original application program has opened. The code is generated when an alternate application program takes over because the original application program must issue a CLOSE ACB, and the original application has enabled persistence.

For more information on the conditions in effect when VTAM is shutting down, refer to z/OS Communications Server: SNA Programming.

LOGON

The LOGON exit routine, if present, is scheduled when a CINIT is received from the SSCP. The CINIT might be the result of a prior APPCCMD CONTROL=ALLOC issued by the local application program, or it might be a session initiation request from a partner LU. The CINIT could also result from VTAM's automatic activation of sessions as a result of session limits being raised. One exception exists to the LOGON exit being scheduled for LU 6.2 sessions. It is not scheduled for control operator sessions that VTAM might activate.

When the LOGON exit is driven, register 1 contains the address of a parameter list. Word 5 of the parameter list contains the address of a read-only RPL. The RPLAREA field of the read-only RPL contains the address of a read-only copy of the CINIT. The RPLAAREA field points to the beginning of the control vector within that CINIT (not the RPL extension). If bytes 14 and 15 of the BIND image in the CINIT are X'0602', the CINIT represents an LU 6.2 session. If the RPLVACS bit is set on in the read-only RPL, the CINIT represents a VTAM-initiated session. For more information on the LOGON exit, refer to z/OS Communications Server: SNA Programming.

The application program can examine the CINIT and determine whether to accept or reject the session. To accept the CINIT and establish the session, the application program should issue APPCCMD CONTROL=OPRCNTL, QUALIFY=ACTSESS.

Attention: If both the local and the partner LU are the same LU, then the LOGON exit routine must be exited before the APPCCMD CONTROL=OPRCNTL, QUALIFY=ACTSESS is issued. Otherwise, the session will hang.

To reject the session, the application program must issue APPCCMD CONTROL=OPRCNTL, QUALIFY=DACTSESS. If the session is not VTAM-initiated, the application program can change the session parameters to non-LU 6.2 and accept the session with a non-LU 6.2 record API macroinstruction.

The application program uses the SETLOGON macroinstruction to regulate when the LOGON exit routine is scheduled. The SETLOGON macroinstruction functions exactly as described in z/OS Communications Server: SNA Programming.

In order to take advantage of VTAM's verification reduction function, application programs equipped with a LOGON exit routine must be able to tolerate different session information to be passed to the exit for a particular session request. For these applications, VTAM can perform verification reduction if the application informs VTAM that the exit routine can be driven more than once per session request. To indicate multiple LOGON support, applications build a vector list with the application-capabilities vector and reference the vector list on the OPEN ACB macroinstruction. For more information about the applications-capabilities vector, see "Vector lists supplying information to VTAM" on page 21.

Refer to z/OS Communications Server: SNA Network Implementation Guide for more information about verification reduction.

Application programs without a LOGON exit routine

LU 6.2 capable application programs that omit the LOGON exit routine have the PLU session establishment function performed by VTAM. VTAM's APPC support accepts the following three types of session establishment requests on behalf of the application program:

- CINITs that are the result of a prior APPCCMD CONTROL=ALLOC issued by the local application program
- CINITs that result from VTAM's automatic activation of sessions as the result of a CNOS request
- CINITs that contain an LU type of 6.2

No session establishment requests are accepted by VTAM's APPC support other than those listed above.

Note: VTAM accepts non-LU 6.2 session establishment requests, but they remain queued until you issue a non-LU 6.2 macroinstruction to process the request. Refer to z/OS Communications Server: SNA Programming for the appropriate macroinstructions.

The application program must issue SETLOGON OPTCD=START to allow VTAM to establish sessions. The function provided by SETLOGON is still needed even though the application program has no LOGON exit routine.

The set of session parameters used by VTAM in building the BIND are those contained in the CINIT along with overrides of certain parameters. See "BIND image and response" on page 124 for more information on what values are used by VTAM.

SCIP

The SCIP exit routine, if present, is scheduled when a BIND is received. If the BIND is for an LU 6.2 session, the application program must use the APPCCMD CONTROL=OPRCNTL macroinstruction to respond to the session request. Specifying QUALIFY=ACTSESS responds positively to the session request. Specifying QUALIFY=DACTSESS rejects the session request.

When the SCIP exit is driven, register 1 contains the address of a parameter list. Word 4 and word 5 of the parameter list have specific uses for LU 6.2 application programs.

Word 4 of the parameter list contains the address of the session parameters of the BIND in a format that can be examined by using the ISTDBIND DSECT.

Word 5 of the parameter list contains the address of a read-only RPL. The RPLAREA field of the read-only RPL contains the address of a read-only copy of the BIND. To examine the BIND request unit as VTAM received it, use the RPLAREA field to access the read-only copy of the BIND.

The RPLAAREA field of the read-only RPL points to the beginning of the control vectors within the BIND area (no RPL extension is provided).

The application program also can determine whether the BIND is an LU 6.2 BIND by testing the RPLVACS bit in the read-only RPL. If the bit is set on, the application program should use the APPCCMD macroinstruction to respond to the session request. For more information on the SCIP exit, refer to z/OS Communications Server: SNA Programming.

After a session is established as a VTAM-controlled LU 6.2 session, the SCIP exit routine is not scheduled again for the session. The session control requests CLEAR, RQR, STSN, and SDT are not allowed on an LU 6.2 session.

The application program uses the SETLOGON macroinstruction to regulate when the SCIP exit routine is scheduled. The SETLOGON macroinstruction functions exactly as described in z/OS Communications Server: SNA Programming.

Application programs without a SCIP exit routine

LU 6.2-capable application programs that omit the SCIP exit routine have the SLU session establishment function performed by VTAM. VTAM's APPC support accepts only those BINDs on behalf of the application program that specify an LU type of 6.2. VTAM's APPC support does not accept BINDs for LU types other than 6.2.

Note: VTAM accepts non-LU 6.2 BIND requests, but they remain queued until you issue a non-LU 6.2 macroinstruction to process the request. Refer to z/OS Communications Server: SNA Programming for the appropriate macroinstructions.

The application program must issue SETLOGON OPTCD=START to allow VTAM to establish sessions. The function provided by SETLOGON is still needed even though the application program has no SCIP exit routine.

The set of session parameters used by VTAM in building the BIND response are those contained in the BIND along with overrides of certain parameters. See “BIND image and response” on page 124 for more information on what values are used by VTAM.

LOSTERM exit routine

VTAM can schedule a LOSTERM exit routine when a session with an application program is terminated or potentially disrupted, or when a conditional terminate request for a session is received. Alternatively, for some of these conditions, an SCIP exit routine is scheduled with UNBIND, or an NSEXIT exit routine is scheduled with CLEANUP. The application program might deactivate the session as described in “Draining and session deactivation responsibility” on page 104. (If the application program deactivates the session, the LU with which the application program is, or was, in session might be unavailable for a session with any other application program. This occurs if the LU is at its session limit as a result of this session.)

If a session outage occurs, VTAM posts any outstanding requests associated with the affected session with an appropriate return code. If there are no outstanding requests, whenever the program makes the next request, it is posted with an appropriate return code.

A LOSTERM exit routine is especially recommended for an application program that does not issue specific-mode communication requests for its sessions, but is driven instead by input arriving as the result of APPCCMD CONTROL=RECEIVE macroinstructions issued in any-mode. Use of the exit routine is also recommended for an application program when there is the possibility that the LU can fill VTAM's buffers (obtained from application program storage) faster than the application program is emptying them with APPCCMD CONTROL=RECEIVE macroinstructions.

For complete information about the LOSTERM exit, refer to z/OS Communications Server: SNA Programming.

Chapter 13. VTAM's LU 6.2 security options

About this chapter

VTAM implements LU 6.2 security option sets for session-level security and data encryption. VTAM also offers pass-through support for conversation-level security for application programs that implement the conversation-level security option sets. For a complete description of the LU 6.2 security option sets, see “VTAM and security option sets” on page 36.

VTAM's session-level verification, conversation-level security, and data encryption services are functionally independent. An LU 6.2 application program can use any combination of these services.

Security management product requirements

VTAM relies on an external security management product equivalent to RACF® 1.9 or later to manage the LU-LU pair profiles, which contain the LU-LU pair session key, and to provide encryption services.

If security management functions are to be used, the following conditions must be true before an application issues OPEN ACB to identify itself to VTAM:

- The security management product must be installed and active.
- The resource class APPCLU must be active. The APPCLU class is used by RACF to verify the identity of partner logical units during VTAM session establishment.

If either of the above conditions is not met, the OPEN ACB fails with an ACBERFLAG value of X'72', indicating a security error. Security profiles are normally created using the application's network name, defining partner LUs, session security requirements, and conversations with those partners. A generic resource application can have a security profile defined with its generic resource name. If profiles are defined for both names, only the profile for the application network name is used.

For more information concerning functions provided by the external security management product and the interface between the external security management product and VTAM, refer to z/OS Security Server RACROUTE Macro Reference.

Defining profiles for LU-LU session pairs in RACF

Before activating sessions using session-level verification between LUs, define LU-LU session pair profiles in RACF. These profiles are used by RACF to determine which type 6.2 LUs can establish sessions with each other. Application programs with VERIFY=OPTIONAL session keys do not have to be defined in the profiles. For more information, see “Defining the degree of level 1 session-level verification” on page 251.

LU-LU session pairs are defined to RACF by the RDEFINE command in the APPLU resource class. The format of the LU-LU session pair profiles differs depending on whether the application uses network-qualified names, as determined by the value of the NQNAMES specification on the ACB.

- If NQNAMES=NO, then the LU-LU profiles must be defined to RACF using the following format:
  local_netid.local_lu.remote_lu
- If NQNAMES=YES, then the LU-LU profiles must be defined to RACF using the following format:
  local_netid.local_lu.remote_netid.remote_lu

For more information about how to define LU-LU session pair profiles, refer to z/OS Security Server RACF Security Administrator's Guide.

Session-level verification

Session-level verification enables an LU to verify its partner's identity every time a session between the LU-LU pair is established. When using session-level security, each LU must have access to a session key

that has been previously defined for the LU-LU pair by the network security administrator in an external security management product. Two protocols are available:

- Level 1 (also referred to as basic)
- Level 2 (also referred to as enhanced)

Session activation using level 1 session-level verification

When activating a session using Level 1 session-level verification, VTAM verifies the partner's identity by passing data in user data structured subfields on the BIND or BIND response that has been encrypted with the LU-LU pair session key. VTAM application programs are not responsible for any of the processing for session-level verification.

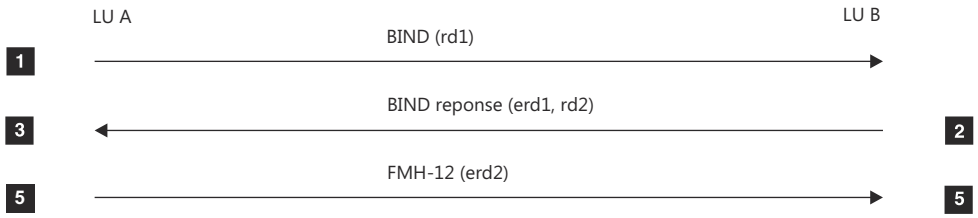


Figure 31. Information exchanged between LUs during session activation

1 VTAM 1 generates a random data field, rd1, and attaches it to the BIND in the random data structured subfield.

2 VTAM 2 receives the BIND and uses an installed security management product to encrypt rd1 using the LU-LU pair session key. The encrypted random data, erd1, is attached to the BIND response in the encrypted random data structured subfield of the BIND response. VTAM also generates a second random data field, rd2, and attaches it to the BIND response in the random data structured subfield.

The session is put into pending FMH-12 state from VTAM 2's point of view, and cannot receive any data until an FMH-12 has been received.

3 VTAM 1 receives the BIND response and uses a security management product to encrypt rd1. If the encryption result does not match erd1, the session activation is failed and a security violation is logged in the external security management product.

If the encryption result matches erd1, the SLU has verified its identity to the PLU and the session activation continues.

4 VTAM 1 encrypts rd2 and the encrypted random data, erd2, is sent to the SLU in an FMH-12. VTAM 1 also logs the successful session activation in the external security management product.

5 VTAM 2 receives the FMH-12 and encrypts rd2. If the encryption result does not match the erd2 received on the FMH-12, VTAM deactivates the session, and a security violation is logged. If the encryption result matches erd2, the PLU has verified its identity to the SLU. The session is placed in active state, and the successful session activation is logged.

Session activation using level 2 session-level verification

When activating a session using Level 2 session-level verification, VTAM verifies the partner's identity by passing data on the BIND or BIND response that has been encrypted with the LU-LU pair session key. VTAM application programs are not responsible for any of the processing for session-level verification.

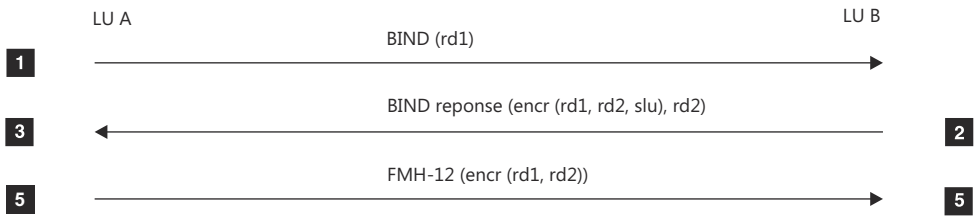


Figure 32. Information exchanged between LUs during session activation

1 VTAM 1 generates a random data field, rd1, and attaches it to the BIND in the random data structured subfield.

2 VTAM 2 receives the BIND, generates a random data field, rd2, and uses an installed security management product to encrypt the following items into one field:

rd1
Received in the BIND

rd2
Generated

slu
Fully qualified SLU name placed on the BIND response

The encrypted data (rd1, rd2, and slu) is shown here as one unit: (encr (rd1, rd2, slu)). This encrypted data is attached to the BIND response in the encrypted random data structured subfield of the BIND response. VTAM 2 also generates a second random data field, rd2, and attaches it to the BIND response in the random data structured subfield.

The session is put into pending FMH-12 state from VTAM 2's point of view and cannot receive any data until an FMH-12 has been received.

3 VTAM 1 receives the BIND response and uses a security management product to encrypt rd1, rd2, and slu. If the encryption result does not match the encrypted data (encr(rd1, rd2, slu)) received on the BIND response, the session activation is failed, and a security violation is logged in the external security management product.

If the encryption result matches (encr(rd1, rd2, slu)), the SLU has verified its identity to the PLU, and the session activation continues.

4 VTAM 1 encrypts rd1 along with rd2, and this encrypted data, (encr(rd1 and rd2), is sent to the SLU in an FMH-12. VTAM 1 also logs the successful session activation in the external security management product.

5 VTAM 2 receives the FMH-12 and encrypts rd1 along with rd2. If the encryption result does not match the encrypted data (encr(rd1 and rd2) received in the FMH-12, VTAM 2 deactivates the session, and a security violation is logged. If the encryption result matches (encr(rd1 and rd2), the PLU has verified its identity to the SLU. The session is placed in active state, and the successful session activation is logged.

Enabling session-level security

Application programs are not responsible for any of the processing needed to implement session-level verification. However, application programs must be defined to use session-level verification and an external security management product must be available to provide the security related services.

Defining the degree of level 1 session-level verification

Session-level verification is specified differently for LU-LU and CP-CP sessions:

LU-LU sessions
Code the VERIFY parameter on the APPL definition statement.

CP-CP sessions
Code the VERIFYCP start option.

Coding the APPL definition statement

The VERIFY parameter on the application program's APPL definition statement indicates the degree of Level 1 session-level verification it requires.

The valid values on the VERIFY parameter are:

NONE
The application program does not support session-level verification. Sessions with this LU will not be activated using session-level verification.

OPTIONAL
If a session key is defined in the LU-pair profile, all sessions between the LU-LU pair must be activated using session level verification. If no session key is defined in the LU-LU pair profile, sessions between the LU-LU pair are not activated using session-level verification.

REQUIRED
All sessions with this LU must be activated using session level verification.

Session-level LU-LU verification byte

After the application program has issued OPEN ACB, the application program can determine the VERIFY value on its APPL definition statement by examining the session-level verification byte in the LU 6.2 APPL definition vector in the resource-information vector list.

For more information on the resource-information vector list, refer to z/OS Communications Server: SNA Programming.

Coding the VERIFYCP start option

For CP-CP sessions, you can indicate the degree of Level 1 session-level verification with the VERIFYCP start option. The valid values are:

- NONE
- OPTIONAL
- REQUIRED

For descriptions of these values, refer to z/OS Communications Server: SNA Resource Definition Reference.

Defining the degree of level 2 session-level verification

Level 2 session-level verification is specified differently for LU-LU and CP-CP sessions:

LU-LU sessions
Code the SECLVL parameter on the APPL definition statement.

CP-CP sessions
Code the SECLVLCP start option.

Coding the APPL definition statement

The SECLVL parameter on the application program's APPL definition statement indicates the degree of Level 2 session-level verification to be used if session-level verification is active.

Some products allow only Level 1 session-level security and some products allow Level 1 or Level 2 session-level security.

Products that allow both levels fall into two classes:

1. Free choice of Level 1 or Level 2 (VTAM is in this class).
2. Restricted choice. Once Level 2 is used for a particular partner, only Level 2 can be used to that partner from then on.

Level 1 is useful in a VTAM that can be backed out to an earlier level of VTAM that can only use Level 1 session-level security. This will prevent the class 2 products (restricted choice) from locking out subsequent sessions to the earlier release of VTAM.

SECLVL=ADAPT is useful in a VTAM that will communicate with either of the following items:

- VTAMs that can use only Level 1 session-level security
- Class 2 products

Level 2 is useful when all the communicating products are capable of Level 2 session-level security to insure that an attempt at penetration, using the weaker Level 1 protocols, will be prevented.

The valid values on the SECLVL parameter are:

LEVEL1
VTAM uses the Level 1 version of the session-level protocol. If the partner LU does not support the Level 1 version, VTAM rejects the session with a sense code of X'080F0002', which indicates a session-level verification protocol mismatch. If you specify LEVEL1, VTAM will not use the Level 2 version of the session-level protocol.

ADAPT
The application program accepts either the Level 2 or Level 1 version of the session-level verification protocol, depending on the level supported by the partner LU. VTAM attempts to use the Level 2 version but allows the use of the Level 1 version if the partner LU does not support the Level 2 version.

LEVEL2
VTAM uses only the Level 2 version of the session-level protocol. If the partner LU does not support the Level 2 version, VTAM rejects the session with a sense code of X'080F0002', which indicates a session-level verification protocol mismatch.

For LU 6.2 sessions to use Level 2 session verification, each application program must be running under VTAM or under another product that supports Level 2 session verification.

For each LU 6.2 application program, decide between the following alternatives:

- Allow LU 6.2 sessions with partner LUs, only if they support Level 2 session verification.
- Allow LU 6.2 sessions with partner LUs, regardless of whether they support Level 2 session verification.

If you choose the first alternative, you specify the SECLVL=LEVEL2 operand, which means that Level 2 session verification is used between LU 6.2 application programs that support Level 2 verification, and sessions are not allowed with partner LUs that do not support Level 2 verification.

If you choose the second alternative, you specify the SECLVL=ADAPT operand, which means that Level 2 session verification is used between LU 6.2 application programs that support Level 2 verification, and the earlier level of session verification is used when the partner LU does not support Level 2 verification.

If communicating with a Class 2 product and this level of VTAM can be backed out and replaced by a VTAM that does not support Level 2, then Level 1 must be specified.

Coding the SECLVLCP start option

For CP-CP sessions, you can indicate the degree of Level 2 session-level verification with the SECLVLCP start option. The valid values are:

- LEVEL1
- ADAPT
- LEVEL2

For descriptions of these values, refer to z/OS Communications Server: SNA Resource Definition Reference.

Informing the application program of verified sessions

Although the application program plays no direct role in activating the session, it might need to determine whether a session has been activated using session-level verification.

On completion of the APPCCMD CONTROL=ALLOC and APPCCMD CONTROL=RCVFMH5 macroinstructions, VTAM sets the SLS field (RPL6SLS) in the RPL6 to indicate whether or not the session allocated to the conversation has been activated using session-level verification.

Session activation failures

Session activations can fail due to a variety of session-level verification related reasons. For example, the two LUs may have incompatible VERIFY parameters coded on their APPL statements, the LUs may have different session keys defined in their LU-LU pair profiles, or a session key might be changed in the middle of a session activation.

When a session activation fails due to a session-level verification error, VTAM deactivates the session using a sense code of X'080F6051' on either an UNBIND or a negative BIND response, as needed.

VTAM also creates an SMF Type 80 record in the external security management product, which causes the security product to issue messages to the network security administrator. (For more information on the security product, see the *Resource Access Control Facility (RACF) Security Administrator's Quick Reference*.) VTAM sets the following reason codes in this log record:

Hex Code
Description

00
Partner LU successfully verified.

01
Partner LU not verified (but session activated).

02
Session key expiration warning.

03
The security manager locked the profile.

04
The profile contains a session key that is not valid.

05
Partner LU rejected the session due to a security related error.

06
Local LU was defined with VERIFY=REQUIRED session-level LU-LU verification, but no session key exists for the local LU; or no random data field was in the BIND; or the partner LU is the PLU requesting the session but is not using session-level LU-LU verification.

07
Session-level LU-LU verification data for the session between the local LU and the partner LU matched the data for an outstanding session activation.

08
Local LU was defined with optional verification, and a session key was defined for the profile, indicating that session-level LU-LU verification is necessary. Partner LU requested a session without verification.

09
Local LU was defined with optional verification, and no session key was defined for the profile, indicating that session-level LU-LU verification should not be used. Partner LU requested a session with verification.

0A
Protocol violation.

0B
Profile was changed during session activation.

0C
Session key for the profile expired.

0D
Local LU was defined to use only the Level 2 protocol (SECLVL=LEVEL2 is specified on the APPL definition statement). Partner LU does not support the Level 2 protocol.

VTAM also issues message IST1213I when a session activation fails due to session-level verification errors. For a description of this message, refer to z/OS Communications Server: SNA Messages.

VTAM's support for session-level verification

An application program can determine the installed VTAM's support for session-level security when the application program is assembled and again after the application program has issued OPEN ACB.

To determine VTAM's support for session-level verification when the application program is assembled, the application program can reference the variable &ISTGA07, which is created by the ISTGAPPC macroinstruction. For more information, refer to z/OS Communications Server: SNA Programmer's LU 6.2 Reference.

After the application program has started and completed issuing OPEN ACB, the function-list vector of the access-method-support vector can be checked to determine the installed VTAM's level of support for session-level verification. For information about the access-method-support vector list, see "Access-method-support vector list" on page 23.

Conversation-level security

Typically, conversations are associated with an end user or a single transaction being performed by an end user. The conversation-level security option sets allow the application program to verify the identity of the end user. By contrast, session-level verification is used to verify the identity of the partner LU.

Verifying end users using conversation-level security

When an application program allocates a conversation to satisfy a request from an end user, it builds an FMH-5 to represent the conversation and then issues an APPCCMD CONTROL=ALLOC macroinstruction. The FMH-5 can contain information about the end user, which VTAM passes to the partner application program. It is the application program's responsibility to coordinate the usage of this information. VTAM checks the FMH-5 only to verify that the FMH-5 conforms to the architected format.

The conversation-level security related fields in the FMH-5 are:

- Security access subfields (mapped by FM5ACCSE DSECT; refer to z/OS Communications Server: SNA Programmer's LU 6.2 Reference):
  - User ID
  - Password
  - Profile
- Already verified indicator (FM5UIDAV)
- Persistent-verification indicators:
  - Sign-On indicator (FM5PV2)
  - Signed-On indicator (FM5PV1)

For a complete description of the format of the FMH-5 and the valid combinations of these fields, see "FMH-5 fields" on page 151.

Security acceptance levels

An application program's security acceptance level indicates which combinations of conversation-level security fields it accepts on FMH-5s it receives from partner LUs.

The security levels are established when the first session between an application program and a partner LU is established. This session can be the first between the application program and an LU after the application program is started, or it can be the first new session started after all active sessions with the partner LU have been terminated. For example, the application program can specify security information on a CNOS request only if the request initializes session limits, either for the first time or after session limits have been reset to 0 and after active sessions have been terminated. VTAM ignores the security acceptance information included on a CNOS request when sessions exist with the partner LU.

The security acceptance level applies to all modes with a given partner LU. For example, if the first CNOS with a partner LU involved a mode name of EXAMPLE1, the security information established with that CNOS would apply to all other modes with that partner LU, not just EXAMPLE1. The security level acceptance indicator cannot be changed until all session limits with a given partner LU have been set to 0 and all active sessions have been terminated.

An LU can support different levels of conversation-level security with each partner LU. For example, LUA might indicate to LUB that it supports the already-verified level of security and then indicate to LUC that it supports only the conversation level of security.

The partner LUs do not have to accept the same level of security. LUA might accept the already-verified level of security on conversation requests from LUB. LUB might accept only the security access subfields of security on conversation requests.

VTAM's level of support

VTAM provides five levels of support for security access subfields and indicators:

NONE
Security access subfields are not allowed on the FMH-5. This is the lowest level of security and is the default value.

CONV
Security access subfields are allowed on the FMH-5. If a user ID is supplied on the FMH-5, a password subfield must also be supplied.

ALREADYV
Security access subfields are allowed on the FMH-5. If a user ID is supplied, either a password subfield must be supplied, or the already-verified indicator must be set.

PERSISTV
Specifies that this application program supports security access subfields, like CONV, and accepts persistent-verification indicators in the FMH-5s that it receives from partner LUs. If the sign-on bit is set, user and password subfields are required. If the signed-on bit is set, the password subfield should not be specified.

AVPV
Specifies that this application program supports security access subfields, such as CONV, and accepts the already-verified indicator and the persistent-verification indicators in the FMH-5s that it receives from partner LUs. If the sign-on bit is set, user and password subfields are required. If the signed-on bit is set, the password subfield should not be specified.

Already-verified support

The already-verified indicators are used when a conversation request is being passed from one application program to another. The LU that receives the FMH-5 can use the user ID and password to validate the conversation request. If the conversation request is valid, the LU can remove the password field from the FMH-5 and pass the conversation request to another LU. This communication is shown in Figure 33 on page 256.

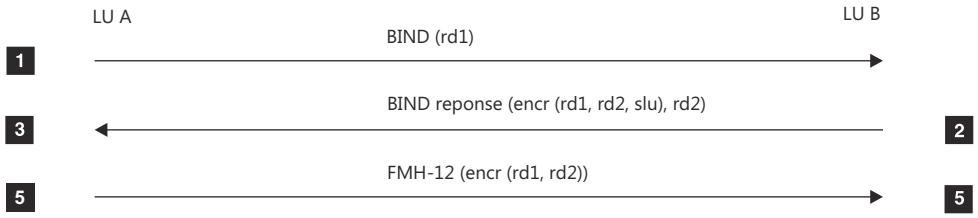


Figure 33. Example of already-verified support processing

1. LUA requests a conversation with LUB.
2. LUB uses the user ID and password to validate the conversation request.
3. As a result of the conversation request from LUA, LUB requests a related conversation with LUC. Because LUB has already verified the user ID, the FMH-5 sent to LUC contains the user ID and the already-verified indicator.

Persistent-verification support

Persistent verification enables two LUs to verify the initial conversation on a session and to assume that future conversations are verified for the duration of the session or until the user ID is signed off. This communication is shown in Figure 34 on page 257.

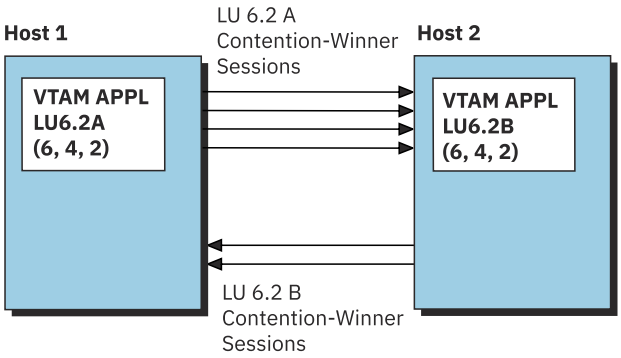


Figure 34. Example of persistent-verification support processing

1. LUA requests a conversation with LUB. The FMH-5 includes a user ID, a password, and an indication that the user ID is signing on to LUB.

   When the current transaction completes, the conversation is deallocated normally.
2. At a later time, LUA requests another conversation with LUB for the same user ID. This time, the FMH-5 contains the user ID and the already-signed-on indicator.

The application's maximum security acceptance level

The application program can determine the partner's support for security subfields and indicators. It must also specify to VTAM its level of support for the subfields. The following list shows three methods for specifying support.

- Alternate BIND

  After the application program's optional LOGON or SCIP exit is driven, it must issue APPCCMD CONTROL=OPRCNTL, QUALIFY=ACTSESS to activate the session. The application program can specify an alternate BIND using the AREA field. Three bits on the alternate BIND indicate the application program's security acceptance level. The bit settings for each security level are shown in Table 41 on page 257.

Table 41. Application program's security acceptance level, alternate BIND

Security Level	**BIND Bits**		
	BINCLSS	BINAVFS	BINPV
NONE	0	0	0
CONV	1	0	0
ALREADYV	1	1	0
PERSISTV	1	0	1
AVPV	1	1	1

Note: When an APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS macroinstruction causes a session to be established on the SNASVCMG mode, VTAM intercepts the LOGON exit, and the application program cannot specify an alternate BIND. For those CNOS requests that do not invoke negotiation, such as a CNOS with a partner that VTAM knows to be single-session capable, the application program can specify security acceptance information on the initial session. In all cases, the SCIP exit is driven for the

initial session so the application program can supply an alternate BIND response for a CNOS-initiated session.

- CNOS Session Limits Structure

  The application program can also specify the security acceptance level by setting 3 bits on the CNOS session limits structure (ISTSLCNS) when it issues APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS. Table 42 on page 258 shows the bit settings for each security level.

Table 42. Application program's security acceptance level, CNOS

Security Level	**CNOS Bits**		
	SLCLCONV	SLCLAVFA	SLCLPV
NONE	0	0	0
CONV	1	0	0
ALREADYV	1	1	0
PERSISTV	1	0	1
AVPV	1	1	1

- SECACPT Operand on the APPL Definition Statement

  If the security acceptance level information is not supplied on either the CNOS session limits structure or an alternate BIND, the SECACPT operand on the application program's APPL definition statement is used.

  For information on how to code the SECACPT operand, refer to z/OS Communications Server: SNA Resource Definition Reference.

Note: If a security management product equivalent to RACF 1.9.1 or greater is installed, it can limit the application's maximum security acceptance level. For a complete description of how the security management product can override this setting, refer to z/OS Security Server RACROUTE Macro Reference.

Partner application's maximum security acceptance level

Before using such security features, the application program must know what security level its potential conversation partners can support. VTAM determines the partner's level of support for security when the first session with a partner LU is established. VTAM returns this information to the application program in the RPL extension. This information is returned when an APPCCMD CONTROL=ALLOC or APPCCMD CONTROL=OPRCNTL, QUALIFY=DISPLAY macroinstruction completes and activates the first session between the two LUs. Three bits in the RPL extension indicate the partner LU's security acceptance level. The bit settings are shown in Table 43 on page 258.

Table 43. Partner LU's security acceptance level

Security Level	**RPL6X Bits**		
	RPL6CLSA	RPL6AVFA	RPL6PV
NONE	0	0	0
CONV	1	0	0
ALREADYV	1	1	0
PERSISTV	1	0	1
AVPV	1	1	1

To determine the partner LU's support for conversation-level security, the application program issues APPCCMD CONTROL=OPRCNTL, QUALIFY=DISPLAY and examines 3 bits in the returned session limits display (ISTSLD) structure. The bit settings for each security level are shown in Table 44 on page 259.

Table 44. Partner LU's support for conversation-level security			
Security Level	**ISTSLD Bits**		
	SLDPCLSA	SLDPAVFA	SLDPPV
NONE	0	0	0
CONV	1	0	0
ALREADYV	1	1	0
PERSISTV	1	0	1
AVPV	1	1	1

Note: The application program can issue APPCCMD CONTROL=OPRCNTL, QUALIFY=DISPLAY before the first session with the partner LU is activated and before the partner LU's support for conversation-level security is determined. In this case, the bits in the session limits display structure are not valid. Therefore, when the application program issues APPCCMD CONTROL=OPRCNTL, QUALIFY=DISPLAY, it first must check a bit in the session limits display structure, SLDCSLV, to determine whether the bits that indicate the partner's security acceptance level are valid.

Specifying a conversation's security level

When requesting a conversation, the application program is responsible for building an FMH-5 and including the security related subfields in it.

VTAM ensures that the security level in the FMH-5 is not higher than the partner LU's maximum security acceptance level. If the FMH-5 uses a conversation-security level higher than what the partner LU accepts, VTAM attempts to downgrade the FMH-5 to the appropriate security level for the partner LU.

For example, if an application program builds an FMH-5 with the persistent-verification indicators set, but the partner application accepts only conversation-security levels up to already verified, VTAM resets the persistent-verification indicators.

VTAM also examines the FMH-5 and attempts to optimize the security level. For example, an application program could build an FMH-5 that has the user ID and password security access subfields and has the already-verified indicator set. If the partner application supports the ALREADYV level of conversation-level security, VTAM removes the password subfield from the FMH-5 before sending it to the partner LU. If the partner LU supports only the CONV level, the already-verified indicator is reset before the FMH-5 is sent to the partner LU.

Data encryption

VTAM implements LU 6.2 option sets to encrypt and decrypt data before it is sent and after it is received on LU 6.2 sessions.

Levels of data encryption

VTAM offers three levels of data encryption:

NONE
Data encryption is not requested.

Note: VTAM may be capable of providing encryption even though it is not requested.

SELECTIVE
Data encryption is available. The application indicates which data it wants to have encrypted.

REQUIRED
All data on the session must be encrypted.

Determining a session's data encryption level

The data encryption level is negotiated in the BIND and BIND response for each session. The session's data encryption level is determined by the ENCR parameter on the LUs' APPL definition statements, the ENCR parameter on the logmode table entry, and the MODIFY ENCR operator command.

For more information about coding the ENCR parameter, refer to z/OS Communications Server: SNA Resource Definition Reference. For more information about the MODIFY ENCR command, refer to z/OS Communications Server: SNA Operation. For additional information about session level cryptography, refer to z/OS Communications Server: SNA Programming.

The partner LU can negotiate the data encryption to a higher level, but it cannot negotiate it to a lower level. On completion of either an APPCCMD CONTROL=ALLOC, or APPCCMD CONTROL=RCVFMH5 macro, the RPL6CRYP field of the RPL6 will indicate the data encryption level of the session allocated to the conversation.

Table 45 on page 260 shows the selection process that VTAM uses to establish the session level of cryptography, based on the values coded for the primary LU, the secondary LU, and the logon mode table entry.

Note: The cryptographic requirements specified on the VTAM APPL definition statement or VTAM operator MODIFY ENCR command for an LU and the logon mode table entry are compared. The higher of the cryptographic levels is used.

Table 45. Level of cryptography for LU 6.2 cryptographic sessions

Primary LU, from VTAM Definition or VTAM Operator Command (See Note)	Secondary LU, from VTAM Definition or VTAM Operator Command (See Note)	Logon Mode Table Entry	Level of Cryptography Used for Session
Required	Required	Required Selective None	A required session is established.
	Selective	Required Selective None	
	None, but capable of cryptography	Required Selective None	
	None, and not capable of cryptography	Required Selective None	The request for session establishment fails.

Table 45. Level of cryptography for LU 6.2 cryptographic sessions (continued)

Primary LU, from VTAM Definition or VTAM Operator Command (See Note)	Secondary LU, from VTAM Definition or VTAM Operator Command (See Note)	Logon Mode Table Entry	Level of Cryptography Used for Session
Selective	Required	Required Selective None	A required session is established.
	Selective	Required	A required session is established.
		Selective None	A selective session is established.
	None, but capable of cryptography	Required	A required session is established.
		Selective None	A selective session is established.
	None, and not capable of cryptography	Required Selective None	The request for session establishment fails.
Conditional	Required	Required Selective None	A required session is established.
	Selective	Required Selective None	
	None, but capable of cryptography	Required Selective None	
	None, and not capable of cryptography	Required Selective	The request for session establishment fails.
		None	A session is established without encryption.

Table 45. Level of cryptography for LU 6.2 cryptographic sessions (continued)

Primary LU, from VTAM Definition or VTAM Operator Command (See Note)	Secondary LU, from VTAM Definition or VTAM Operator Command (See Note)	Logon Mode Table Entry	Level of Cryptography Used for Session
Optional or no specification	Required	Required Selective None	A required session is established.
	Selective	Required	A required session is established.
		Selective None	A selective session is established.
	None, but capable of cryptography	Required	A required session is established.
		Selective	A selective session is established.
		None	A session is established without encryption.
	None, and not capable of cryptography	Required Selective	The request for session establishment fails.
		None	A session is established without encryption.

Selective data encryption

The application program specifies whether data is encrypted by using the keyword CRYPT=YES or CRYPT=NO on the APPCCMD macroinstruction. YES means to encrypt the data before it is sent. This keyword applies to the following macroinstructions:

- APPCCMD CONTROL=DEALLOC, QUALIFY=DATACON (HDX only)
- APPCCMD CONTROL=DEALLOC, QUALIFY=DATAFLU
- APPCCMD CONTROL=PREPRCV, QUALIFY=DATACON (HDX only)
- APPCCMD CONTROL=PREPRCV, QUALIFY=DATAFLU (HDX only)
- APPCCMD CONTROL=SEND, QUALIFY=DATA
- APPCCMD CONTROL=SEND, QUALIFY=DATACON (HDX only)
- APPCCMD CONTROL=SEND, QUALIFY=DATAFLU

If a session is not using selective or required cryptography and CRYPT=YES is specified in any of the preceding macroinstructions, VTAM issues the return code RCPRI=002C, RCSEC=0028, which means that cryptography is not allowed.

Note: If any of the data in the SEND buffer is to be sent encrypted (as indicated by the application program on the SEND, PREPRCV, or DEALLOC macroinstruction), all data in the RU is encrypted. (The buffer size is determined by the user's maximum RU setting.)

Chapter 14. Handling errors

About this chapter

This chapter discusses how to analyze feedback from the VTAM program for errors and special conditions associated with the APPCCMD macroinstruction. For a general discussion of error-handling for RPL-based macroinstructions and for information on OPEN/CLOSE errors, errors in manipulative macroinstructions, or software error analysis that is operating system dependent, refer to z/OS Communications Server: SNA Programming.

General sequence of error checking

As with any RPL-based VTAM macroinstruction, error information for the APPCCMD macroinstruction can be returned in two stages if the macroinstruction was issued asynchronously. In such cases, VTAM can pass back information when the request is accepted and when it is completed. (Refer to z/OS Communications Server: SNA Programming for complete details. Table 46 on page 264 and Table 47 on page 266 illustrate feedback completion information available at the two stages for the APPCCMD macroinstruction.) The completion codes are not set in the RPL or the RPL extension until the APPCCMD macroinstruction request completes. If the macroinstruction was issued synchronously, only completion information is returned.

For an asynchronous request that was accepted initially (register 15 equals 0 at the next instruction after the macroinstruction that was issued), the completion codes are available after the completion exit is scheduled or the completion ECB is posted. (You can issue the APPCCMD CONTROL=CHECK macroinstruction to make the RPL and RPL extension available for reuse by another macroinstruction request.) For a synchronous request or for an asynchronous request that was not accepted initially, the completion codes are available when the application program gets control at the next instruction following the macroinstruction request.

Register 15 contains the RTNCD value and register 0 contains the FDB2 value following the completion of the macroinstruction for synchronous APPCCMD requests or completion of the APPCCMD CONTROL=CHECK macroinstruction for asynchronous APPCCMD requests. However, a SYNAD or LERAD user exit routine can change register 15.

SYNAD and LERAD exit routines are not given control for APPCCMD macroinstruction errors except for a few errors indicated by a general return code higher than 0. For more information on these errors, see “SYNAD” on page 244 and “LERAD” on page 245.

The starting points for checking feedback from an APPCCMD macroinstruction are registers 15 and 0. VTAM cannot always set the feedback fields in the RPL and RPL extension for certain errors, but the registers always contain valid feedback codes.

Register 15 contains the general return code from VTAM. Only a few values are defined for the general return code, and it gives a quick indication of whether the macroinstruction was successful. Register 0 contains either a conditional completion return code or recovery action return code. In the case of errors, it provides more detail on the cause of the error.

If both register 15 and register 0 are 0, the APPCCMD has completed (or been accepted) without error. If register 15 is 0 but register 0 contains X'0000000B', the request completed conditionally. This may not indicate an error. The application program must check the RCPRI (RPLRCPR) and RCSEC (RPLRCSC) fields in the RPL extension to determine whether an error occurred.

An RCPRI value of 0 indicates that no error occurred. The RCSEC field contains a nonzero value that contains information about the processing of the macroinstruction. For example, a successful CNOS request can complete without error, but be negotiated by the partner LU. In such cases, RCSEC is set to X'0002' to indicate that negotiation took place.

A nonzero value for RCPRI indicates abnormal completion of an APPCCMD macroinstruction. The RCPRI and RCSEC fields contain the information needed to determine the error.

The preceding guidelines all assumed a 0 general return code in register 15. Because VTAM support of LU 6.2 is designed to intercept many errors and present them to the application program in an orderly fashion, this will be the case for most APPCCMD errors. The normal situation would be a register 15, register 0 combination of X'00000000', X'0000000B' (general return code 0, conditional completion return code X'B'), with the RCPRI and RCSEC fields in the RPL extension containing return codes that define the error.

Errors that cause a general return code higher than 0 (usually X'4') indicate a logic error in the application program. They are most likely to occur during program development. These errors include:

- Incorrectly setting an RPL or RPL address or using an asynchronous macroinstruction without an exit
- Incorrectly setting an RPL extension or RPL extension address
- Attempting to use an RPL or RPL extension marked as active
- Using a non-APPCCMD CHECK macroinstruction to complete an APPCCMD macroinstruction or using an APPCCMD CONTROL=CHECK macroinstruction to complete a non-APPCCMD VTAM macroinstruction

For more information on these errors, see “LERAD” on page 245.

Note: Error conditions usually are reported to the application program as completion information for a macroinstruction after the macroinstruction completes. However, the application program might not be made aware of some error conditions until after the next command processes. (The exception to this situation would be if the error condition caused the session to end.) An operator display command indicates that the conversation is present until the application program is notified that the conversation failed, even though the conversation might have been deallocated.

Table 46 on page 264 and Table 47 on page 266 indicate when RPL and RPL extension fields are set by VTAM. (Refer to z/OS Communications Server: SNA Programming for complete details.)

Table 46. Completion conditions available at acceptance stage of asynchronous requests

Completion Condition	Registers at Entry to LERAD/SYNAD	Registers at Exit from LERAD/SYNAD	Registers at NSI (LERAD/SYNAD not Available or not Applicable)	RPL Feedback Fields Set
Request accepted General return code=0 Recovery action return code=0	N/A	N/A	R15=0 R0=0	RTNCD=0 FDB2=0 RCPRI not set RCSEC not set
Conditional Completion General return code=0 Conditional Completion code=X'0B'	N/A	N/A	N/A	RTNCD=0 FDB2=X'0B' RCPRI set RCSEC set

Table 46. Completion conditions available at acceptance stage of asynchronous requests (continued)

Completion Condition	Registers at Entry to LERAD/SYNAD	Registers at Exit from LERAD/SYNAD	Registers at NSI (LERAD/SYNAD not Available or not Applicable)	RPL Feedback Fields Set
Request not accepted due to environment error General return code=4 Recovery action return code=X'10'	SYNAD entered R15=SYNAD exit address R0=X'10'	R15 and R0 set by SYNAD	R15=4 R0=X'10'	RTNCD=X'10' FDB2=specific error return code RCPRI not set RCSEC not set
General logic error General return code=4 Recovery action return code=X'14'	LERAD entered R15=LERAD exit address R0=X'14'	R15 and R0 set by LERAD	R15=4 R0=X'14'	RTNCD=X'14' FDB2=specific error return code RCPRI not set RCSEC not set
Logic error due to invalid RPL General return code=4 Recovery action return code=X'18'	LERAD entered R15=LERAD exit address R0=X'18'	R15 and R0 set by LERAD	R15=4 R0=X'18'	RTNCD not set FDB2 not set RCPRI not set RCSEC not set
Logic error due to invalid RPL extension General return code=4 Recovery action return code=X'1C'	LERAD entered R15=LERAD exit address R0=X'1C'	R15 and R0 set by LERAD	R15=4 R0=X'1C'	RTNCD not set FDB2 not set RCPRI not set RCSEC not set
Logic error due to RPL in wrong state General return code=4 Recovery action return code=X'20'	LERAD entered R15=LERAD exit address R0=X'20'	R15 and R0 set by LERAD	R15=4 R0=X'20'	RTNCD not set FDB2 not set RCPRI not set RCSEC not set
Request not accepted because ACB is not open General return code=32 No recovery action return code	LERAD/SYNAD not entered	LERAD/SYNAD not entered	Reg 15=32 Reg 0=Request code (see description of RPL's REQ field)	RPL not set

Table 47. Completion conditions for synchronous requests or CHECK of asynchronous requests

Completion Condition	Registers at Entry to LERAD/SYNAD	Registers at Exit from LERAD/SYNAD	Registers at NSI (LERAD/SYNAD not Available or not Applicable)	RPL Feedback Fields Set
Normal completion (no special conditions) General return code=0 Recovery action return code=0	N/A	N/A	R15=0 R0=0	RTNCD=0 FDB2=0 RCPRI=0 RCSEC=0
Normal completion (with special conditions) General return code=0 Recovery action return code=X'B'	N/A	N/A	R15=0 R0=X'B'	RTNCD=0 FDB2=X'B' RCPRI=0 RCSEC=nonzero return code
Abnormal completion due to APPCCMD error General return code = 0 Recovery action return code = X'B'	N/A	N/A	R15=0 R0=X'B'	RTNCD=0 FDB2=X'B' RCPRI=nonzero return code RCSEC=nonzero return code
Abnormal completion due to environment error General return code=4 Recovery action return code=X'10'	SYNAD entered R15=SYNAD exit address R0=X'10'	R15 and R0 set by SYNAD	R15=4 R0=X'10'	RTNCD=X'10' FDB2=specific error return code RCPRI not set RCSEC not set
General logic error General return code=4 Recovery action return code=X'14'	LERAD entered R15=LERAD exit address R0=X'14'	R15 and R0 set by LERAD	R15=4 R0=X'14'	RTNCD=X'14' FDB2=specific error return code RCPRI not set RCSEC not set
Logic error due to invalid RPL General return code=4 Recovery action return code=X'18'	LERAD entered R15=LERAD exit address R0=X'18'	R15 and R0 set by LERAD	R15=4 R0=X'18'	RTNCD not set FDB2 not set RCPRI not set RCSEC not set

Table 47. Completion conditions for synchronous requests or CHECK of asynchronous requests (continued)

Completion Condition	Registers at Entry to LERAD/SYNAD	Registers at Exit from LERAD/SYNAD	Registers at NSI (LERAD/SYNAD not Available or not Applicable)	RPL Feedback Fields Set
Logic error due to invalid RPL extension General return code=4 Recovery action return code=X'1C'	LERAD entered R15=LERAD exit address R0=X'1C'	R15 and R0 set by LERAD	R15=4 R0=X'1C'	RTNCD not set FDB2 not set RCPRI not set RCSEC not set
Logic error due to RPL in wrong state General return code=4 Recovery action return code=X'20'	LERAD entered R15=LERAD exit address R0=X'20'	R15 and R0 set by LERAD	R15=4 R0=X'20'	RTNCD not set FDB2 not set RCPRI not set RCSEC not set
Request not accepted because ACB is not open General return code=32 No recovery action return code	LERAD/SYNAD not entered	LERAD/SYNAD not entered	Reg 15=32 Reg 0=Request code (see description of RPL's REQ field)	RPL not set

Using exit routines to handle errors

The application program can supply VTAM with special-purpose exit routines to handle error situations. These are the LERAD and SYNAD exit routines. In addition, the application program can supply VTAM with a TPEND exit routine that is called when VTAM is forced to close down.

The LERAD or SYNAD exit (usually the LERAD) is scheduled by VTAM when an APPCCMD has a general return code higher than 0. VTAM interrupts whatever part of the application program is executing when the error is discovered (including another exit routine) and schedules the exit. The exit is responsible for setting register 15 and register 0 before the application program again receives control.

The LERAD and SYNAD exits might be called after the application program has already processed an error. Much of the error feedback information in the RPL is available when the RPL is checked prior to being cleared and marked as inactive. For example, if an APPCCMD request specifies an RPL exit, error feedback information is available to the RPL exit. Even if the exit takes recovery action, the LERAD or SYNAD is still scheduled for error situations and is driven after the RPL exit completes and the APPCCMD is considered complete.

If no LERAD or SYNAD exit routine is provided, register 15 and register 0 are set to the general return code and recovery action return code, respectively. If LERAD or SYNAD is entered, register 0 contains the recovery action return code.

For more information on these exits, refer to z/OS Communications Server: SNA Programming and Chapter 12, “Using exit routines,” on page 239 in this book.

Evaluating RCPRI, RCSEC return codes

Most of the error feedback for APPCCMD macroinstructions is found in the RPL extension fields RCPRI and RCSEC. The principal cause of the error is described in RCPRI, with RCSEC further qualifying the description.

Many of the RCPRI,RCSEC combinations indicate whether the error is the result of a temporary condition or a more serious problem. Some combinations indicate whether the macroinstruction should be issued again. (To determine these combinations, refer to z/OS Communications Server: SNA Programmer's LU 6.2 Reference.) The return codes also indicate if data was lost when sending or receiving data.

Error return codes can report changes in the state of the conversation. For example, an error return code on an APPCCMD CONTROL=SEND macroinstruction can mean that side of the conversation has been placed in RECEIVE state. The application program can determine the conversation state by examining the RPL6CCST field at the completion of most APPCCMD macroinstructions. If the application program is maintaining conversation states, it should update the conversation state in such cases.

Each possible variation of the APPCCMD macroinstruction is listed in the z/OS Communications Server: SNA Programmer's LU 6.2 Reference with a complete list of possible return codes for that macroinstruction. You should use that list as a starting point for deciding which return codes your application program should process.

Error return codes might not be caused by the macroinstruction on which they are returned. Allocation errors, for example, are frequently reported on APPCCMD macroinstructions issued after APPCCMD CONTROL=ALLOC. An application program could issue APPCCMD CONTROL=ALLOC and several APPCCMD CONTROL=SEND macroinstructions before any data is actually sent through the network. When designing your application program, you should keep in mind that errors reported by the partner LU might not necessarily pertain to the macroinstruction with the nonzero return code. Errors such as parameter errors or state errors are detected by VTAM when the macroinstruction is issued and will pertain to the macroinstruction with the nonzero return code.

Response to errors

The APPCCMD macroinstruction gives the application program three major alternatives when it discovers an error. The application program can do one of the following actions:

- Report it to the conversation partner
- Deallocate the conversation
- Deallocate the conversation and end the session supporting the conversation

The last of these is meant to be used only when a protocol error is discovered, indicating serious problems with the exchange of data over the session.

These three alternatives correspond to the following macroinstructions:

- APPCCMD CONTROL=SEND, QUALIFY=ERROR
- The four abnormal deallocation varieties of APPCCMD CONTROL=DEALLOC|DEALLOCQ
- APPCCMD CONTROL=REJECT

APPCCMD CONTROL=DEALLOC|DEALLOCQ and APPCCMD CONTROL=SEND, QUALIFY=ERROR have additional keywords to further define the type of error to which the application program is responding. This additional level of detail helps create more understandable return codes to pass to the partner LU.

The APPCCMD CONTROL=SEND variations are:

- APPCCMD CONTROL=SEND, QUALIFY=ERROR,TYPE=PROGRAM
- APPCCMD CONTROL=SEND, QUALIFY=ERROR,TYPE=SERVICE
- APPCCMD CONTROL=SEND, QUALIFY=ERROR,TYPE=USER

The APPCCMD CONTROL=DEALLOC variations are:

- APPCCMD CONTROL=DEALLOC, QUALIFY=ABNDPROG

- APPCCMD CONTROL=DEALLOC, QUALIFY=ABNDSERV
- APPCCMD CONTROL=DEALLOC, QUALIFY=ABNDTIME
- APPCCMD CONTROL=DEALLOC, QUALIFY=ABNDUSER

The APPCCMD CONTROL=DEALLOCQ variations are:

- APPCCMD CONTROL=DEALLOCQ, QUALIFY=ABNDPROG
- APPCCMD CONTROL=DEALLOCQ, QUALIFY=ABNDSERV
- APPCCMD CONTROL=DEALLOCQ, QUALIFY=ABNDTIME
- APPCCMD CONTROL=DEALLOCQ, QUALIFY=ABNDUSER

Choice of response

Whether an application program reports an error or performs a more drastic action, such as deallocating a conversation or ending the session, depends on the severity of the error and the ability of the logic within the application program to recover from the error. A few rules govern which step the application program should take. However, the most drastic step (ending the conversation and terminating the session) should be used only if the error seems to be caused by a violation of the architecture's protocols.

For most errors, if the application program cannot recover from the error on the conversation, only the conversation should be ended. Sometimes APPCCMD CONTROL=DEALLOC|DEALLOCQ cannot be issued (possibly because another APPCCMD macroinstruction is outstanding), and in these cases the application program must use the APPCCMD CONTROL=REJECT, QUALIFY=CONV macroinstruction. (For more information on ending a conversation, see Chapter 8, “Deallocating a conversation,” on page 167.)

When an application program reports an error on a half-duplex conversation with APPCCMD CONTROL=SEND, QUALIFY=ERROR, it is placed in SEND state, regardless of the state of the conversation up to that point. It can send additional data regarding the error or do anything else that the application program can do in SEND state. VTAM creates an FMH-7 as a result of this macroinstruction and stores it in the SEND buffer. The application program can use a macroinstruction with the FLUSH capability to force VTAM to send the error notification to the partner. For full-duplex conversations, the FMH-7 is not buffered, but sent immediately to the conversation partner.

Error types

Three of the error types—PROGRAM, SERVICE, and USER— are common to both APPCCMD CONTROL=SEND, QUALIFY=ERROR and the abnormal deallocation macroinstructions. (These errors are indicated on deallocation macroinstructions by the QUALIFY values of ABNDPROG, ABNDSERV, and ABNDUSER.) The PROGRAM and SERVICE types are defined in LU 6.2 architecture. The USER type is provided to give you more flexibility in reporting the cause of errors.

The distinction between PROGRAM and SERVICE errors is based on transaction programs. PROGRAM errors are reported to a transaction program. SERVICE errors are reported to a part of the application program that is implementing a component of the LU 6.2. Because the application program can implement LU 6.2 options in addition to transaction programs, VTAM reports both types of errors to the application program. The application program must screen service errors from the processing threads in the application program that are analogous to transaction programs.

The application program might not need to report SERVICE errors unless it is implementing LU 6.2 options, such as mapped conversations. As an example, suppose an application program expects data (representing a date) from another application program in a format of mm/dd/yy, but instead receives the date in a dd/mm/yy format. If the conversation on which the data flowed is a basic conversation (the conversation level provided by VTAM), the error probably was caused by the processing thread in the partner LU using the conversation. (That processing thread is analogous to a transaction program). The application program used the conversation correctly, and VTAM transferred the data without error, but a misunderstanding occurred between the two LUs regarding the data to send and receive. To report this type of error, the application program could issue an APPCCMD SEND, QUALIFY=ERROR, TYPE=PROGRAM macroinstruction.

Suppose instead that the conversation type was a mapped conversation and that the application program discovers a problem with the format of the data it is receiving. Because the LU portion of the application determines how to format data for a mapped conversation, the error is probably in the mapping component of the partner LU. In such a case, the application program detecting the error would issue an APPCCMD CONTROL=SEND, QUALIFY=ERROR, TYPE=SERVICE macroinstruction, indicating that the error was caused by something within the LU itself.

For both the PROGRAM and SERVICE errors, the reporting application program specifies the type of error, and can optionally include error log data in the form of a formatted GDS variable.

Timer errors

The APPCCMD CONTROL=DEALLOC|DEALLOCQ, QUALIFY=ABNDTIME macroinstruction can be used to deallocate a conversation when the application program has not received information during the application-specified amount of time.

The primary use is for an LU to detect timing errors on its side of the conversation. For example, if an LU in a sending state does not obtain any data to send from a transaction program within a specified time frame, it could use this type of deallocation.

Several restrictions apply to the use of this type of deallocation to detect timing errors while in RECEIVE state or awaiting a confirmation reply. The abnormal deallocation APPCCMD CONTROL=DEALLOC|DEALLOCQ macroinstructions cannot be used while the application program is waiting for a confirmation reply. In such cases, the application program must use APPCCMD CONTROL=REJECT to terminate the conversation. An example of a situation in which the ABNDTIME macroinstruction could be used would be if an application program had been sending data to a partner LU and realized that the partner LU had not issued an APPCCMD CONTROL=SEND, QUALIFY=RQSEND macroinstruction within a specified time frame.

Sense codes

In some cases, the application program might need to include sense information on the FMH-7 header or UNBIND RU that VTAM creates as a result of the APPCCMD macroinstruction. This might be particularly true when the LU validates LU 6.2 protocols, as in the case of option sets that it is implementing or data in an FMH-5 header. VTAM provides the USER error type so that the application program can specify an appropriate sense code. When providing error data explicitly, use APPCCMD CONTROL=SEND, QUALIFY=ERROR, TYPE=USER or APPCCMD CONTROL=DEALLOC|DEALLOCQ, QUALIFY=ABNDUSER. In each of these cases, the sense data is coded on the SENSE parameter.

If the application program provides a USER error type, it must supply an appropriate sense code for the error. (For a list of these sense codes, see “VTAM sense codes” on page 273.) The receiver uses the content of the sense code to determine the response or actions of the sender.

Consider the results desired before specifying the sense code. Otherwise, improper processing of the macroinstruction might result.

Situations that the application program might be able to detect are indicated by the sense codes for:

- REQUEST REJECT (category byte X'08')
- REQUEST ERROR (category byte X'10')

The LU 6.2 application program probably should not use sense codes that indicate situations that it might not be able to detect. These situations are indicated by the sense codes for:

- STATE ERROR (category byte X'20')
- PATH ERROR (category byte X'80')
- RH USAGE ERROR (category byte X'40')

The sense codes that an application program is most likely to use are those that deal with:

- Security violations
- Insufficient resources
- Error logging

- Errors in FMH headers (because the application program validates the FMH-5)

However, your application program might have unique needs that make other sense codes feasible. Consult the LU 6.2 architecture manuals for a complete list of sense codes.

Data purging and truncating

When an application program reports an error, the data that is waiting to be received or sent over the network can be lost. When a PROGRAM or SERVICE error is reported with APPCCMD CONTROL=SEND, QUALIFY=ERROR, a return code indicates whether any data has been lost. The abnormal APPCCMD CONTROL=DEALLOC|DEALLOCQ macroinstructions and APPCCMD CONTROL=SEND, QUALIFY=ERROR,TYPE=USER macroinstructions do not create a return code that indicates whether data purging or truncating has taken place, although it might have happened.

Note: Purging does not occur when an APPCCMD CONTROL=SEND, QUALIFY=ERROR macroinstruction is issued on a full-duplex conversation.

Purging error codes

If the application program is receiving data on a half-duplex conversation when it issues APPCCMD CONTROL=SEND, QUALIFY=ERROR to report a PROGRAM or SERVICE error, return codes of PROGRAM_ERROR_PURGING (X'0034' in RCPRI) or SERVICE_ERROR_PURGING (X'0040' in RCPRI) are reported to the partner. The PURGING return codes are reported to the former sender before any other error data is sent.

Whenever purging is reported, the sending LU should assume that all data sent because the last positive confirmation response has been lost. Receipt of a positive confirmation response indicates that data has been sent by the local transaction program and received by the partner without error.

Application programs also use APPCCMD CONTROL=SEND, QUALIFY=ERROR to respond negatively to a confirmation request on a half-duplex conversation. In such cases, the partner LU receives one of the return codes indicating purging.

When an application program issues APPCCMD CONTROL=SEND, QUALIFY=ERROR on a half-duplex conversation in RECEIVE state, information other than conversation data might be lost. Table 48 on page 271 shows the return codes VTAM may fail to report to the application program in this circumstance.

Table 48. Unreported return codes

Error Message	RCPRI Code
ALLOCATION_ERROR	X'0004'
DEALLOCATE_ABEND_PROGRAM	X'0014'
DEALLOCATE_ABEND_SERVICE	X'0018'
DEALLOCATE_ABEND_TIMER	X'001C'
PROGRAM_ERROR_TRUNCATING	X'0038'
PROGRAM_ERROR_NO_TRUNC	X'0030'
PROGRAM_ERROR_PURGING	X'0034'
SERVICE_ERROR_TRUNCATING	X'0044'
SERVICE_ERROR_NO_TRUNC	X'003C'
SERVICE_ERROR_PURGING	X'0040'

When an allocation error or abnormal deallocation indication is lost, VTAM passes back a return code of DEALLOCATE_NORMAL (X'0080' in RCPRI) on the APPCCMD CONTROL=SEND, QUALIFY=ERROR macroinstruction. When program and service error indications are lost, VTAM passes back a return code of OK (X'0000' in RCPRI). The application program receives no indication that the partner reported an error.

When an application program issues an abnormal termination macroinstruction, either APPCCMD CONTROL=DEALLOC|DEALLOCQ or APPCCMD CONTROL=REJECT, the partner LU receives no indication of whether data has been purged. Because the application program is responding to a rather severe error, however, the LU can assume that purging or truncating has occurred.

Truncating error codes

Application programs can issue APPCCMD CONTROL=SEND, QUALIFY=ERROR when either sending or receiving data on half-duplex conversations. Application programs can issue this macroinstruction only when allowed to send data on full-duplex conversations. If an application program is sending data, and a complete logical record has not been sent, a return code of PROGRAM_ERROR_TRUNCATING (X'0038' in RCPRI) or SERVICE_ERROR_TRUNCATING (X'0044' in RCPRI) is reported to the partner LU. The partner LU will not receive the remainder of the record being sent. Reporting this error is the orderly way for an application program to stop sending a logical record.

If a complete logical record has been sent, return codes of PROGRAM_ERROR_NO_TRUNC (X'0030' in RCPRI) or SERVICE_ERROR_NO_TRUNC (X'003C'in RCPRI) are reported to the partner.

Error log variables

When an application program reports an error, it can specify a data area containing error information to send to the partner LU. This data area should contain a formatted error log general data stream (GDS) variable as defined by SNA architecture. VTAM does no error checking, however, to ensure that it does. VTAM treats the error log variable as a normal logical record.

The ability to specify data when reporting an error enables application programs to implement the LU 6.2 option of logging error data in a system log. Because the application program records the data, it receives the error log data. The LU 6.2 architecture does not intend for this data to be passed to a transaction program.

The LOGRCV indicator in the RPL extension is set on when an error is reported to indicate that error log data follows. The application program should check this field when error return codes are received to determine whether such data will follow. Application programs receiving error log data must issue APPCCMD CONTROL=RECEIVE, QUALIFY=SPEC|ISPEC to receive the data. The APPCCMD CONTROL=RECEIVE, QUALIFY=ANY|IANY *cannot* be used to receive log data.

When an application program is informed that error log data is to follow, it must issue the RECEIVE macroinstruction even if the conversation is being terminated with a deallocation macroinstruction. Errors in receiving this data should be reported as protocol errors and the session terminated with APPCCMD CONTROL=REJECT.

The description of the fields is:

Bytes
Meaning

0–1
Length, in binary, of error log variable, including this length field.

2–3
General data stream ID: X'12E1'.

4–5
Length, in binary, of product set identifier subvector, including this length field.

6–m
Product set ID subvector. It need not be present. If it is not present, its length field is set to X'02'.

m+1
Length, in binary, of message text, including this length field.

m+3
Message text. The LUs that are involved define this.

VTAM sense codes

VTAM puts sense codes in the FMH-7 and UNBIND requests that it generates when it discovers an error or that it generates in response to an application program that reports an error.

Sense codes for FMH-7

The application program can set the sense data sent by VTAM in an FMH-7 through the SENSE field on the APPCCMD CONTROL=SEND, QUALIFY=ERROR, TYPE=USER, or APPCCMD CONTROL=DEALLOC|DEALLOCQ, QUALIFY=ABNDUSER macroinstruction. For the other types of error reporting, VTAM sets the sense data. For more information on FMH-7s, refer to *SNA Transaction Programmer's Reference Manual for LU Type 6.2*.

For a complete listing of all sense codes, including sense codes for FMH-7, refer to z/OS Communications Server: IP and SNA Codes. VTAM can set the following sense data for FMH-7:

Code
Meaning

X'080Fnnnn'
End user not authorized. The end user making the request does not have access to the requested resource.

Bytes 2 and 3 can contain sense-code-specific information.

X'080F6051'
Access security violation. VTAM or the application program detects a security protocol violation.

X'084Bnnnn'
Requested resources not available. Resources named in the request, and required to honor it, are not currently available. It is not known when the resources are to be available.

Bytes 2 and 3 contain the following sense-code-specific information.

X'084B6031'
Transaction program not available—retry allowed. The FMH-5 ATTACH command specifies a transaction program that the receiver is unable to start. Either the program is not authorized to run or the resources to run it are not available at this time. The condition is temporary. The sender determines when and if to try the operation again.

X'084Cnnnn'
Permanent insufficient resource. The receiver cannot act on the request because resources required to honor the request are permanently unavailable. The sender should not try again immediately because the situation is not transient.

Bytes 2 and 3 can contain sense-code-specific information.

X'084C0000'
Transaction program not available—no retry. The FMH-5 ATTACH command specifies a transaction program that the receiver is unable to start. The sender should not try again immediately.

X'0864nnnn'
Function terminated abnormally. The conversation was terminated abnormally. Other terminations can occur after repeated reexecution; the request sender must detect such a loop.

Bytes 2 and 3 can contain sense-code-specific information.

Note: Sense codes in the X'0864nnnn' range should not be used with APPCCMD CONTROL=SEND, QUALIFY=ERROR, TYPE=USER unless followed by an APPCCMD CONTROL=DEALLOC, QUALIFY=FLUSH macroinstruction. These codes indicate to the receiver that deallocation is occurring.

X'08640000'
Premature conversation termination. The conversation terminates abnormally; for example, the transaction program might issue a DEALLOCATE_ABEND verb, or the application program might terminate (normally or abnormally) without explicitly terminating the conversation.

X'08640001'
System logic error, no retry. A system logic error is detected. No retry of the conversation should be attempted.

X'08640002'
Excessive elapsed time, no retry. Excessive time elapses while the conversation waits for a required action or event. For example, a transaction program fails to issue a conversation-related protocol boundary verb. No retry of the conversation should be attempted.

X'08640003'
Allocation error. This code is used in a negative response to an allocation request on a full-duplex conversation. The sense code indicating the specific allocation error will be carried on the subsequent FMH-7.

X'0889nnnn'
Transaction program error. The transaction program has detected an error.

Bytes 2 and 3 can contain sense-code-specific information.

X'08890000'
Program error, no data truncation. The transaction program that is sending data detects an error but does not truncate a logical record.

Program error, purging. The transaction program that is receiving data detects an error. All remaining information, if any, that the receiving program has not received and that the sending program sent before being notified of the error is discarded. This is applicable only to half-duplex conversations.

Note: This code is not valid for an APPCCMD CONTROL=DEALLOC|DEALLOCQ, QUALIFY=ABNDUSER issued for a full-duplex conversation.

X'08890001'
Program error, data truncation. The transaction program that is sending data detects an error and truncates the logical record that it is sending.

Note: This code is not valid for an APPCCMD CONTROL=DEALLOC|DEALLOCQ, QUALIFY=ABNDUSER issued for a full-duplex conversation.

X'08890100'
Service transaction program error, no data truncation. The service transaction program that is sending data detects an error and does not truncate a logical record.

Service transaction program error, purging. The service transaction program that is receiving data detects an error. All remaining information, if any, that the receiving service transaction program has not yet received and that the sending service transaction program has sent before being notified of the error is discarded. This is applicable only to half-duplex conversations.

Note: This code is not valid for an APPCCMD CONTROL=DEALLOC|DEALLOCQ, QUALIFY=ABNDUSER issued for a full-duplex conversation.

X'08890101'
Service transaction program error, data truncation. The service transaction program that is sending data detects an error and truncates the logical record that it is sending.

Note: This code is not valid for an APPCCMD CONTROL=DEALLOC|DEALLOCQ, QUALIFY=ABNDUSER issued for a full-duplex conversation.

X'1008nnnn'
Not valid FM header. The receiver cannot understand or translate the FM header, or the receiver expects an FM header and one is not present.

Bytes 2 and 3 can contain sense-code-specific information.

X'1008600B'
Unrecognized FM header command code. The partner LU receives an FM header command code that it does not recognize.

X'10086021'
Transaction program name not recognized. The FMH-5 ATTACH command specifies a transaction program name that the receiver does not recognize.

X'10086031'
PIP not allowed. The FMH-5 ATTACH command specifies that program initialization parameter (PIP) data is present but the receiver does not support PIP data for the specified transaction program.

X'10086032'
PIP not specified correctly. The FMH-5 ATTACH command specifies a transaction program name that requires program initialization parameter (PIP) data and either the FMH-5 specifies that PIP data is not present or the number of PIP subfields present does not agree with the number required for the program.

X'10086034'
Conversation type mismatch. The FMH-5 ATTACH command specifies a conversation type that the receiver does not support for the specified transaction program.

X'10086041'
Synchronization level not supported. The FMH-5 ATTACH command specifies a synchronization level that the receiver does not support for the specified transaction program.

X'10086042'
Reconnection not supported. The FMH-5 ATTACH command specifies reconnection support, but the receiver does not support reconnection for the specified transaction program.

X'10086043'
Unable to reconnect transaction program, no retry. The FMH-5 RECONNECT command specifies the conversation correlator of a transaction program to which the receiver cannot reconnect.

X'10086044'
Unable to reconnect transaction program, retry allowed. The FMH-5 RECONNECT command specifies the conversation correlator of a transaction program to which the receiver cannot reconnect. The condition is temporary.

Sense codes for UNBIND

The following sense data can be sent by VTAM in the UNBIND RU:

Code
Meaning

X'080Fnnnn'
End user not authorized. The requesting end user does not have access to the requested resource.

Bytes 2 and 3 can contain sense-code-specific information.

X'080F0002'
Session-Level LU-LU Verification Protocol Mismatch. An LU that supports only the enhanced LU-LU verification protocol received a BIND or BIND response that specified the basic LU-LU verification protocol.

X'080F6051'
Access security violation. VTAM or the application program detects a security protocol violation.

X'0812nnnn'
Insufficient resource. The receiver cannot act on the request because of a temporary lack of resources.

Bytes 2 and 3 can contain sense-code-specific information.

X'08120000'
No specific code applies.

X'0812001E'
A session has failed because depletion of pooled buffer storage has exceeded a critical threshold resulting from that session's monopolizing usage.

X'08130000'
Bracket bid reject, no RTR forthcoming. The BID (or BB) is received while the contention winner is in the in-bracket state or the between-brackets state, and the contention winner denies permission. No RTR is to be sent.

X'08140000'
Bracket bid reject, RTR forthcoming. The BID (or BB) is received while the contention winner is in the in-bracket state or the between-brackets state, and the contention winner denies permission. RTR is sent.

X'08190000'
RTR not required. The receiver of READY TO RECEIVE has nothing to send.

X'081Cnnnn'
Request not executable. The requested function cannot be executed because of a permanent error condition in the receiver.

Bytes 2 and 3 can contain sense-code-specific information.

X'081C0000'
No specific code applies.

X'08460000'
ERP message forthcoming. The received request is rejected for a reason that is to be specified in a forthcoming request.

X'08880010'
Name conflict. The partner LU returned a name in the user data field of its RSP(BIND) that differs from the name it returned in the user data field of its RSP(BIND) for a previous BIND. Either the partner changed its name or name changes in the network have caused delivery of the latest BIND to a different partner.

X'08880011'
Name conflict. The partner receiving a BIND carrying one specific target SLU name returned a name in the user data field of its RSP(BIND) that is the same as it returned in response to a previous BIND carrying a different target SLU name.

X'08880012'
Name conflict. A session initiation request is received from the partner LU containing a LUNAME found in an internal table, but with a different network qualifier.

X'1001nnnn'
RU data error. Data in the request RU is not acceptable to the receiving component. For example:

- A character code is not in the set that is supported.
- A formatted data field is not acceptable to presentation services.
- A value specified in the length field (LL) of a structured field is not valid.
- A required name in the request is omitted.

Bytes 2 and 3 can contain sense-code-specific information.

X'10010000'
No specific code applies.

X'10010003'
An isolated pacing message is received that is not valid.

X'10020000'
RU length error. The request RU is too long or too short.

X'10030000'
Function not supported. The function requested is not supported. The function might have been specified by a formatted request code, a field in an RU, or a control character.

X'10030004'
Function not supported. A SIGNAL RU (indicating REQUEST_TO_SEND) was received on a full-duplex conversation, or an EXPD RU was received on a session that does not support full-duplex protocols. The session is unbound.

X'10050000'
Parameter error. A parameter that modifies a control function is not valid or is outside the range that the receiver allows.

An EXPD RU was received with a format or length that was not valid. The session is unbound.

X'10070000'
Category not supported:

- DFC, SC, NC, or FMD request is received by a half-session not supporting any requests in that category.
- An NS request byte 0 is not set to a defined value.
- Byte 1 is not set to an NS category that the receiver supports.

X'1008nnnn'
Not valid FM header. The receiver cannot understand or translate the FM header, or the receiver expects an FM header and one is not present.

Bytes 2 and 3 can contain sense-code-specific information.

X'1008200E'
Not valid concatenation indicator. The concatenation indicator is on but concatenation is not allowed.

X'1008201D'
FM header and associated data mismatch. The FM header indicates that:

- Associated data does or does not follow (such as FMH-7 followed by log data or FMH-5 followed by program initialization parameters), but this indication is in error.
- A previously received RU implies that an FM header follows, but none is received.

X'10084001'
Not valid FM header type. The type of the FM header is other than 5, 7, or 12.

X'10086000'
FM header length not correct. The value in the FM header length field differs from the sum of the lengths of the subfields of the FM header.

X'10086005'
Access security information length field not correct. The value in the access security information length field differs from the sum of the lengths of the access security information subfields.

X'10086009'
Not valid parameter length. The field that specifies the length of fixed-length parameters has a not valid setting.

X'10086011'
Not valid logical unit of work:

- The LUW length field (in a compare-states GDS variable or an FMH-5) is incorrect.
- The LUW is not valid.
- An LUWID is not present but is required by the setting of the synchronization level field.

X'10086040'
Not valid ATTACH parameter. A parameter in the FMH-5 ATTACH command conflicts with the statement of LU capability that was previously in the BIND negotiation.

X'20010000'
Sequence number. The sequence number received on the normal-flow request is not one greater than the last sequence number.

X'20020000'
Chaining. An error exists in the sequence of the chain indicator settings (BCI, ECI), such as first, middle, first.

X'20030000'
Bracket. An error results because the sender fails to enforce the bracket rules for the session. (This error does not apply to contention or race conditions.)

X'20040000'
Direction error. An error that results because a normal-flow request is received, and the half-duplex flip-flop state is not received.

Data was received from a partner LU on a full-duplex conversation that is in SEND_ONLY state. The session is unbound.

X'20080000'
No begin bracket. An FMD request that specifies BBI=BB is received after the receiver has received a BRACKET INITIATION STOPPED request.

X'200A0000'
Immediate request mode error. A request violates the immediate request mode protocol.

X'200B0000'
Queued response error. A request violates a queued response protocol, such as QRI=¬QR, when an outstanding request has QRI=QR.

X'200E0000'
Response correlation error. A response is received that does not correlate to a previous request.

A response for an EXPD RU was received, but an expedited request is not outstanding. This session is unbound.

X'200F0000'
Response protocol error. A violation occurs in the response protocol, such as the generation of +RSP to an RQE chain.

An EXPD RU was received, but no response has been sent to a previously received EXPD request. The session is unbound.

X'20110000'
Pacing error. A half-session receives a normal-flow request after the pacing count is reduced to 0 and before a pacing response is sent.

X'20120000'
Not valid sense code received. A negative response is received that contains an SNA-defined sense code that cannot be used for the request that is sent.

X'2013nnnn'
Decompression protocol error: A request containing compressed data was received in error.

Bytes 2 and 3 can contain sense-code-specific information.

X'40030000'
BB not allowed. The begin bracket indicator (BBI) is specified incorrectly, such as BBI=BB with BCI=¬BC.

X'40040000'
CEB or EB not allowed. The conditional end bracket indicator (CEBI) or end bracket indicator (EBI) is specified incorrectly.

X'40070000'
Definite response not allowed. A definite response is requested and is not permitted.

A Confirm request was received on a full-duplex conversation. The session is unbound.

X'40080000'
Pacing not supported. The pacing indicator is set on a request, but the receiving half-session or boundary function half-session does not support pacing for this session.

X'40090000'
CD not allowed. The change direction indicator (CDI) is specified incorrectly, such as CDI=CD with EBI=EB.

X'400B0000'
Chaining not supported. The chaining indicators (BCI and ECI) are specified incorrectly, such as chaining bits indicated other than (BC,EC), but multiple-request chains are not supported for the session or for the category specified in the request header.

A full-duplex conversation received a request carrying a CD indicator with ECI specified.

X'400C0000'
Brackets not supported. The bracket indicators (BBI, CEBI, and EBI) are specified incorrectly. For example, a bracket indicator is set (BBI=BB, CEBI=CEB, or EBI=EB), but brackets are not used for the session.

X'400F0000'
Incorrect use of format indicator. The format indicator (FI) is specified incorrectly. For example, FI is set with BCI=¬BC, or FI is not set on a DFC request.

X'40100000'
Alternate code not supported. The code selection indicator (CSI) is set but is not supported for the session.

X'40110000'
Incorrect specification of RU category. The RU category indicator is specified incorrectly. For example, an expedited-flow request or response is specified with RU category indicator=FMD.

X'40120000'
Incorrect specification of request code. The request code on a response does not match the request code on its corresponding request.

A received expedited flow response was not a SIGNAL or EXPD RU. The session is unbound.

X'40130000'
Incorrect specification of SDI, RTI. The sense data included indicator (SDI) and the response type indicator (RTI) are not specified properly on a response. The proper value pairs are (SDI=SD, RTI=negative) and (SDI=¬SD, RTI=positive).

X'40140000'
Incorrect use of DR1I, DR2I, ERI. The definite response 1 indicator (DR1I), definite response 2 indicator (DR2I), and exception response indicator (ERI) are specified incorrectly. For example, a SIGNAL request is not specified with DR1I=DR1, DR2I=¬DR2, and ERI=¬ER.

X'40150000'
Incorrect use of QRI. The queued response indicator (QRI) is specified incorrectly. For example, QRI=QR is specified on an expedited-flow request.

X'40160000'
Incorrect use of EDI. The enciphered data indicator (EDI) is specified incorrectly. For example, EDI=ED is specified on a DFC request.

X'40170000'
Incorrect use of PDI. The padded data indicator (PDI) is specified incorrectly. For example, PDI=PD is specified on a DFC request.

X'40180000'
Incorrect setting of QRI with loser's BB. The winner of a half-session receives a BB chain that requests use of a session (by means of LUSTAT(X'0006')), but the QRI is specified incorrectly; that is, QRI=¬QR.

X'40190000'
Incorrect indicators with last-in-chain request. A last-in-chain request specifies RH settings that are incompatible, such as RQE*, CEBI=¬CEB, and CDI=¬CD.

X'40210000'
QRI setting in response different from that in request. The QRI setting in the response differs from the QRI setting in the corresponding request.

X'8005nnnn'
No session. No half-session is active in the receiving end node for the indicated origination-destination pair, or no boundary function session connector is active for the origination-destination pair in a node providing the boundary function. A session activation request is needed.

Bytes 2 and 3 can contain sense-code-specific information.

X'80050000'
No specific code applies.

Appendix A. Conversation states

As part of its implementation of the LU 6.2 architecture, VTAM maintains a finite state machine for each conversation. This mechanism enables VTAM to regulate when an application may issue certain macroinstructions during a conversation. This appendix lists the possible states for a conversation, and includes a chart showing the relation of the states to one another, and when transitions from one state to the next occur.

States of conversations

The state of the conversation determines the type of APPCCMD, as denoted by the CONTROL and QUALIFY fields, that an application may issue. As the application issues APPCCMD macroinstructions, the state of the conversation changes. The change is a result of the function of the APPCCMD macroinstruction, a result of a verb issued by the remote transaction program, or a result of network errors.

The conversation state is defined in terms of the local application's view of the conversation. The states of other conversations allocated to the program can be different. For example, one conversation can be in RECEIVE state and another in SEND state, concurrently.

The conversation can be either a half-duplex conversation or a full-duplex conversation. Each of these two conversation types has its own set of possible states, with differing rules to allow transition from one state to another.

Descriptions of the possible conversation states for half-duplex conversations are located in “Half-duplex conversation states” on page 281. Possible conversation states for full-duplex conversations are described in “Full-duplex conversation states” on page 283.

Half-duplex conversation states

The following half-duplex conversation states are defined:

- **RESET** is the initial state in which the application program can allocate a conversation. There are no state transitions to the RESET state.
- **SEND** is the state in which the application program can send data, request confirmation, or request sync point.
- **RECEIVE** is the state in which the application program can receive information from the remote transaction program.
- **CONFIRM** is the state in which the application program can reply to a confirmation request and includes the actual states of RECEIVE_CONFIRM, RECEIVE_CONFIRM_SEND, and RECEIVE_CONFIRM_DEALLOCATE.
- **PEND_END_CONV_LOG** is the state in which the application program can receive error log data that immediately precedes the end of the conversation. (Error log data can also be received by the application program when the conversation is in RECEIVE state, but in that case the log data does not immediately precede the end of the conversation.) The PEND_END_CONV_LOG state is entered only when a conversation has ended and there is error log data to be received by the application.
- **PEND_RCV_LOG** is the state in which the application program can receive error log data that does not immediately precede the end of the conversation.
- **PEND_SEND** is the state in which the conversation is placed when data and a change-direction indicator are both returned on an APPCCMD CONTROL=RECEIVE macroinstruction. This state is transient, but you can display it with MODIFY DISPLAY. Look at the CD indicator to determine whether a response was sent. If the conversation is in PEND_SEND state, only an FMH-7 is sent.

- **PENDING_ALLOCATE** is the state in which the conversation is placed when the application issues the APPCCMD CONTROL=PREALLOC macroinstruction. The conversation remains in PENDING_ALLOCATE until the application issues the APPCCMD CONTROL=SENDFMH5 macroinstruction.
- **END_CONV** is the state in which the conversation is placed when the conversation is being deallocated. This state is transient. Although you might be able to see it by using MODIFY DISPLAY, once it completes the conversation ID is returned to the conversation pool for reuse.
- **PEND_DEALL** is the state in which the conversation is placed when the application issues the APPCCMD CONTROL=DEALLOC, QUALIFY=CONFIRM|DATACON macroinstruction. If the partner LU sends a positive reply, the conversation is placed in END_CONV state. If the partner LU sends a negative reply, the conversation is placed in RECEIVE state.

Note: SYNC_POINT, SYNC_POINT_SEND, SYNC_POINT_DEALLOCATE, DEFER_RECEIVE, DEFER_DEALLOCATE, DEALLOCATE, and BACKOUT_REQUIRED are not maintained by VTAM LU 6.2.

The state of the conversation determines the APPCCMD that the application is allowed to issue. Table 49 on page 282 correlates the APPCCMD and its CONTROL and QUALIFY settings to the half-duplex conversation states. For each APPCCMD and state, a "yes," "no," or "n/a" is indicated.

yes
The application program is allowed to issue the APPCCMD when the conversation is in that state.

no
The application program cannot issue the APPCCMD when the conversation is in that state. An APPCCMD issued for a conversation in a disallowed state completes with a return code indicating STATE_ERROR.

n/a
The state is not applicable either because it cannot exist at the time the APPCCMD is issued or because it is not relevant to the APPCCMD.

Table 49. Correlation of basic conversation macroinstructions to half-duplex conversation states

APPCCMD CONTROL= ,QUALIFY=	RESET	SEND	RECEIVE	CONFIRM	PEND END CONV LOG	PEND RCV LOG	PEND SEND	PEND ALLOC
DEALLOC, ABND*nnnn*	n/a	yes	yes	yes	yes	yes	yes	yes
DEALLOC, CONFIRM	n/a	yes	no	no	no	no	yes	no
DEALLOC, DATACON	n/a	yes	no	no	no	no	yes	no
DEALLOC, DATAFLU	n/a	yes	no	no	no	no	yes	no
DEALLOC, FLUSH	n/a	yes	no	no	no	no	yes	no
DEALLOCQ, ABND*nnnn*	n/a	yes	yes	yes	yes	yes	yes	yes
PREPRCV	n/a	yes	no	no	no	no	yes	no
RCVEXPD, ANY [4]	n/a	yes	yes	yes	yes	yes	yes	no
RCVEXPD, IANY [4]	n/a	yes	yes	yes	yes	yes	yes	no
RCVEXPD, ISPEC	n/a	yes	yes	yes	yes	yes	yes	no
RCVEXPD, SPEC	n/a	yes	yes	yes	yes	yes	yes	no
RECEIVE, ANY [5]	n/a	n/a	yes	n/a	n/a	n/a	n/a	no
RECEIVE, IANY [5]	n/a	n/a	yes	n/a	n/a	n/a	n/a	no
RECEIVE, ISPEC	n/a	no	yes	no	yes	yes	no	no
RECEIVE, SPEC	n/a	yes	yes	no	yes	yes	yes	no
REJECT, CONV	n/a	yes	yes	yes	yes	yes	yes	yes
RESETRCV	n/a	yes	yes	yes	yes	yes	yes	no
SEND, CONFIRM	n/a	yes	no	no	no	no	yes	no
SEND, CONFRMD	n/a	no	no	yes	no	no	no	no
SEND, DATA	n/a	yes	no	no	no	no	yes	no
SEND, DATACON	n/a	yes	no	no	no	no	yes	no

Table 49. Correlation of basic conversation macroinstructions to half-duplex conversation states (continued)

APPCCMD CONTROL= ,QUALIFY=	RESET	SEND	RECEIVE	CONFIRM	PEND END CONV LOG	PEND RCV LOG	PEND SEND	PEND ALLOC
SEND, DATAFLU	n/a	yes	no	no	no	no	yes	no
SEND, ERROR	n/a	yes	yes	yes	no	yes	yes	no
SEND, FLUSH	n/a	yes	no	no	no	no	yes	no
SEND, RQSEND	n/a	yes	yes	yes [6]	no	no	no	no
SENDEXPD, DATA [7]	n/a	yes	yes	yes	yes	yes	yes	no
SENDFMH5	n/a	n/a	no	no	no	no	no	yes

APPCCMD CONTROL=RCVFMH5, APPCCMD CONTROL=PREALLOC, and CONTROL=ALLOC cannot be issued against a specific conversation and therefore are not included in Table 49 on page 282. APPCCMD CONTROL=CHECK is also not included because it cannot cause a state error to occur. It causes the application program to synchronously wait until a previously-issued asynchronous APPCCMD completes. Because APPCCMD CONTROL=CHECK is issued against a prior APPCCMD, any state errors that occur would have been detected when the prior macroinstruction was issued. APPCCMD CONTROL=TESTSTAT is not included because it cannot cause a state error to occur. Its processing does not alter the conversation.

Full-duplex conversation states

The following full-duplex conversation states are defined:

- **RESET** is the initial state in which the application program can allocate a conversation and the ending state to which it returns when the conversation has ended.
- **FDX RESET** is the state in which the conversation is placed when the conversation is being deallocated. This is an intermediate state that is transparent to the application.
- **SEND/RECEIVE** is the state in which the application program can send and receive data simultaneously. This state is the normal state for a full-duplex conversation, entered initially when the allocation request is issued or when an Attach (FMH-5) is received from a remote transaction program.
- **SEND_ONLY** is the state in which the application program can send information to the remote transaction program but cannot issue any further receive requests. This state is entered when an application program in SEND/RECEIVE state receives a normal deallocation request from the remote transaction program.

---

[4] The APPCCMD CONTROL=RCVEXPD, QUALIFY=ANY|IANY cannot be issued against a specific conversation. This macroinstruction is shown in the table to illustrate the fact that when the application program issues the APPCCMD CONTROL=RCVEXPD, QUALIFY=ANY|IANY, a conversation in continue-any expedited data mode will apply to the APPCCMD in any active state.

[5] APPCCMD CONTROL=RECEIVE, QUALIFY=ANY|IANY cannot be issued against a specific conversation. This macroinstruction is shown in this table to illustrate the fact that when the application program issues the APPCCMD CONTROL=RECEIVE, QUALIFY=ANY, a conversation in continue-any normal data mode has to be in RECEIVE state in order to apply the APPCCMD.

[6] APPCCMD CONTROL=SEND, QUALIFY=RQSEND cannot be issued in CONFIRM state if the received confirmation request accompanied a request for termination of the conversation.

[7] The APPCCMD CONTROL=SENDEXPD, QUALIFY=DATA can be issued only against a half-duplex conversation if its underlying session supports full-duplex and expedited data protocols.

- **RECEIVE_ONLY** is the state in which the application program can receive information from the remote transaction program, but cannot issue any further send queue requests. This state is entered when an application in SEND/RECEIVE state issues a deallocation request.
- **PENDING_SEND/RECEIVE_LOG** is the state in which the application program can receive error log data that does not immediately precede the end of the conversation. Normal sending of data may continue, but a specific receive must be issued to receive the log data before normal receive operation can be resumed. This state is entered when an application in SEND/RECEIVE state receives an error response accompanied by error log data.
- **PENDING_RECEIVE-ONLY_LOG** is the state in which the applicator program can receive error log data that does not immediately precede the end of the conversation. A specific receive must be issued to receive the log data before normal receive operation can be resumed. This state is entered when an application in RECEIVE_ONLY state receives an error response accompanied by error log data.
- **PENDING_RESET_LOG** is the state in which the application program can receive error log data that immediately precedes the end of the conversation. This state is entered only when a conversation has ended and there is error log data to be received by the application.
- **PENDING_ALLOCATE** is the state in which the conversation is placed when the application issues the APPCCMD CONTROL=PREALLOC macroinstruction. The conversation remains in PENDING_ALLOCATE until the application issues the APPCCMD CONTROL=SENDFMH5 macroinstruction.

The state of the local conversation determines the APPCCMDs that the application is allowed to issue. Several APPCCMD CONTROL and QUALIFY value combinations that are allowed for half-duplex conversations are not valid if they are issued against a full-duplex conversation. Table 50 on page 284 correlates the APPCCMD and its CONTROL and QUALIFY values to the conversation states for full-duplex conversations. For each APPCCMD and conversation state, a "yes," "no," "inv," or "n/a" is indicated:

yes
The application program is allowed to issue the APPCCMD when the conversation is in that state.

no
The application program cannot issue the APPCCMD when the local conversation is in that state. An APPCCMD issued for a conversation in a disallowed state completes with a return code indicating STATE_ERROR.

inv
The application may not issue this APPCCMD on a full-duplex conversation.

n/a
The state is not applicable because it cannot exist at the time the APPCCMD is issued or because it is not relevant to the APPCCMD.

Table 50. Correlation of basic conversation macroinstructions to full-duplex conversation states

APPCCMD CONTROL= ,QUALIFY=	RESET	FDX RESET	SEND/ RECEIVE	SEND ONLY	RECEIVE ONLY	PEND SND/RCV LOG	PEND RCV ONLY LOG	PEND RESET LOG	PEND ALLOC
DEALLOC, ABND*nnnn*	n/a	n/a	yes	yes	yes	yes	yes	yes	yes
DEALLOC, CONFIRM	inv	inv	inv	inv	inv	inv	inv	inv	no
DEALLOC, DATACON	inv	inv	inv	inv	inv	inv	inv	inv	no
DEALLOC, DATAFLU	n/a	n/a	yes	yes	no	yes	no	no	no
DEALLOC, FLUSH	n/a	n/a	yes	yes	no	yes	no	no	no
DEALLOCQ, ABND*nnnn*	n/a	yes	yes	yes	yes	yes	yes	no	yes
PREPRCV	inv	inv	inv	inv	inv	inv	inv	inv	no
RCVEXPD, ANY [8]	n/a	n/a	yes	yes	yes	yes	yes	yes	no
RCVEXPD, IANY [8]	n/a	n/a	yes	yes	yes	yes	yes	yes	no
RCVEXPD, ISPEC	n/a	n/a	yes	yes	yes	yes	yes	yes	no
RCVEXPD, SPEC	n/a	n/a	yes	yes	yes	yes	yes	yes	no
RECEIVE, ANY [9]	n/a	n/a	yes	n/a	yes	n/a	n/a	n/a	no

Table 50. Correlation of basic conversation macroinstructions to full-duplex conversation states (continued)

APPCCMD CONTROL= ,QUALIFY=	RESET	FDX RESET	SEND/ RECEIVE	SEND ONLY	RECEIVE ONLY	PEND SND/RCV LOG	PEND RCV ONLY LOG	PEND RESET LOG	PEND ALLOC
RECEIVE, IANY [9]	n/a	n/a	yes	n/a	yes	n/a	n/a	n/a	no
RECEIVE, ISPEC	n/a	n/a	yes	no	yes	yes	yes	yes	no
RECEIVE, SPEC	n/a	n/a	yes	no	yes	yes	yes	yes	no
REJECT, CONV	n/a	n/a	yes	yes	yes	yes	yes	yes	yes
RESETRCV	n/a	n/a	yes	yes	yes	yes	yes	yes	no
SEND, CONFIRM	inv	inv	inv	inv	inv	inv	inv	inv	no
SEND, CONFRMD	inv	inv	inv	inv	inv	inv	inv	inv	no
SEND, DATA	n/a	n/a	yes	yes	no	yes	no	no	no
SEND, DATACON	inv	inv	inv	inv	inv	inv	inv	inv	no
SEND, DATAFLU	n/a	n/a	yes	yes	no	yes	no	no	no
SEND, ERROR	n/a	n/a	yes	yes	no	yes	no	no	no
SEND, FLUSH	n/a	n/a	yes	yes	no	yes	no	no	no
SEND, RQSEND	inv	inv	inv	inv	inv	inv	inv	inv	no
SENDEXPD, DATA [10]	n/a	n/a	yes	yes	yes	yes	yes	yes	no
SENDFMH5	n/a	n/a	no	no	no	no	no	no	yes

APPCCMD CONTROL=RCVFMH5, CONTROL=PREALLOC, and CONTROL=ALLOC cannot be issued against a specific conversation and therefore are not included in Table 49 on page 282. APPCCMD CONTROL=CHECK is also not included because it cannot cause a state error to occur. It causes the application program to synchronously wait until a previously-issued asynchronous APPCCMD completes. Because APPCCMD CONTROL=CHECK is issued against a prior APPCCMD, any state errors that occur would have been detected when the prior macroinstruction was issued. APPCCMD CONTROL=TESTSTAT is not included because it cannot cause a state error to occur. Its processing does not alter the conversation.

State matrix

A conversation enters a particular state when the application program issues an APPCCMD macroinstruction that causes a state transition, or when the application program receives an RCPRI, RCSEC combination or a WHATRCV value that indicates a state transition. The specific state transitions can be found under the heading "State Changes" for each APPCCMD macroinstruction description in the z/OS Communications Server: SNA Programmer's LU 6.2 Reference. State changes as a result of RCPRI, RCSEC return codes can also be found in that manual. (Also see Figure 35 on page 286 and Figure

[8] The APPCCMD CONTROL=RCVEXPD, QUALIFY=ANY|IANY cannot be issued against a specific conversation. This macroinstruction is shown in the table to illustrate the fact that when the application program issues the APPCCMD CONTROL=RCVEXPD, QUALIFY=ANY|IANY, a conversation in continue-any expedited data mode will apply to the APPCCMD in any active state.

[9] APPCCMD CONTROL=RECEIVE, QUALIFY=ANY|IANY cannot be issued against a specific conversation. This macroinstruction is shown in this table to illustrate the fact that when the application program issues the APPCCMD CONTROL=RECEIVE, QUALIFY=ANY, a conversation in continue-any normal data mode has to be in RECEIVE state in order to apply the APPCCMD.

[10] The APPCCMD CONTROL=SENDEXPD, QUALIFY=DATA can be issued only against a half-duplex conversation if its underlying session supports full-duplex and expedited data protocols.

36 on page 287 for half-duplex and full-duplex finite-state machines (FSM) that illustrate the possible conversation state transitions.)

FSM_CONVERSATION_HDX	RESET	SEND	RCV	RCVD CONFRM	RCVD CONFRM SEND	RCVD CONFRM DEALL	PEND DEALL	PEND END CONV LOG	END CONV	PEND SEND	PEND RCV LOG
Inputs	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
S, ALLOC	1	/	/	/	/	/	/	/	/	/	/
R, FMH-5	2	/	/	/	/	/	/	/	/	/	/
S, SEND, DATA	/	-	>	>	>	>	/	>	>	1	1
S, SEND, FLUSH|DATAFLU	/	-	>	>	>	>	/	>	>	1	>
S, SEND, CONFIRM|DATACON	/	-	>	>	>	>	/	>	>	1	>
S, PREPRCV	/	2	>	>	>	>	/	>	>	2	>
S, SEND, ERROR	/	-	1	1	1	1	/	>	>	1	1
S, SEND, RQSEND	/	-	-	-	-	-	>	>	>	1	>
S, SEND, CONFRMD	/	>	>	2	1	8	/	>	>	>	>
S, SENDEXPD, DATA	/	-	-	-	-	-	>	-	>	-	-
S, RCVEXPD	/	-	-	-	-	-	>	-	>	-	-
S, RECEIVE	/	2	-	>	>	>	/	-	>	2	>
S, RECEIVE, IMMED	/	>	-	>	>	>	/	-	>	>	>
R, SEND_INDICATOR	/	/	1	/	/	/	/	/	/	/	/
R, CONFIRM_INDICATOR	/	/	3	/	/	/	/	/	/	/	/
R, CONFIRM_SEND_INDICATOR	/	/	4	/	/	/	/	/	/	/	/
R, CONFIRM_DEAL_INDICATOR	/	/	5	/	/	/	/	/	/	/	/
R, PROGRAM_ERROR_RC	/	2	-	/	/	/	2	/	/	2	/
R, SERVICE_ERROR_RC	/	2	-	/	/	/	2	/	/	2	/
R, USER_ERROR_CODE_RC	/	2	-	/	/	/	2	/	/	2	/
R, SEND_ERROR_LOG_RC	/	10	10	/	/	/	10	/	/	/	/
R, DEALLOCATE_NORMAL_RC	/	8	8	/	/	/	8	/	/	8	8
R, DEALLOCATE_ABEND_RC	/	8	8	/	/	/	8	/	/	/	8
R, DEALL_ABEND_RC_LOG	/	7	7	/	/	/	7	/	/	7	7
R, RESOURCE_FAILURE_RC	/	8	8	8	8	8	8	8	/	8	8
R, RESOURCE_FAILURE_RC_LOG	/	7	7	7	7	7	7	-	/	7	7
R, ALLOC_ERROR_RC_LOG	/	7	7	/	/	/	7	/	/	7	/
R, ALLOC_ERROR_RC	/	8	8	/	/	/	8	/	/	8	/
S, DEALLOC, FLUSH|DATAFLU	/	8	>	>	>	>	/	>	>	8	>
S, DEALLOC, CONFIRM|DATACON	/	6	>	>	>	>	/	>	>	6	>
S, DEALLOC, ABND	/	8	8	8	8	8	/	8	>	8	8
S, REJECT, CONV	/	8	8	8	8	8	8	8	>	8	8
R, DEALLOCATED_IND	/	/	/	/	/	/	8	/	/	/	/
R, END_OF_LOG_IND	/	/	/	/	/	/	/	8	/	/	2
R, RECEIVE_DATA_&_SEND_IND	/	/	9	/	/	/	/	/	/	/	/

Figure 35. Half-duplex conversation state transitions

Note:

1. The initial state of the FSM is reset.
2. PEND_DEALL is a transient state. The application program can display these states using the APPCCMD CONTROL=TESTSTAT macroinstruction.
3. The action code specified at each intersection of this FSM matrix can be one of four values.

- A number indicates the next state that the FSM assumes if the matrix intersection is addressed.
- A dash (–) indicates that no state change is to occur.
- A greater-than sign (>) indicates an error condition (and causes a return code indicating STATE_ERROR to be returned to the application program).
- A slash (/) indicates a "cannot-occur" situation, for example, that this matrix intersection cannot be selected because of previous checks that will appear in supporting code.

4. An "S" in the inputs column indicates that the input is a result of an APPCCMD macroinstruction "sent" by the application program to VTAM. The inputs associated with an "S" are the values for control and qualify fields on the APPCCMD macroinstruction. An "R" indicates that the input is being "received" by VTAM from the partner LU.

For more information, refer to the individual descriptions of these macroinstructions in the z/OS Communications Server: SNA Programmer's LU 6.2 Reference. Note, however, that the FSM does not attempt to reflect any information concerning the issuance of an APPCCMD macroinstruction in this situation.

FSM_CONVERSATION_FDX

Inputs	0 RESET	80 FDX RESET	81 SEND/RECEIVE	82 SEND-ONLY	83 RECEIVE-ONLY	84 PEND SND/RCV LOG	85 PEND RCV-ONLY LOG	86 PEND RESET LOG
S, ALLOC	81	/	/	/	/	/	/	/
R, FMH-5	81	/	/	/	/	/	/	/
S, SEND, DATA	/	/	-	-	>	-	>	>
S, SEND, FLUSH|DATAFLU	/	/	-	-	>	-	>	>
S, SENDEXPD, DATA	/	/	-	-	-	-	-	-
S, SEND, ERROR	/	/	-	-	>	-	>	>
S, RCVEXPD	/	/	-	-	-	-	-	-
S, RECEIVE, SPEC	/	/	-	>	-	-	-	-
S, RECEIVE, ISPEC	/	/	-	>	-	-	-	-
R, PROGRAM_ERROR_RC	/	/	-	/	-	-	-	/
R, SERVICE_ERROR_RC	/	/	-	/	-	-	-	/
R, USER_ERROR_CODE_RC	/	/	-	/	-	-	-	/
R, SEND_ERROR_RC_LOG	/	/	84	/	85	-	-	/
R, DEALLOCATE_NORMAL_RC	/	/	82	/	80	82	80	/
R, ERROR_INDICATION_RC	/	/	-	80	/	-	/	/
R, DEALLOCATE_ABEND_RC	/	/	80	/	80	80	80	/
R, DEALLOC_ABND_RC_LOG	/	/	86	/	86	86	86	/
R, RESOURCE_FAILURE_RC	/	/	80	/	80	80	80	/
R, RESOURCE_FAIL_RC_LOG	/	/	86	/	86	86	86	/
R, ALLOC_ERROR_RC	/	/	80	/	80	80	80	/
R, ALLOC_ERROR_RC_LOG	/	/	86	/	86	86	86	/
S, DEALLOC, FLUSH|DATAFLU	/	/	83	80	>	85	>	>
S, DEALLOC, ABND	/	/	80	80	80	80	80	80
S, REJECT, CONV	/	/	80	80	80	80	80	80
R, END_OF_LOG_IND	/	/	/	/	/	81	83	80

Figure 36. Full-duplex conversation state transitions

Note:

1. The initial state of the FSM is reset.
2. The action code specified at each intersection of this FSM matrix can be one of four values.
   - A number indicates the next state that the FSM assumes if the matrix intersection is addressed. The state numbers for FSM_CONVERSATION_FDX are shown as hexadecimal values.
   - A dash (–) indicates that no state change is to occur.
   - A greater-than sign (>) indicates an error condition (and causes a return code indicating STATE_ERROR to be returned to the application program).
   - A slash (/) indicates a "cannot-occur" situation, for example, that this matrix intersection cannot be selected because of previous checks that will appear in supporting code.
3. An "S" in the inputs column indicates that the input is a result of an APPCCMD macroinstruction "sent" by the application program to VTAM. The inputs associated with an "S" are the values for control and qualify fields on the APPCCMD macroinstruction. An "R" indicates that the input is being "received" by VTAM from the partner LU.
4. The ERROR_INDICATION_RC can be returned to the application on an APPCCMD CONTROL=SEND or an APPCCMD CONTROL=DEALLOC, QUALIFY=FLUSH|DATAFLU macroinstruction. The other inputs received by the application can be returned only on an APPCCMD CONTROL=RECEIVE macroinstruction.

For more information, refer to the individual descriptions of these macroinstructions in the z/OS Communications Server: SNA Programmer's LU 6.2 Reference. Note, however, that the FSM does not attempt to reflect any information concerning the issuance of an APPCCMD macroinstruction in this situation.

Appendix B. APPCCMD macroinstruction overview

The following tables are a cross reference between specific APPCCMDs and the following items:

- Macroinstruction session and conversation information
- Information from the application to VTAM
- Information from VTAM to the application

Session and conversation information

Table 51 on page 289 provides the following information regarding APPCCMD macroinstructions and the following session and conversation information:

1. Initiates a session, if required, and if current limits allow
2. Allowed on any half-duplex conversation
3. Allowed on a half-duplex conversation established on a full-duplex-capable session
4. Allowed on a full-duplex conversation

Table 51. Session / conversation information

1= Initiates session, if required and if session limits allow 2= Allowed on any HDX conversation 3= Allowed on an HDX conversation - established on an FDX session 4= Allowed on an FDX conversation	**1**	**2**	**3**	**4**
ALLOC.				
ALLOCD	Y	Y	Y	Y
CONVGRP	N	Y	Y	Y
CONWIN	Y	Y	Y	Y
IMMED	N	Y	Y	Y
WHENFREE	Y	Y	Y	Y
CHECK	n/a	n/a	n/a	n/a
DEALLOC.				
ABNDPROG	n/a	Y	Y	Y
ABNDSERV	n/a	Y	Y	Y
ABNDTIME	n/a	Y	Y	Y
ABNDUSER	n/a	Y	Y	Y
CONFIRM	n/a	Y	Y	N
DATACON	n/a	Y	Y	N
DATAFLU	n/a	Y	Y	Y

Table 51. Session / conversation information (continued)

1= **Initiates session, if required and if session limits allow** **2=** **Allowed on any HDX conversation** **3=** **Allowed on an HDX conversation - established on an FDX session** **4=** **Allowed on an FDX conversation**	**1**	**2**	**3**	**4**
FLUSH	n/a	Y	Y	Y
DEALLOCQ.				
ABNDPROG	n/a	Y	Y	Y
ABNDSERV	n/a	Y	Y	Y
ABNDTIME	n/a	Y	Y	Y
ABNDUSER	n/a	Y	Y	Y
OPRCNTL.				
ACTSESS	N	n/a	n/a	n/a
CNOS	Y	n/a	n/a	n/a
DACTSESS	N	n/a	n/a	n/a
DEFINE	N	n/a	n/a	n/a
DISPLAY	N	n/a	n/a	n/a
RESTORE	N	n/a	n/a	n/a
PREALLOC.				
ALLOCD	Y	Y	Y	Y
CONVGRP	N	Y	Y	Y
CONWIN	Y	Y	Y	Y
IMMED	N	Y	Y	Y
WHENFREE	Y	Y	Y	Y
PREPRCV.				
CONFIRM	n/a	Y	Y	N
DATACON	n/a	Y	Y	N
DATAFLU	n/a	Y	Y	N
FLUSH	n/a	Y	Y	N
RCVEXPD.				
ANY	n/a	Y	Y	Y
IANY	n/a	Y	Y	Y
ISPEC	n/a	Y	Y	Y

Table 51. Session / conversation information (continued)

1= **Initiates session, if required and if session limits allow** **2=** **Allowed on any HDX conversation** **3=** **Allowed on an HDX conversation - established on an FDX session** **4=** **Allowed on an FDX conversation**	**1**	**2**	**3**	**4**
SPEC	n/a	Y	Y	Y
RCVFMH5.				
DATAQUE	n/a	Y	Y	Y
NULL	n/a	Y	Y	Y
QUEUE	n/a	Y	Y	Y
RECEIVE.				
ANY	n/a	Y	Y	Y
IANY	n/a	Y	Y	Y
ISPEC	n/a	Y	Y	Y
SPEC	n/a	Y	Y	Y
REJECT.				
CONV	n/a	Y	Y	Y
CONVGRP	n/a	Y	Y	Y
SESSION	n/a	n/a	n/a	n/a
RESETRCV.				
NULL	n/a	Y	Y	Y
SEND.				
CONFIRM	n/a	Y	Y	N
CONFRMD	n/a	Y	Y	N
DATA	n/a	Y	Y	Y
DATACON	n/a	Y	Y	N
DATAFLU	n/a	Y	Y	Y
ERROR	n/a	Y	Y	Y
FLUSH	n/a	Y	Y	Y
RQSEND	n/a	Y	Y	N
SENDEXPD.				
DATA	n/a	N	Y	Y
SENDFMH5.				

Table 51. Session / conversation information (continued)

1= **Initiates session, if required and if session limits allow** **2=** **Allowed on any HDX conversation** **3=** **Allowed on an HDX conversation - established on an FDX session** **4=** **Allowed on an FDX conversation**	**1**	**2**	**3**	**4**
NULL	n/a	Y	Y	Y
SENDRCV.				
DATAFLU	n/a	Y	Y	N
SETSESS.				
RESUME	n/a	Y	Y	N
SUSPEND	n/a	Y	Y	N
SYNCBEG	n/a	Y	Y	N
SYNCEND	n/a	Y	Y	N
TESTSTAT.				
ALL	n/a	Y	Y	Y
IALL	n/a	Y	Y	Y
ISPEC	n/a	Y	Y	Y
SPEC	n/a	Y	Y	Y

Information from the application to VTAM

Table 52 on page 293 provides the following information regarding APPCCMD macroinstructions and the flow of information from the application to VTAM:

1. Transfers normal data from the application to VTAM
2. Flushes (empties) SEND buffer (normal data) into the network
3. Possibly causes information to be sent into the network
4. Transfers expedited data from the application to VTAM
5. Causes a SEND indication to be sent to the partner
6. Causes a DEALLOC indication to be sent to the partner
7. Causes a CONFIRM indication to be sent to the partner
8. Causes a CONFIRMD indication to be sent to the partner
9. Causes an ERROR indication to be sent to the partner

Table 52. Flow from local application to VTAM

1= **Transfers normal data from application to VTAM** **2=** **Flushes SEND buffer (normal data) into network** **3=** **Possibly causes information to be sent into the network** **4=** **Transfers expedited data from application to VTAM** **5=** **Causes a SEND indication to be sent to the partner** **6=** **Causes a DEALLOC indication to be sent to the partner** **7=** **Causes a CONFIRM indication to be sent to the partner** **8=** **Causes a CONFIRMD indication to be sent to the partner** **9=** **Causes an ERROR indication to be sent to the partner**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
ALLOC (N|Y means - N for HDX sessions, Y for FDX sessions)									
ALLOCD	Y	N|Y	N|Y	N	N	N	N	N	N
CONVGRP	Y	N|Y	N|Y	N	N	N	N	N	N
CONWIN	Y	N|Y	N|Y	N	N	N	N	N	N
IMMED	Y	N|Y	N|Y	N	N	N	N	N	N
WHENFREE	Y	N|Y	N|Y	N	N	N	N	N	N
CHECK	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
DEALLOC.									
ABNDPROG	Y	Y	Y	N	N	Y	N	N	Y
ABNDSERV	Y	Y	Y	N	N	Y	N	N	Y
ABNDTIME	Y	Y	Y	N	N	Y	N	N	Y
ABNDUSER	Y	Y	Y	N	N	Y	N	N	Y
CONFIRM	N	Y	Y	N	N	Y	Y	N	N
DATACON	Y	Y	Y	N	N	Y	Y	N	N
DATAFLU	Y	Y	Y	N	N	Y	N	N	N
FLUSH	N	Y	Y	N	N	Y	N	N	N

Table 52. Flow from local application to VTAM (continued)

1= **Transfers normal data from application to VTAM** **2=** **Flushes SEND buffer (normal data) into network** **3=** **Possibly causes information to be sent into the network** **4=** **Transfers expedited data from application to VTAM** **5=** **Causes a SEND indication to be sent to the partner** **6=** **Causes a DEALLOC indication to be sent to the partner** **7=** **Causes a CONFIRM indication to be sent to the partner** **8=** **Causes a CONFIRMD indication to be sent to the partner** **9=** **Causes an ERROR indication to be sent to the partner**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
DEALLOCQ.									
ABNDPROG	Y	Y	Y	N	N	Y	N	N	Y
ABNDSERV	Y	Y	Y	N	N	Y	N	N	Y
ABNDTIME	Y	Y	Y	N	N	Y	N	N	Y
ABNDUSER	Y	Y	Y	N	N	Y	N	N	Y
OPRCNTL.									
ACTSESS	Y	N/a	Y	N	N/a	N/a	N/a	N/a	N/a
CNOS	Y	N/a	Y	N	N/a	N/a	N/a	N/a	N/a
DACTSESS	N	N/a	Y	N	N/a	N/a	N/a	N/a	N/a
DEFINE	Y	N/a	N	N	N/a	N/a	N/a	N/a	N/a
DISPLAY	N	N/a	N	N	N/a	N/a	N/a	N/a	N/a
RESTORE	N	N/a	N	N	N/a	N/a	N/a	N/a	N/a
PREALLOC (N|Y means - N for HDX sessions, Y for FDX sessions)									
ALLOCD	N	N	N	N	N	N	N	N	N
CONVGRP	N	N	N	N	N	N	N	N	N
CONWIN	N	N	N	N	N	N	N	N	N

Table 52. Flow from local application to VTAM (continued)

1= **Transfers normal data from application to VTAM** **2=** **Flushes SEND buffer (normal data) into network** **3=** **Possibly causes information to be sent into the network** **4=** **Transfers expedited data from application to VTAM** **5=** **Causes a SEND indication to be sent to the partner** **6=** **Causes a DEALLOC indication to be sent to the partner** **7=** **Causes a CONFIRM indication to be sent to the partner** **8=** **Causes a CONFIRMD indication to be sent to the partner** **9=** **Causes an ERROR indication to be sent to the partner**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
IMMED	N	N	N	N	N	N	N	N	N
WHENFREE	N	N	N	N	N	N	N	N	N
PREPRCV.									
CONFIRM	N	Y	Y	N	Y	N	Y	N	N
DATACON	Y	Y	Y	N	Y	N	Y	N	N
DATAFLU	Y	Y	Y	N	Y	N	N	N	N
FLUSH	N	Y	Y	N	Y	N	N	N	N
RCVEXPD.									
ANY	N	N	Y	N	N	N	N	N	N
IANY	N	N	Y	N	N	N	N	N	N
ISPEC	N	N	Y	N	N	N	N	N	N
SPEC	N	N	Y	N	N	N	N	N	N
RCVFMH5.									
DATAQUE	N	N	N	N	N	N	N	N	N
NULL	N	N	N	N	N	N	N	N	N
QUEUE	N	N	N	N	N	N	N	N	N

Table 52. Flow from local application to VTAM (continued)

1= **Transfers normal data from application to VTAM** **2=** **Flushes SEND buffer (normal data) into network** **3=** **Possibly causes information to be sent into the network** **4=** **Transfers expedited data from application to VTAM** **5=** **Causes a SEND indication to be sent to the partner** **6=** **Causes a DEALLOC indication to be sent to the partner** **7=** **Causes a CONFIRM indication to be sent to the partner** **8=** **Causes a CONFIRMD indication to be sent to the partner** **9=** **Causes an ERROR indication to be sent to the partner**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
RECEIVE.									
ANY	N	N	N	N	N	N	N	N	N
IANY	N	N	N	N	N	N	N	N	N
ISPEC	N	N	N	N	N	N	N	N	N
SPEC	N	Y	Y	N	Y	N	N	N	N
REJECT.									
CONV	N	N	N	N	N	N	N	N	Y
CONVGRP	N	N	N	N	N	N	N	N	Y
SESSION	N	N	N	N	N	N	N	N	Y
RESETRCV.									
NULL	N	N	N	N	N	N	N	N	N
SEND.									
CONFIRM	N	Y	Y	N	N	N	Y	N	N
CONFRMD	N	N	Y	N	N	N	N	Y	N
DATA	Y	N	Y	N	N	N	N	N	N
DATACON	Y	Y	Y	N	N	N	Y	N	N

Table 52. Flow from local application to VTAM (continued)

1= **Transfers normal data from application to VTAM** **2=** **Flushes SEND buffer (normal data) into network** **3=** **Possibly causes information to be sent into the network** **4=** **Transfers expedited data from application to VTAM** **5=** **Causes a SEND indication to be sent to the partner** **6=** **Causes a DEALLOC indication to be sent to the partner** **7=** **Causes a CONFIRM indication to be sent to the partner** **8=** **Causes a CONFIRMD indication to be sent to the partner** **9=** **Causes an ERROR indication to be sent to the partner**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
DATAFLU	Y	Y	Y	N	N	N	N	N	N
ERROR	Y	Y	Y	N	N	N	N	N	Y
FLUSH	N	Y	Y	N	N	N	N	N	N
RQSEND	N	N	Y	N	N	N	N	N	N
SENDEXPD.									
DATA	N	N	Y	Y	N	N	N	N	N
SENDFMH5.									
NULL	Y	N|Y	N|Y	N	N	N	N	N	N
SENDRCV.									
DATAFLU	Y	Y	Y	N	Y	N	N	N	N
SETSESS.									
RESUME	N	N	N	N	N	N	N	N	N
SUSPEND	N	N	N	N	N	N	N	N	N
SYNCBEG	N	N	N	N	N	N	N	N	N
SYNCEND	N	N	N	N	N	N	N	N	N
TESTSTAT.									

Table 52. Flow from local application to VTAM (continued)

1= **Transfers normal data from application to VTAM** **2=** **Flushes SEND buffer (normal data) into network** **3=** **Possibly causes information to be sent into the network** **4=** **Transfers expedited data from application to VTAM** **5=** **Causes a SEND indication to be sent to the partner** **6=** **Causes a DEALLOC indication to be sent to the partner** **7=** **Causes a CONFIRM indication to be sent to the partner** **8=** **Causes a CONFIRMD indication to be sent to the partner** **9=** **Causes an ERROR indication to be sent to the partner**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
ALL	N	N	N	N	N	N	N	N	N
IALL	N	N	N	N	N	N	N	N	N
ISPEC	N	N	N	N	N	N	N	N	N
SPEC	N	N	N	N	N	N	N	N	N

Information flow from VTAM to the application

Table 53 on page 299 provides the following information regarding the flow of information from VTAM to the application:

1. Transfers normal data from VTAM to the application
2. Transfers expedited data from VTAM to the application
3. Receives SEND indication from the partner
4. Receives DEALLOC indication from the partner
5. Receives CONFIRM indication from the partner
6. Receives CONFIRMD indication from the partner
7. Receives ERROR indication from the partner
8. Receives SIGRCV and SIGDATA (RTSRTRN=BOTH specified)
9. Receives SIGRCV and SIGDATA (RTSRTRN=EXPD specified)

Table 53. Flow from VTAM to local application

1= **Transfers normal data from VTAM to application** **2=** **Transfers expedited data from VTAM to application** **3=** **Receives SEND indication from the partner** **4=** **Receives DEALLOC indication from the partner** **5=** **Receives CONFIRM indication from the partner** **6=** **Receives CONFIRMD indication from the partner** **7=** **Receives ERROR indication from the partner** **8=** **Receives SIGRCV and SIGDATA (RTSRTRN=BOTH)** **9=** **Receives SIGRCV and SIGDATA (RTSRTRN=EXPD)**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
ALLOC.									
ALLOCD	N	N	N	N	N	N	N	N	N
CONVGRP	N	N	N	N	N	N	N	N	N
CONWIN	N	N	N	N	N	N	N	N	N
IMMED	N	N	N	N	N	N	N	N	N
WHENFREE	N	N	N	N	N	N	N	N	N
CHECK	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
DEALLOC.									
ABNDPROG	N	N	N	N	N	N	N	N	N
ABNDSERV	N	N	N	N	N	N	N	N	N
ABNDTIME	N	N	N	N	N	N	N	N	N
ABNDUSER	N	N	N	N	N	N	N	N	N
CONFIRM	N	N	N	N	N	Y	Y	Y	N
DATACON	N	N	N	N	N	Y	Y	Y	N
DATAFLU	N	N	N	N	N	N	Y	Y	N
FLUSH	N	N	N	N	N	N	Y	N	N

Table 53. Flow from VTAM to local application (continued)

1= **Transfers normal data from VTAM to application** **2=** **Transfers expedited data from VTAM to application** **3=** **Receives SEND indication from the partner** **4=** **Receives DEALLOC indication from the partner** **5=** **Receives CONFIRM indication from the partner** **6=** **Receives CONFIRMD indication from the partner** **7=** **Receives ERROR indication from the partner** **8=** **Receives SIGRCV and SIGDATA (RTSRTRN=BOTH)** **9=** **Receives SIGRCV and SIGDATA (RTSRTRN=EXPD)**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
DEALLOCQ.									
ABNDPROG	N	N	N	N	N	N	N	N	N
ABNDSERV	N	N	N	N	N	N	N	N	N
ABNDTIME	N	N	N	N	N	N	N	N	N
ABNDUSER	N	N	N	N	N	N	N	N	N
OPRCNTL.									
ACTSESS	N	N	N/a	N/a	N/a	N/a	N/a	N/a	N/a
CNOS	Y	N	N/a	N/a	N/a	N/a	N/a	N/a	N/a
DACTSESS	N	N	N/a	N/a	N/a	N/a	N/a	N/a	N/a
DEFINE	N	N	N/a	N/a	N/a	N/a	N/a	N/a	N/a
DISPLAY	Y	N	N/a	N/a	N/a	N/a	N/a	N/a	N/a
RESTORE	Y	N	N/a	N/a	N/a	N/a	N/a	N/a	N/a
PREALLOC.									
ALLOCD	N	N	N	N	N	N	N	N	N
CONVGRP	N	N	N	N	N	N	N	N	N
CONWIN	N	N	N	N	N	N	N	N	N

Table 53. Flow from VTAM to local application (continued)

1= **Transfers normal data from VTAM to application** **2=** **Transfers expedited data from VTAM to application** **3=** **Receives SEND indication from the partner** **4=** **Receives DEALLOC indication from the partner** **5=** **Receives CONFIRM indication from the partner** **6=** **Receives CONFIRMD indication from the partner** **7=** **Receives ERROR indication from the partner** **8=** **Receives SIGRCV and SIGDATA (RTSRTRN=BOTH)** **9=** **Receives SIGRCV and SIGDATA (RTSRTRN=EXPD)**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
IMMED	N	N	N	N	N	N	N	N	N
WHENFREE	N	N	N	N	N	N	N	N	N
PREPRCV.									
CONFIRM	N	N	N	N	N	Y	Y	N	N
DATACON	N	N	N	N	N	Y	Y	Y	N
DATAFLU	N	N	N	N	N	N	Y	Y	N
FLUSH	N	N	N	N	N	N	N	N	N
RCVEXPD.									
ANY	N	Y	N	N	N	N	N	Y	Y
IANY	N	Y	N	N	N	N	N	Y	Y
ISPEC	N	Y	N	N	N	N	N	Y	Y
SPEC	N	Y	N	N	N	N	N	Y	Y
RCVFMH5.									
DATAQUE	Y	N	N	N	N	N	Y	N	N
NULL	Y	N	N	N	N	N	Y	N	N
QUEUE	Y	N	N	N	N	N	Y	N	N

Table 53. Flow from VTAM to local application (continued)

1= **Transfers normal data from VTAM to application** **2=** **Transfers expedited data from VTAM to application** **3=** **Receives SEND indication from the partner** **4=** **Receives DEALLOC indication from the partner** **5=** **Receives CONFIRM indication from the partner** **6=** **Receives CONFIRMD indication from the partner** **7=** **Receives ERROR indication from the partner** **8=** **Receives SIGRCV and SIGDATA (RTSRTRN=BOTH)** **9=** **Receives SIGRCV and SIGDATA (RTSRTRN=EXPD)**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
RECEIVE.									
ANY	Y	N	Y	Y	Y	N	Y	Y	N
IANY	Y	N	Y	Y	Y	N	Y	Y	N
ISPEC	Y	N	Y	Y	Y	N	Y	Y	N
SPEC	Y	N	Y	Y	Y	N	Y	Y	N
REJECT.									
CONV	N	N	N	N	N	N	N	N	N
CONVGRP	N	N	N	N	N	N	N	N	N
SESSION	N	N	N	N	N	N	N	N	N
RESETRCV.									
NULL	N	N	N	N	N	N	N	N	N
SEND.									
CONFIRM	N	N	N	N	N	Y	Y	Y	N
CONFRMD	N	N	N	N	N	N	N	N	N
DATA	N	N	N	N	N	N	Y	Y	N
DATACON	N	N	N	N	N	Y	Y	Y	N

Table 53. Flow from VTAM to local application (continued)

1= **Transfers normal data from VTAM to application** **2=** **Transfers expedited data from VTAM to application** **3=** **Receives SEND indication from the partner** **4=** **Receives DEALLOC indication from the partner** **5=** **Receives CONFIRM indication from the partner** **6=** **Receives CONFIRMD indication from the partner** **7=** **Receives ERROR indication from the partner** **8=** **Receives SIGRCV and SIGDATA (RTSRTRN=BOTH)** **9=** **Receives SIGRCV and SIGDATA (RTSRTRN=EXPD)**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
DATAFLU	N	N	N	N	N	N	Y	Y	N
ERROR	N	N	N	Y	N	N	Y	Y	N
FLUSH	N	N	N	N	N	N	Y	N	N
RQSEND	N	N	N	N	N	N	N	N	N
SENDEXPD.									
DATA	N	N	N	N	N	N	N	Y	Y
SENDFMH5.									
NULL	N	N	N	N	N	N	N	N	N
SENDRCV.									
DATAFLU	Y	N	Y	Y	Y	N	Y	Y	N
SETSESS.									
RESUME	N	N	N	N	N	N	N	N	N
SUSPEND	N	N	N	N	N	N	N	N	N
SYNCBEG	N	N	N	N	N	N	N	N	N
SYNCEND	N	N	N	N	N	N	N	N	N
TESTSTAT (I mean availability indicated in Status Data Structure but not altered by this macroinstruction)									

Table 53. Flow from VTAM to local application (continued)

1= **Transfers normal data from VTAM to application** **2=** **Transfers expedited data from VTAM to application** **3=** **Receives SEND indication from the partner** **4=** **Receives DEALLOC indication from the partner** **5=** **Receives CONFIRM indication from the partner** **6=** **Receives CONFIRMD indication from the partner** **7=** **Receives ERROR indication from the partner** **8=** **Receives SIGRCV and SIGDATA (RTSRTRN=BOTH)** **9=** **Receives SIGRCV and SIGDATA (RTSRTRN=EXPD)**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
ALL	I	I	I	I	I	I	I	I	I
IALL	I	I	I	I	I	I	I	I	I
ISPEC	I	I	I	I	I	I	I	I	I
SPEC	I	I	I	I	I	I	I	I	I

Appendix C. Example of a sample LU 6.2 application program

This appendix contains the source code of a sample LU 6.2 application program and a console log created from the execution of the sample program.

The source code is designed to generate two sample programs capable of using VTAM LU 6.2 macroinstruction for communication. The program presented here is designed for a z/OS operating system. The sample uses the WTO macroinstruction as a trace mechanism to identify various code points being executed.

A console log of the two applications running is displayed after the source code. The console log shows the activity each sample program performs in order to establish conversations, communicate data over the conversations, and terminate two conversations.

In order to better isolate the processing being performed by the two applications, information regarding one application is on the left side of the console log and information regarding the other application is on the right side of the console log. The console log represents the whole run time integrated vertically.

The interaction between the two programs demonstrates an application program:

- Identifying itself to VTAM
- Negotiating session limits
- Processing the ATTN exit routine
- Allocating and deallocating a conversation
- Sending and receiving logical records
- Changing conversation states

Sample VTAM LU 6.2 application program

```
*********************************************************************
***                                                               ***
***  INTRODUCTION:  This sample application is provided to give   ***
***     potential VTAM LU 6.2 application writers a simple        ***
***     example of the major processes needed to facilitate LU 6.2 ***
***     conversations.                                            ***
***                                                               ***
***     One of the main objectives of this sample is to show      ***
***     the logic of a successful conversation.  However          ***
***     keeping the sample simple and easy to follow was made     ***
***     possible by omitting many error paths that can occur      ***
***     during a conversation.  For instance a TPEND exit, which  ***
***     notifies an application of VTAM services ending, was      ***
***     not provided.  Testing to insure a macro instruction      ***
***     was accepted successfully and recovery routines to address ***
***     completion errors were not provided.  Much of a user      ***
***     written application deals with error conditions as opposed ***
***     to successful communication.                              ***
***                                                               ***
***     Note that the application assumes no processing           ***
***     contentions when it allocates storage for                 ***
***     containing conversation related information.              ***
***                                                               ***
***     Much of the logic which complicates the transaction       ***
***     program processing has been replaced by a WTOR to the     ***
***     console operator to dictate the next communication        ***
***     action a transaction program should take.                 ***
***                                                               ***
***  BASIC STRUCTURE of the SAMPLE:  The MAIN code will deal      ***
***     with managing the ACB and upon request issue the          ***
***     necessary macroinstructions to start a conversation.  The ***
***     rest of the routines will be executed from the completion ***
***     RPL exit of a previous macro instruction or from the      ***
***     ACB ATTN exit.  All exits are executed unauthorized in    ***
```

```
***     order to allow WTO/WTOR macroinstruction to work           ***
***     successfully.  Most of the VTAM macroinstructions are      ***
***     issued OPTCD=ASY and EXIT=address.  This allows the        ***
***     application to take advantage of the processing tasks      ***
***     created by VTAM for RPL completions.  This simplifies      ***
***     the application from doing ATTACHes for the transaction    ***
***     programs but still achieves an asynchronous like           ***
***     operation.                                                 ***
***                                                                ***
```

```
***  OTHER DEFINITION NEEDED:                                      ***
***                                                                ***
***     The following are the APPL definitions that would support  ***
***     the two applications that can be generated by this sample  ***
***     code.  Even though the SRBEXIT=NO is defaulted, it is      ***
***     still shown here, to highlight that the ACB exits are      ***
***     executed unauthorized:                                     ***
***                                                                ***
***                                                        col 72 ***
***                                                             | ***
***                                                            V ***
***          VBUILD TYPE=APPL                                    ***
*** ACBVICKY APPL APPC=YES,SRBEXIT=NO,                          X ***
***               DSESLIM=2,DMINWNL=1,DMINWNR=1                  ***
*** ACBGARY  APPL APPC=YES,SRBEXIT=NO,                          X ***
***               DSESLIM=2,DMINWNL=1,DMINWNR=1                  ***
***                                                              ***
***                                                              ***
***     The following is the MODEENT macro that was added to     ***
***     logon mode table (ISTINCLM).  This entry was patterned   ***
***     after the SNASVCMG entry which is shipped with the VTAM  ***
***     default logmode table.  In this example, the max RU      ***
***     size was changed to 512 bytes.                           ***
***                                                              ***
***                                                              ***
*** SNASVCMG MODEENT LOGMODE=SNASVCMG,FMPROF=X'13',TSPROF=X'07', X ***
***            PRIPROT=X'B0',SECPROT=X'B0',COMPROT=X'D0B1',      X ***
***            RUSIZES=X'8585',ENCR=B'0000',TYPE=0,              X ***
***            PSERVIC=X'06020000000000000000300'                  ***
*** LU62CONV MODEENT LOGMODE=LU62CONV,FMPROF=X'13',TSPROF=X'07', X ***
***            PRIPROT=X'B0',SECPROT=X'B0',COMPROT=X'D0B1',      X ***
***            RUSIZES=X'8686',ENCR=B'0000',TYPE=0,              X ***
***            PSERVIC=X'06020000000000000000300'                  ***
***                                                              ***
********************************************************************
         EJECT
```

```
********************************************************************
***  ASSEMBLER NOTES:  The following assembler variable (&SIDE) is ***
***  used to include appropriate instructions for building one of  ***
***  two VTAM LU 6.2 sample applications.  This assembler variable ***
***  can be set to "LEFT " or "RIGHT".  The result of each value   ***
***  will generate the following information:                      ***
***                                                                ***
***    &SIDE SETC 'LEFT ' ==> will generate a CSECT name of        ***
***      APPCAPPL, OPEN an ACB with an APPLID value of "ACBVICKY"  ***
***      and pre-allocate storage to contain conversation-related  ***
***      information.  Names, JEFFERY and KIMBERLY, have been      ***
***      assigned to each pre-allocated storage.                   ***
***                                                                ***
***    &SIDE SETC 'RIGHT' ==> will generate a CSECT name of        ***
***      APPCAPPR, OPEN an ACB with an APPLID value of "ACBGARY"   ***
***      and pre-allocate storage to contain conversation-related  ***
***      information.  Names, JAIME and TIMOTHY, have been         ***
***      assigned to each pre-allocated storage.                   ***
***                                                                ***
***  Assembler H was used to assemble this source which allows     ***
***  symbols to be used before they are defined.  If Assembler F   ***
***  is used, the DSECTs will need to be moved ahead of the code   ***
***  that references the symbols defined in the DSECTs.            ***
********************************************************************
         SPACE 3
         LCLC  &SIDE
&SIDE    SETC  'LEFT '
* &SIDE    SETC  'RIGHT'
         SPACE 3
         AIF   ('&SIDE' EQ 'LEFT ').LCSECT
APPCAPPR CSECT
         AGO   .ECSECT
.LCSECT  ANOP
```

```
APPCAPPL CSECT
.ECSECT  ANOP
         SPACE 3
*************************************************************************
*                                                                       *
*    Register EQUATES                                                   *
*                                                                       *
*************************************************************************
R0       EQU   0
R1       EQU   1
R2       EQU   2
R3       EQU   3
R4       EQU   4
R5       EQU   5
R6       EQU   6
R7       EQU   7
R8       EQU   8
R9       EQU   9
R10      EQU   10
R11      EQU   11
R12      EQU   12
R13      EQU   13
R14      EQU   14
R15      EQU   15
         EJECT
```

***

```
*************************************************************************
*                                                                       *
*    Initialization Section - Perform standard linkage convention       *
*    and establish addressability for the program and RPL and           *
*    RPL extension.                                                     *
*                                                                       *
*      Registers on entry:                                              *
*                R13 address of an 18 word save area                    *
*                R14 return address                                     *
*                R15 entry address of this program                      *
*************************************************************************
         SPACE 3
MAIN     EQU   *
         STM   R14,R12,12(R13)         Save registers
         LR    R12,R15                 Establish base for MAIN routine
         AIF   ('&SIDE' EQ 'LEFT ').LUSING
         USING APPCAPPR,R12
         AGO   .EUSING
.LUSING  ANOP
         USING APPCAPPL,R12
.EUSING  ANOP
         CNOP  0,4                     Fullword boundary
         BAL   R1,*+76                 Branch around save area
         DC    18F'0'                  Save area for macroinstruction
         ST    R1,8(0,R13)             Chain
         ST    R13,4(0,R1)               save area
         LR    R13,R1                  Set register 13 to save area
         LA    R9,MAINRPL              Establish addressability
         USING IFGRPL,R9                  to MAIN routine RPL
         LA    R8,MAINRPL6             Establish addressability
         USING ISTRPL6X,R8                to MAIN routine RPL Extension
         SPACE 3
```

***

```
*************************************************************************
*                                                                       *
*    Issue OPEN macroinstruction to identify this program to VTAM.      *
*    The OPEN macroinstruction references a VTAM ACB with a label       *
*    of ACB.  The ACB is VTAM's representation of the LU.  All VTAM     *
*    macros used will reference this ACB.                               *
*                                                                       *
*************************************************************************
         SPACE 3
         WTO   'ISSUING OPEN ACB MACRO REQUEST',ROUTCDE=(1)
         XR    R15,R15                 Initialize register 15 = 0
         OPEN  ACB
         LTR   R15,R15                 Test OPEN
         BZ    OPENOK                  Branch if successful
         WTO   'OPEN ACB MACRO REQUEST FAILED ',ROUTCDE=(1)
         B     MAINRETN                Branch to MAIN return
         SPACE 3
```

```
*********************************************************************
*                                                                   *
*    Issue SETLOGON macroinstruction to enable VTAM to start        *
*    accepting LU 6.2 session-initiation request on behalf of the   *
*    application program.  The RPL operand of this and other        *
*    macroinstructions specifies the request parameter list (RPL)   *
*    that is used to send and receive information about the         *
*    macroinstruction to and from VTAM.                             *
*                                                                   *
*********************************************************************
         SPACE 3
```

```
OPENOK   EQU   *
         WTO   'ISSUING SETLOGON MACRO REQUEST',ROUTCDE=(1)
         SETLOGON RPL=MAINRPL,OPTCD=START
         CLI   RPLRTNCD,USFAOK         RTNCD = X'00' ?
         BNE   SETLOGFL                branch if no
         CLI   RPLFDB2,USFAOOK         FDB2 = X'00'
         BE    LOOP                    branch if yes
SETLOGFL EQU   *
         WTO   'SETLOGON MACRO REQUEST FAILED ',ROUTCDE=(1)
         B     MAINRETN
         EJECT
*********************************************************************
*                                                                   *
*    In any application a user would write, there will be some      *
*    stimulus which will initiate a transaction program.  That      *
*    stimulus will be simulated with a reply from the following     *
*    WTOR macro request.  A reply of "START" will drive code to     *
*    start a conversation.  A reply of "CLOSE" will drive code to   *
*    terminate this application.                                    *
*                                                                   *
*********************************************************************
         SPACE 3
LOOP     EQU   *
         MVC   REPLY,=CL8'        '    Initialize reply storage
         XC    MAINECB,MAINECB         Initialize WTOR ECB
         WTOR  'Enter START to start a transaction or CLOSE to close thX
               e ACB',REPLY,8,MAINECB,ROUTCDE=(1)
         WAIT  ECB=MAINECB             Wait for reply
         CLC   REPLY,=CL8'CLOSE'       Operator reply close?
         BE    MAINEND                 Branch yes
         CLC   REPLY,=CL8'START   '    Operator wants to start a
*                                      conversation?
         BE    STARTUP                 Branch yes
         WTO   'OPERATOR REPLY IS INVALID     ',ROUTCDE=(1)
         B     LOOP
         EJECT
```

```
*********************************************************************
*                                                                   *
*    In LU 6.2 architecture before a conversation can be started    *
*    the LUs must first have established some rules and limits      *
*    for the sessions that will be used by the transaction programs.*
*    Therefore this program determines if these rules and limits    *
*    have been established, and if not, issues the appropriate      *
*    macroinstructions to cause this process to happen.  This sample*
*    application will be concerned with LU 6.2 sessions established *
*    using the LOGMODE named "LU62CONV".                            *
*    The APPCCMD CONTROL=OPRCNTL, QUALIFY=CNOS is the               *
*    macroinstruction used to cause the process for establishing    *
*    these rules and limits.                                        *
*                                                                   *
*********************************************************************
         SPACE 3
STARTUP  EQU   *
         CLI   LUSTATE,LUCNOSD         As a CNOS set establishing
*                                      session limits > 0 been
*                                      processed?
         BE    DOALLOC                 Branch yes (bypass CNOS)
         MVC   RPL6LU,LUNAME           Set RPL extension LU name
         MVC   RPL6MODE,LOGMODE        Set RPL extension LOGMODE
         WTO   'ISSUING CONTROL=OPRCNTL,QUALIFY=CNOS MACRO REQUEST',ROUX
               TCDE=(1)
         APPCCMD CONTROL=OPRCNTL,QUALIFY=CNOS,                         X
               RPL=MAINRPL,AAREA=MAINRPL6,                             X
               AREA=0,RECLEN=0,                                        X
               OPTCD=SYN
         SPACE 3
*********************************************************************
```

```
*                                                                     *
*  Verify macroinstruction completed successfully.  If it did not,   *
*  provide notification of its failure.                               *
*                                                                     *
***********************************************************************
         CLI   RPLRTNCD,USFAOK         RTNCD = X'00' ?
         BNE   CNOSFL                  branch no
         CLI   RPLFDB2,USF6APPC        FDB2 = X'0B' ?
         BNE   CNOSFL                  Branch no
         CLC   RPL6RCPR,=AL2(USF6OK)   RCPRI = X'00'?
         BNE   CNOSFL                  branch no
         MVI   LUSTATE,LUCNOSD         set LU state to CNOSed
         B     DOALLOC                 branch to do the allocate
CNOSFL   EQU   *
         SPACE 3
***********************************************************************
*                                                                     *
*  To simplify this sample we will assume an unsuccessful CNOS       *
*  implies the partner LU is not available.                           *
*                                                                     *
***********************************************************************
         WTO   'CONTROL=OPRCNTL,QUALIFY=CNOS MACRO REQUEST FAILED',ROUTX
               CDE=(1)
         B     LOOP
         EJECT
```

```
***********************************************************************
*                                                                     *
*    Once a successful CNOS has been achieved, then the start         *
*    transaction process can be performed.  This sample program       *
*    will first find an available conversation entry in the local      *
*    conversation table, which will be used to maintain knowledge      *
*    of the conversation.  The conversation entry will consist of      *
*    the storage needed to issue VTAM macroinstruction for the         *
*    conversation.                                                     *
*                                                                      *
*    The APPCCMD CONTROL=ALLOC macroinstruction will be issued to      *
*    initiate a conversation.  The AREA field will point to an         *
*    FMH-5 which is used to identify the transaction processing        *
*    this conversation applies.                                        *
*                                                                      *
*    Most of the macroinstruction this sample program issues will      *
*    complete asynchronously with an exit which VTAM will invoke       *
*    when the request has finished.                                    *
*                                                                      *
*   This sample will also utilize the USERFLD of the RPL6 to           *
*   pass the address of the conversation entry to the RPL exit         *
*   specified on the request.                                          *
*                                                                      *
*    NOTE: This sample program will assume that two conversations      *
*    will "NOT" be started at the same time.  This implies that the    *
*    ATTN FMH-5 exit and this procedure will not overlap during        *
*    execution.                                                        *
*                                                                      *
***********************************************************************
         SPACE 3
DOALLOC  EQU   *
         LA    R6,2(0,0)               R6 assigned to 2 (#entries)
         L     R10,=A(CONVTBL)         R10 address of conversation
*                                      table (1st entry)
         USING CONVTBLD,R10            Establish addressability
*                                      of conversation entry
TSTOPEN  EQU   *
         CLI   CONVSTAT,CONVOPEN       Entry available ?
         BE    GOTONE                  Branch yes
         LA    R10,CONVLEN(0,R10)      Address next entry
         BCT   R6,TSTOPEN              Branch to TESTOPEN if we
*                                      haven't reached end of table.
         SPACE 1
         WTO   'ALL CONVERSATION ENTRIES IN USE',ROUTCDE=(1)
         B     LOOP                    return to mainline prompt
         SPACE 3
GOTONE   EQU   *                       Have conversation entry
```

```
         MVI   CONVSTAT,CONVACT        Make entry as allocated
         MVC   CONVRPL,MODLRPL         Initialize RPL
         MVC   CONVRPL6,MODLRPL6       Initialize RPL6
         LA    R1,CONVRPL6             Establish RPL6 basing
         MVC   RPL6LU-ISTRPL6X(L'RPL6LU,R1),LUNAME    Set LUNAME
*                                      field in RPL6
```

```
         MVC   RPL6MODE-ISTRPL6X(L'RPL6MODE,R1),LOGMODE  Set LOGMODE
*                                    field in RPL6
         MVC   CONVAREA(L'MODLFMH5),MODLFMH5     Initialize AREA with
*                                    a valid FMH-5
         SPACE 1
         WTO   'ISSUING CONTROL=ALLOC,QUALIFY=ALLOCD MACRO REQUEST',ROUX
               TCDE=(1)
         SPACE 1
         APPCCMD CONTROL=ALLOC,QUALIFY=ALLOCD,                         X
               RPL=CONVRPL,AAREA=CONVRPL6,                             X
               AREA=CONVAREA,RECLEN=L'MODLFMH5,                        X
               OPTCD=ASY,EXIT=AEXT,USERFLD=(R10)
         SPACE 1
         LTR   R15,R15                ALLOC request accepted OK?
         BZ    LOOP                   Branch if yes
         SPACE 1
         WTO   'CONTROL=ALLOC,QUALIFY=ALLOCD MACRO REQUEST FAILED',ROUTX
               CDE=(1)
         SPACE 1
         B     LOOP                   Go back to WTOR prompt
         EJECT
```

```
*********************************************************************
*                                                                   *
*    This termination routine is making the assumption that all     *
*    conversations have completed successfully.  Therefore the work *
*    remaining is to CNOS the LOGMODE mode LU62CONV session limits  *
*    to zero and then CNOS the SNASVCMG limits to zero.  Once the   *
*    the CNOSes have completed the ACB will be CLOSED and then the  *
*    application will return to the system.                         *
*                                                                   *
*********************************************************************
         SPACE 3
MAINEND  EQU   *
         CLI   LUSTATE,LUCNOSD        Have session limits been set?
         BNE   DOCLOSE                branch if not
         MVC   RPL6LU,LUNAME          Set RPL extension LU name
         MVC   RPL6MODE,LOGMODE       Set RPL extension LOGMODE
         XC    MAINAREA,MAINAREA      initializing the AREA to zero
*                                     will produce a CNOS structure
*                                     with session limits set to zero
         LA    R1,MAINAREA            Address AREA storage
         SPACE 1
         WTO   'RESETTING SESSION LIMITS FOR LU62CONV MODE       ',ROUX
               TCDE=(1)
         SPACE 1
         APPCCMD CONTROL=OPRCNTL,QUALIFY=CNOS,                         X
               ACB=ACB,RPL=MAINRPL,AAREA=MAINRPL6,                     X
               AREA=MAINAREA,RECLEN=SLCLEN,                            X
               OPTCD=SYN
         SPACE 1
         MVC   RPL6MODE,=CL8'SNASVCMG' Set to CNOS the SNASVCMG mode
         SPACE 1
         WTO   'RESETTING SESSION LIMITS FOR SNASVCMG MODE       ',ROUX
               TCDE=(1)
         SPACE 1
         APPCCMD CONTROL=OPRCNTL,QUALIFY=CNOS,                         X
               ACB=ACB,RPL=MAINRPL,AAREA=MAINRPL6,                     X
               OPTCD=SYN
         SPACE 3
DOCLOSE  EQU   *
         WTO   'ISSUING CLOSE ACB MACRO REQUEST        ',ROUTCDE=(1)
         CLOSE ACB                    Close ACB
         SPACE 3
*********************************************************************
*      Registers on exit:                                           *
*                R0-R14 values that were set upon entry             *
*                R15 set to zero                                    *
*********************************************************************
         SPACE 3
MAINRETN DS    0H
         L     R13,4(0,R13)           Reload register 13
         LM    R14,R12,12(R13)        Reload the remaining registers
         XR    R15,R15                Set return register to 0
         BR    R14                    Return to system dispatcher
         DROP  R8,R9,R10,R12
         EJECT
```

```
*********************************************************************
*                                                                   *
```

```
*    MAIN routine data area                                          *
*                                                                    *
**********************************************************************
         SPACE 3
REPLY    DC    CL8' '                  WTOR REPLY area
MAINECB  DC    F'0'                    WTOR ECB
         SPACE 3
MAINRPL  RPL   AM=VTAM,ACB=ACB         MAIN task RPL
MAINRPL6 ISTRPL6                       MAIN task RPL6
MAINAREA DC    XL(SLCLEN)'00'          MAIN task AREA
         SPACE 3
         DS    0F
MODLRPL  RPL   AM=VTAM,ACB=ACB         Model RPL
MODLRPL6 ISTRPL6 CONMODE=CS,FILL=LL,LOGMODE=LU62CONV   Model RPL6
MODLFMH5 DC    XL21'150502FF0003D0000008C140D7C5D9E2D6D5000000' Model
*                                      FMH-5
         SPACE 3
ACB      ACB   AM=VTAM,                                                X
               EXLST=EXLST,                                           X
               APPLID=APPLID
EXLST    EXLST AM=VTAM,                                                X
               ATTN=ATTNEXIT
APPLID   DC    AL1(8)
         AIF   ('&SIDE' EQ 'LEFT ').LAPPLID
         DC    CL8'ACBGARY '
         AGO   .EAPPLID
.LAPPLID ANOP
         DC    CL8'ACBVICKY'
.EAPPLID ANOP
         LTORG
         EJECT
```

```
**********************************************************************
*                                                                    *
*        ATTN EXIT Routine                                           *
*                                                                    *
* The ATTN Exit is driven by VTAM to notify the application when     *
* certain events have happened.  The exit's address is supplied in   *
* EXLST control block, which is pointed to by the ACB that was       *
* OPENed.  The three events that are reported are 1) the reception   *
* of an FMH-5, 2) the processing of a CNOS transaction, and 3) the   *
* unbind of the last LU 6.2 session of a mode group.                 *
*                                                                    *
* VTAM enters the ATTN exit with a read-only RPL, which points to    *
* a read-only RPL extension.  The RPL extension contains the name    *
* of the partner LU and logon mode.                                  *
*                                                                    *
* Of the three events, this sample application will be interested in *
* FMH-5 arrivals and CNOS processing.  FMH-5 arrivals will cause     *
* this exit to issue a RCVFMH5 APPCCMD macroinstruction.             *
* For CNOS events this exit will note if session limits are being    *
* initialized or reset.                                              *
*                                                                    *
*  ON ENTRY:                                                         *
*       R1 - address of a 6 word parameter list as documented in     *
*            in the LU 6.2 programming manual                        *
*       R14 - Return address when processing is finished             *
*       R15 - Address of this ATTN exit.                             *
*                                                                    *
**********************************************************************
         SPACE 3
ATTNEXIT DS    0H
         USING ATTNEXIT,R15            Temporary base for this routine
         CNOP  0,4                     Fullword alignment
         BAL   R13,*+76                branch around save area
         DC    18F'0'                  Set up save area chain
         STM   R14,R12,12(R13)         Store current registers
         LR    R12,R15                 Establish normal base register
         DROP  R15                        for this
         USING ATTNEXIT,R12                 ATTN exit
         CNOP  0,4                     Fullword alignment
         BAL   R15,*+76                branch around save area
         DC    18F'0'                  Save area for called routines
*                                      and macro requests
         ST    R15,8(0,R13)            chain save areas
         ST    R13,4(0,R15)               together
         LR    R13,R15                 set R13 to second savearea
         LR    R11,R1                  preserve parameter address
         SPACE 3
         WTO   'ENTERING ATTN ACB EXIT ROUTINE          ',ROUTCDE=(1)
         SPACE 3
```

```
         L     R9,16(R11)              Load address of read only RPL
         USING IFGRPL,R9               establish base register
         L     R8,RPLAAREA             Load address of read only RPL6
         USING ISTRPL6X,R8             establish base register
         CLC   RPL6LU,LUNAME           Verify partner LU name
         BE    LUOK                    Branch if OK
         EX    0,*                     0C3 Definition error with
*                                      partner LU
```

```
LUOK     EQU   *
         CLC   RPL6MODE,LOGMODE        Is LOGMODE = LU62CONV
         BE    MODEOK                  Branch if yes
         CLC   RPL6MODE,=CL8'SNASVCMG' Is this SNASVCMG LOGMODE
         BE    MODEOK                  Branch if yes
         EX    0,*                     0C3 - Definition erro with
*                                      partner LU
MODEOK   EQU   *
         CLC   12(4,R11),=CL4'CNOS'    Is this a CNOS event?
         BE    ATTNCNOS                branch if yes to CNOS process
         CLC   12(4,R11),=CL4'FMH5'    Is this an FMH-5 reception ?
         BE    ATTNFMH5                branch if yes to FMH-5 process
         CLC   12(4,R11),=CL4'LOSS'    Is this an UNBIND session
*                                      event?
         BE    ATTNLOSS                branch if yes to LOSS process
         EJECT
*********************************************************************
*                                                                   *
*        CNOS Processing Routine                                    *
*                                                                   *
* For ATTN exit driven for CNOS, the read RPL also provides a       *
* pointer to a read-only session limits data structure (ISTSLCNS).  *
* The session limits structure contains the negotiated session      *
* limits between the two LUs.                                       *
*********************************************************************
         SPACE 3
ATTNCNOS EQU   *
         WTO   'ATTN EXIT DRIVEN FOR CNOS          ',ROUTCDE=(1)
         SPACE 3
         L     R7,RPLAREA              Establish addressability to
         USING ISTSLCNS,R7                CNOS structure
         CLC   SLCSESSL,=H'0'          Is the a Reset Session limits?
         BE    RESET                   branch if yes
         MVI   LUSTATE,LUCNOSD         Else this request is setting
*                                      limits which this sample
*                                      program will just make note
*                                      session limits have been CNOSed
         B     ATTNRETN                Branch to return processing
RESET    EQU   *
         MVI   LUSTATE,LUNCNOS         Make note that session limits
*                                      have been reset and that no
*                                      limits are established.
         B     ATTNRETN                Branch to return processing
         DROP  R7                      remove basing to CNOS
*                                      structure
         EJECT
```

```
*********************************************************************
*                                                                   *
*        FMH-5 Processing Routine                                   *
*                                                                   *
*   VTAM schedules the ATTN exit with an FMH-5 event that indicates *
*   the partner has allocated a conversation with this LU.          *
*                                                                   *
*   The main propose of this process is to find an open conversation *
*   entry and issue a RCVFMH5 macroinstruction to receive the FMH-5 *
*   and complete the conversation's initialization on this side.    *
*   This macroinstruction will be issued asynchronously with an     *
*   exit to be driven when the request has completed.               *
*   This sample will assume the FMH-5 can be contained within the   *
*   the area supplied in the conversation entry.                    *
*                                                                   *
*   This sample will also utilize the USERFLD of the RPL6 for       *
*   passing the address of the conversation entry to the RPL exit   *
*   specified on the request.                                       *
*                                                                   *
*                                                                   *
*********************************************************************
         SPACE 3
ATTNFMH5 EQU   *
         WTO   'ATTN EXIT DRIVEN FOR FMH5 RECEIVED ',ROUTCDE=(1)
```

```
         SPACE 3
         LA    R6,2(0,0)               R6 assigned to 2 (#entries)
         L     R10,=A(CONVTBL)         R10 address of conversation
*                                      table (1st entry)
         USING CONVTBLD,R10            Establish addressability
*                                      of conversation entry
TESTOPEN EQU   *
         CLI   CONVSTAT,CONVOPEN       Entry available ?
         BE    RCVFMH5                 Branch yes
         LA    R10,CONVLEN(0,R10)      Address next entry
         BCT   R6,TESTOPEN             Branch to TESTOPEN if we
*                                      haven't reached end of table.
         EX    0,*                     Definition error or partner
*                                      LU. Cannot accept this
*                                      conversation request.
         SPACE 3
RCVFMH5  EQU   *
         MVI   CONVSTAT,CONVACT        Set conversation as active
         L     R1,=A(MODLRPL)          Initialize
         MVC   CONVRPL,0(R1)              RPL
         L     R1,=A(MODLRPL6)         Initialize
         MVC   CONVRPL6,0(R1)             RPL6
         SPACE 3
         WTO   'ISSUING CONTROL=RCVFMH5 MACRO REQUEST             ',ROUX
               TCDE=(1)
         SPACE 3
         APPCCMD RPL=CONVRPL,AAREA=CONVRPL6,                          X
               CONTROL=RCVFMH5,AREA=CONVAREA,AREALEN=L'CONVAREA,      X
               OPTCD=ASY,EXIT=R5EXT,USERFLD=(R10)
         B     ATTNRETN                Branch to EXIT return
         EJECT
```

```
*********************************************************************
*                                                                   *
*        LOSS Processing Routine                                    *
*                                                                   *
*********************************************************************
         SPACE 3
ATTNLOSS EQU   *
         WTO   'ATTN EXIT DRIVEN FOR LOSS SESSION  ',ROUTCDE=(1)
         B     ATTNRETN
         EJECT
*********************************************************************
*                                                                   *
*        ATTN return processing                                     *
*                                                                   *
*********************************************************************
         SPACE 3
ATTNRETN EQU   *
         WTO   'EXITING ATTN ACB EXIT ROUTINE           ',ROUTCDE=(1)
         L     R13,4(0,R13)            Load address of first savearea
         LM    R14,R12,12(R13)         Load original registers
         BR    R14                     Branch back to VTAM
         SPACE 3
         DROP  R8,R9,R10,R12           remove basing for passed
*                                      structure
         LTORG
         EJECT
*********************************************************************
*                                                                   *
*        Partner LU information                                     *
*                                                                   *
*********************************************************************
         SPACE 3
LUENTRY  DS    0D
         AIF   ('&SIDE' EQ 'LEFT ').LPARTNR
LUNAME   DC    CL8'ACBVICKY'           Partner LU Name
         AGO   .EPARTNR
.LPARTNR ANOP
LUNAME   DC    CL8'ACBGARY '           Partner LU Name
.EPARTNR ANOP
LUSTATE  DC    XL1'00'
LUNCNOS  EQU   X'00'                   Session limits not established
LUCNOSD  EQU   X'80'                   Session limits established
         DC    XL7'00'
LOGMODE  DC    CL8'LU62CONV'           LOGMODE name
         EJECT
```

```
*********************************************************************
*                                                                   *
```

```
*        RCVFMH5 RPL Exit Routine                                      *
*                                                                      *
*        This routine is given control by VTAM when the RCVFMH5        *
*        macroinstruction has completed.  When the macroinstruction    *
*        completes successfully, VTAM will return the conversation     *
*        ID in the RPL extension.  Because the RPL extension is        *
*        dedicated to this conversation, further manipulation of       *
*        the conversation ID will not be required.  The ID will        *
*        automatically be set for the next macroinstruction issued     *
*        for this conversation.                                        *
*                                                                      *
*  ON ENTRY:                                                           *
*       R1 - address of the RPL used in the macroinstruction           *
*       R14 - Return address when processing is finished               *
*       R15 - Address of this RPL exit                                 *
*                                                                      *
*                                                                      *
************************************************************************
R5EXT    EQU   *
         LR    R12,R15                 Establish normal
         USING R5EXT,R12                    addressability for routine
         SPACE 1
         LR    R9,R1                   Establish base reg for RPL
         USING IFGRPL,R9                   RPL
         SPACE 3
         WTO   'ENTERING RCVFMH5 RPL EXIT ROUTINE       ',ROUTCDE=(1)
         SPACE 3
         L     R8,RPLAAREA             Load address of read only RPL6
         USING ISTRPL6X,R8             set basing
         L     R10,RPL6USR             Establish addressability to
         USING CONVTBLD,R10               conversation entry
         LA    R13,CONVSA              Use the conversation storage
         STM   R14,R12,12(R13)           to save the registers
         LA    R15,CONVCA              establish another save area
*                                        for subsequent calls or
*                                        macro requests.
         ST    R15,8(0,R13)            Normal save area
         ST    R13,4(0,R15)               Save area
         LR    R13,R15                       chaining
         SPACE 3
         APPCCMD RPL=CONVRPL,                                          X
               CONTROL=CHECK
         SPACE 3
```

```
************************************************************************
*                                                                      *
*  Verify macroinstruction completed successfully.  If it did not,     *
*  provide notification of its failure.                                *
*                                                                      *
************************************************************************
         CLI   RPLRTNCD,USFAOK         RTNCD = X'00' ?
         BNE   RCV5FL                  branch no
         CLI   RPLFDB2,USFAOOK         FDB2 = X'00' ?
         BE    DORCV                   Branch yes
RCV5FL   EQU   *
         WTO   'CONTROL=RCVFMH5 failed',ROUTCDE=(1)
         B     R5XTRETN
         SPACE 3
************************************************************************
*                                                                      *
*  At this point we know the conversation is in receive state and      *
*  therefore a RECEIVE macroinstruction will need to be issued.        *
*                                                                      *
************************************************************************
DORCV    EQU   *
         L     R12,=A(RREQ)            Branch to RECEIVE request
*                                       processing
         BR    R12                     Branch to routine
         SPACE 3
R5XTRETN EQU   *
         WTO   'EXITING RCVFMH5 RPL EXIT ROUTINE        ',ROUTCDE=(1)
         L     R13,4(0,R13)            Reload return
         LM    R14,R12,12(R13)           register
         BR    R14                     Branch back to VTAM
         DROP  R8,R9,R10,R12           Drop control block addressing
         EJECT
************************************************************************
*                                                                      *
*        ALLOC RPL Exit Routine                                        *
*                                                                      *
*        This routine is given control by VTAM when the ALLOC          *
```

```
*        macroinstruction has completed.  Because the RPL extension  *
*        is dedicated to this conversation, further manipulation of  *
*        the conversation ID will not be required.  The ID will      *
*        automatically be set for the next macroinstruction issued   *
*        for this conversation.                                      *
*                                                                    *
*  ON ENTRY:                                                         *
*       R1 - address of the RPL used in the macroinstruction         *
*       R14 - Return address when processing is finished             *
*       R15 - Address of this RPL exit                               *
*                                                                    *
*                                                                    *
**********************************************************************

AEXT     EQU   *
         LR    R12,R15                 Establish normal
         USING AEXT,R12                     addressability for routine
         SPACE 1
         LR    R9,R1                   Establish base reg for RPL
         USING IFGRPL,R9                   RPL
         SPACE 3
         WTO   'ENTERING ALLOC RPL EXIT ROUTINE         ',ROUTCDE=(1)
         SPACE 3
         L     R8,RPLAAREA             Load address of read only RPL6
         USING ISTRPL6X,R8             set basing
         L     R10,RPL6USR             Establish addressability to
         USING CONVTBLD,R10               conversation entry
         LA    R13,CONVSA              Use the conversation storage
         STM   R14,R12,12(R13)           to save the registers
         LA    R15,CONVCA              establish another save area
*                                       for subsequent calls or
*                                       macro requests.
         ST    R15,8(0,R13)            Normal save area
         ST    R13,4(0,R15)               Save area
         LR    R13,R15                       chaining
         SPACE 3
         APPCCMD RPL=CONVRPL,                                          X
               CONTROL=CHECK
         SPACE 3
**********************************************************************
*                                                                    *
*  Verify macroinstruction completed successfully.  If it did not,   *
*  provide notification of its failure.                              *
*                                                                    *
**********************************************************************
         CLI   RPLRTNCD,USFAOK         RTNCD = X'00' ?
         BNE   ALLCFL                  branch no
         CLI   RPLFDB2,USFAOOK         FDB2 = X'00' ?
         BE    DOSEND                  Branch yes
ALLCFL   EQU   *
         WTO   'CONTROL=ALLOC   failed',ROUTCDE=(1)
         B     AEXTRETN
         SPACE 3
**********************************************************************
*                                                                    *
*  At this point we know the conversation is in send state and       *
*  therefore a SEND macroinstruction will need to be issued.         *
*                                                                    *
**********************************************************************
         SPACE 3
DOSEND   EQU   *
         L     R12,=A(SREQ)            Load address of SEND request
*                                      routine
         BR    R12                     Branch to it
         SPACE 3
AEXTRETN EQU   *
         WTO   'EXITING ALLOC RPL EXIT ROUTINE          ',ROUTCDE=(1)
         L     R13,4(0,R13)
         LM    R14,R12,12(R13)
         BR    R14
         DROP  R8,R9,R10,R12           Drop control block addressing
         EJECT

**********************************************************************
*                                                                    *
*        SEND  Request Routine                                       *
*                                                                    *
*  When designing an LU 6.2 application, the user is free to utilize *
*  many features of the LU 6.2 protocol to facilitate a transaction. *
*  For our sample program we will                                    *
```

```
*  have three options which will be dictated by the console operator *
*  through a WTOR macroinstruction.  The operator can reply with     *
*  the following strings of data:                                    *
*                                                                    *
*       DEALLOCATE - which will cause this application to issue a    *
*                    DEALLOC CONFIRM macroinstruction                *
*       blanks     - which will cause this application to issue a    *
*                    PREPRCV CONFIRM macroinstruction.  This will    *
*                    eventually put the conversation into receive    *
*                    state.                                          *
*       any other string - which will cause this application to      *
*                          take that string and add a length field   *
*                          to the beginning and issue a SEND DATA    *
*                          macroinstruction                          *
*                                                                    *
*                                                                    *
*  ON ENTRY:                                                         *
*       R10 = Conversation entry                                     *
*       R12 = Address of this routine                                *
*       R13 = address of normal save area chain                      *
**********************************************************************
SREQ     EQU   *
         USING SREQ,R12               Establish basing for routine
         USING CONVTBLD,R10           Establish basing for
*                                     conversation entry
         WTO   'ENTERING SEND REQUEST ROUTINE           ',ROUTCDE=(1)
         SPACE 3
         MVC   CONVWTOR(MODLWTRL),MODLWTOR
         MVC   CONVWTOR+12(L'CONVNAME),CONVNAME
         MVC   CONVTXT,BLANKS          Initialize reply area
         XC    CONVWECB,CONVWECB       Initialize ECB
         WTOR  ,CONVTXT,L'CONVTXT,CONVWECB,MF=(E,CONVWTOR)
         WAIT  ECB=CONVWECB
         CLC   CONVTXT(L'DEALLOC),DEALLOC
         BE    DODALLOC
         CLC   CONVTXT,BLANKS          Blanks specified?
         BE    SNDCFM
         MVC   CONVLL,=AL2(L'CONVAREA)
         SPACE 3
         WTO   'ISSUING CONTROL=SEND QUALIFY=DATA  MACRO REQUEST  ',ROUX
               TCDE=(1)
         SPACE 3
         APPCCMD RPL=CONVRPL,AAREA=CONVRPL6,                          X
               CONTROL=SEND,QUALIFY=DATA,                             X
               AREA=CONVAREA,RECLEN=L'CONVAREA,                       X
               OPTCD=ASY,EXIT=SEXT
         B     SREQRETN
         SPACE 3
```

```
SNDCFM   EQU   *
         WTO   'ISSUING CONTROL=SEND QUALIFY=CONFIRM MACRO REQUEST',ROUX
               TCDE=(1)
         APPCCMD RPL=CONVRPL,AAREA=CONVRPL6,                          X
               CONTROL=PREPRCV,QUALIFY=CONFIRM,                       X
               OPTCD=ASY,EXIT=SEXT
         B     SREQRETN
         SPACE 3
DODALLOC EQU   *
         WTO   'ISSUING CONTROL=DEALLOC QUALIFY=CONFIRM MACRO REQUEST',X
               ROUTCDE=(1)
         APPCCMD RPL=CONVRPL,AAREA=CONVRPL6,                          X
               CONTROL=DEALLOC,QUALIFY=CONFIRM,                       X
               OPTCD=ASY,EXIT=SEXT
SREQRETN EQU   *
         WTO   'EXITING  SEND REQUEST ROUTINE           ',ROUTCDE=(1)
         L     R13,4(0,R13)
         LM    R14,R12,12(R13)
         BR    R14
         DROP  R10,R12                Drop control block addressing
DEALLOC  DC    CL10'DEALLOCATE'
BLANKS   DC    CL(CONVALEN)' '
MODLWTOR WTOR  'convname - ENTER message, DEALLOCATE, or blanks to receX
               ive    ',,L'CONVTXT,,MF=L
MODLWTRL EQU   *-MODLWTOR
         EJECT
```

```
**********************************************************************
*                                                                    *
*        RECEIVE Request Routine                                     *
*                                                                    *
```

```
* This sample program has chosen to receive data in logical records. *
* The FILL=LL is specified to inform VTAM to complete this APPCCMD   *
* CONTROL=RECEIVE macroinstruction one logical record at a time.     *
* The QUALIFY=SPEC operand indicates this macroinstruction applies   *
* to a single conversation that is identified in the CONVID field in *
* the RPL extension.  Remember this field was set by VTAM on the     *
* completion of the macro instruction used to establish this         *
* conversation.  This sample program is written not to disturb the   *
* field once VTAM sets it.                                           *
*                                                                    *
* The AREA operand references storage in the conversation entry      *
* where VTAM can place the logical record.                           *
*                                                                    *
* The APPCCMD CONTROL=RECEIVE will be issued asynchronously with an  *
* exit to be driven by VTAM when the receive operation is complete.  *
* The exit address is supplied in the EXIT= operand.                 *
*                                                                    *
*  ON ENTRY:                                                         *
*        R10 = Conversation entry                                    *
*        R12 = Address of this routine                               *
*        R13 = address of normal save area chain                     *
**********************************************************************
RREQ     EQU   *
         USING RREQ,R12
         WTO   'ENTERING RECEIVE REQUEST ROUTINE        ',ROUTCDE=(1)
         USING CONVTBLD,R10
         WTO   'ISSUING CONTROL=RECEIVE QUALIFY=SPEC MACRO REQUEST   ',X
               ROUTCDE=(1)
         MVI   CONVAREA,C' '
         MVC   CONVAREA+1(L'CONVAREA-1),CONVAREA
         APPCCMD RPL=CONVRPL,AAREA=CONVRPL6,                          X
               CONTROL=RECEIVE,QUALIFY=SPEC,FILL=LL,                  X
               AREA=CONVAREA,AREALEN=L'CONVAREA,                      X
               EXIT=REXT
RREQRETN EQU   *
         WTO   'EXITING RECEIVE REQUEST ROUTINE         ',ROUTCDE=(1)
         L     R13,4(0,R13)
         LM    R14,R12,12(R13)
         BR    R14
         DROP  R10,R12
         EJECT
```

```
**********************************************************************
*                                                                    *
*        RECEIVE RPL Exit Routine                                    *
*                                                                    *
* VTAM drives this RPL exit routine when the operation for the       *
* APPCCMD CONTROL=RECEIVE macroinstruction is complete.  Based on    *
* how elaborate or simple the user chooses to make his transaction   *
* dictates what the application needs to do next.  This sample       *
* will insure the receive is successful and then assume it is        *
* completed only with DATA-COMPLETE or CONFIRM or both.              *
* The process of DATA-COMPLETE will be to display the data received  *
* to the operator.  If CONFIRM is specified, then an APPCCMD         *
* CONTROL=SEND,QUALIFY=CONFRMD macroinstruction is issued to         *
* acknowledge the data was successfully processed.                   *
* If CONFIRM is not received, then this routine will branch to the   *
* RECEIVE request routine to build an APPCCMD CONTROL=RECEIVE        *
* macroinstruction.                                                  *
*                                                                    *
*  ON ENTRY:                                                         *
*       R1 - address of the RPL used in the macroinstruction         *
*       R14 - Return address when processing is finished             *
*       R15 - Address of this RPL exit                               *
*                                                                    *
*                                                                    *
**********************************************************************
REXT     EQU   *
         LR    R12,R15                 Establish normal
         USING REXT,R12                     addressability for routine
         SPACE 1
         LR    R9,R1                   Establish base reg for RPL
         USING IFGRPL,R9                   RPL
         SPACE 3
         WTO   'ENTERING RECEIVE RPL EXIT ROUTINE       ',ROUTCDE=(1)
         SPACE 3
         L     R8,RPLAAREA             Load address of read only RPL6
         USING ISTRPL6X,R8             set basing
         L     R10,RPL6USR             Establish addressability to
         USING CONVTBLD,R10               conversation entry
         LA    R13,CONVSA              Use the conversation storage
```

```
         STM   R14,R12,12(R13)           to save the registers
         LA    R15,CONVCA              establish another save area
*                                      for subsequent calls or
*                                      macro requests.
         ST    R15,8(0,R13)            Normal save area
         ST    R13,4(0,R15)               Save area
         LR    R13,R15                       chaining
         SPACE 3
         APPCCMD RPL=CONVRPL,                                          X
               CONTROL=CHECK
         SPACE 3
```

```
*********************************************************************
*                                                                   *
*  Verify macroinstruction completed successfully.  If it did not,  *
*  provide notification of its failure.                             *
*                                                                   *
*********************************************************************
         CLI   RPLRTNCD,USFAOK         RTNCD = X'00' ?
         BNE   RCVFL                   branch no
         CLI   RPLFDB2,USFAOOK         FDB2 = X'00' ?
         BE    WHATRCV                 Branch yes
RCVFL    EQU   *
         WTO   'CONTROL=RECEIVE FAILED',ROUTCDE=(1)
         B     REXTRETN
         SPACE 3
*********************************************************************
*                                                                   *
*  If data has been received, then display the data to the console  *
*  operator via WTO macroinstruction.                               *
*                                                                   *
*********************************************************************
         SPACE 3
WHATRCV  EQU   *
         TM    RPL6RCV1,RPL6WDAC       DATA-COMPLETE
         BZ    TSTCONF
         MVC   CONVWTO(MODLWTOL),MODLWTO
         MVC   CONVWTO+4(L'CONVNAME),CONVNAME
         MVC   CONVWTO+15(L'CONVTXT),CONVTXT
         WTO   ,,MF=(E,CONVWTO)
         SPACE 3
*********************************************************************
*                                                                   *
*  If the CONFIRM indicator has been set, then issue an APPCCMD     *
*  CONTROL=SEND,QUALIFY=CONFRMD macroinstruction to acknowledge     *
*  the data has been processed.                                     *
*********************************************************************
         SPACE 3
TSTCONF  EQU   *
         TM    RPL6RCV1,RPL6WCFM       CONFIRM
         BZ    RCVTST
         WTO   'ISSUING CONTROL=SEND QUALIFY=CONFRMD MACRO REQUEST   ',X
               ROUTCDE=(1)
         APPCCMD RPL=CONVRPL,AAREA=CONVRPL6,                           X
               CONTROL=SEND,QUALIFY=CONFRMD,                           X
               EXIT=SEXT
         SPACE 3
*********************************************************************
*                                                                   *
*  If the conversation state is in receive state, then branch to the *
*  RECEIVE request issuing routine.  Else assume the conversation   *
*  has ended.                                                       *
*********************************************************************
```

```
         SPACE 3
RCVTST   EQU   *
         CLI   RPL6CCST,RPL6RECV
         BNE   REXTRETN
         L     R12,=A(RREQ)
         BR    R12
         SPACE 3
REXTRETN EQU   *
         WTO   'EXITING RECEIVE RPL EXIT ROUTINE        ',ROUTCDE=(1)
         L     R13,4(0,R13)
         LM    R14,R12,12(R13)
         BR    R14
         DROP  R8,R9,R10,R12
MODLWTO  WTO   'convname - 12345678901234567890123456789012345678901234X
               567890 ',ROUTCDE=(1),MF=L
```

```
MODLWTOL EQU   *-MODLWTO
         EJECT
```

```
*********************************************************************
*                                                                   *
*        SEND RPL Exit Routine                                      *
*                                                                   *
* VTAM drives this RPL exit routine when the operation for the      *
* APPCCMD CONTROL=SEND, CONTROL=PREPRCV, or CONTROL=DEALLOC         *
* macroinstruction has completed.                                   *
* This exit will use the conversation state to dictate what needs to *
* be done next.  This sample will assume only three conditions can  *
* exist at the completion of the macroinstruction that caused this  *
* exit to be driven.  The conversation can be in SEND, RECEIVE, or  *
* DEALLOCATED states.                                               *
*                                                                   *
*  ON ENTRY:                                                        *
*       R1 - address of the RPL used in the macroinstruction        *
*       R14 - Return address when processing is finished            *
*       R15 - Address of this RPL exit                              *
*                                                                   *
*                                                                   *
*********************************************************************
         SPACE 3
SEXT     EQU   *
         LR    R12,R15                 Establish normal
         USING SEXT,R12                     addressability for routine
         SPACE 1
         LR    R9,R1                   Establish base reg for RPL
         USING IFGRPL,R9                   RPL
         SPACE 3
         WTO   'ENTERING SEND RPL EXIT ROUTINE          ',ROUTCDE=(1)
         SPACE 3
         L     R8,RPLAAREA             Load address of read only RPL6
         USING ISTRPL6X,R8             set basing
         L     R10,RPL6USR             Establish addressability to
         USING CONVTBLD,R10               conversation entry
         LA    R13,CONVSA              Use the conversation storage
         STM   R14,R12,12(R13)           to save the registers
         LA    R15,CONVCA              establish another save area
*                                       for subsequent calls or
*                                       macro requests.
         ST    R15,8(0,R13)            Normal save area
         ST    R13,4(0,R15)               Save area
         LR    R13,R15                       chaining
         SPACE 3
         APPCCMD RPL=CONVRPL,                                          X
               CONTROL=CHECK
         SPACE 3
```

```
*********************************************************************
*                                                                   *
*  Verify macroinstruction completed successfully.  If it did not,  *
*  provide notification of its failure.                             *
*                                                                   *
*********************************************************************
         CLI   RPLRTNCD,USFAOK         RTNCD = X'00' ?
         BNE   SENDFL                  branch no
         CLI   RPLFDB2,USFAOOK         FDB2 = X'00' ?
         BE    SENDTST                 Branch yes
SENDFL   EQU   *
         WTO   'APPCCMD request FAILED',ROUTCDE=(1)
         B     SEXTRETN
         SPACE 3
*********************************************************************
*                                                                   *
*  Determine the conversation state and branch to the appropriate   *
*  process.                                                         *
*                                                                   *
*********************************************************************
         SPACE 3
SENDTST  EQU   *
         CLI   RPL6CCST,RPL6ENDC       Has the conversation ended?
         BE    RESETCNV                branch yes to conversation
*                                      end process.
         CLI   RPL6CCST,RPL6SND        Are we in send state ?
         BNE   GO2RCV                  branch no - assume receive
*                                      state.
         L     R12,=A(SREQ)            Load address of SEND request
*                                      routine.
```

```
         BR    R12                     branch to routine
GO2RCV   EQU   *
         L     R12,=A(RREQ)            Load address of RECEIVE request
*                                      routine
         BR    R12                     branch to routine
         SPACE 3
*********************************************************************
*   Because the conversation has ended, we will reset               *
*   the conversation entry as open, indicating entry available      *
*   for the next conversation.                                      *
*********************************************************************
         SPACE 1
RESETCNV EQU   *
         MVI   CONVSTAT,CONVOPEN       Mark entry as available
         SPACE 3
SEXTRETN EQU   *
         WTO   'EXITING SEND RPL EXIT ROUTINE           ',ROUTCDE=(1)
         L     R13,4(0,R13)            reload
         LM    R14,R12,12(R13)           registers
         BR    R14                     branch to VTAM
         DROP  R8,R9,R10,R12
         LTORG
         EJECT
```

```
*********************************************************************
*                                                                   *
*        Conversation table                                         *
*                                                                   *
*********************************************************************
CONVTBL  DS    0D
         AIF   ('&SIDE' EQ 'LEFT ').LCONV
JAIME    DC    (CONVLEN)XL1'00'
         ORG   JAIME
         DC    CL8'JAIME   '
         ORG
TIMOTHY  DC    (CONVLEN)XL1'00'
         ORG   TIMOTHY
         DC    CL8'TIMOTHY '
         ORG
         AGO   .ECONV
.LCONV   ANOP
JEFFERY  DC    (CONVLEN)XL1'00'
         ORG   JEFFERY
         DC    CL8'JEFFERY '
         ORG
KIMBERLY DC    (CONVLEN)XL1'00'
         ORG   KIMBERLY
         DC    CL8'KIMBERLY'
         ORG
.ECONV   ANOP
         EJECT
```

```
*********************************************************************
*                                                                   *
*        Conversation Entry Layout                                  *
*                                                                   *
*********************************************************************
CONVTBLD DSECT
CONVNAME DS    CL8                     Transaction program, name of me
CONVSTAT DS    XL1
CONVOPEN EQU   X'00'                   Conversation entry open
CONVACT  EQU   X'80'                   Conversation active
         DS    XL7                     Unused
CONVSA   DS    18F                     Save area for RPL exits
CONVCA   DS    18F                     Save area for CALL routines
CONVRPL  DS    XL112                   RPL
CONVRPL6 DS    XL112                   RPL6
CONVWECB DS    F
CONVAREA DS    0XL52                   AREA
CONVLL   DS    XL2
CONVTXT  DS    XL50
CONVALEN EQU   L'CONVAREA
         DS    0F
CONVWTO  WTO   '                                                      X
                ',ROUTCDE=(1),MF=L
CONVWTOR WTOR  '                                                      X
                ',CONVTXT,L'CONVTXT,CONVWECB,MF=L
         DS    0D
CONVLEN  EQU   *-CONVTBLD
         EJECT
```

```
          IFGRPL AM=VTAM
          EJECT
          ISTFM5
          EJECT
          ISTSLCNS
SLCLEN    EQU    SLCEND-ISTSLCNS
          ISTUSFBC
          IFGACB AM=VTAM
          END
```

Console log

```
JOB   2  IEF403I VTAM    - STARTED - TIME = 11.54.19
v net,act,id=appcappl
JOB   2  IST097I NOCVA VARY ACCEPTED
JOB   2  IST093I ACCEA APPCAPPL ACTIVE, NODE TYPE = APPL SEGMENT
s appcappl
JOB   6  IEF403I APPCAPPL - STARTED - TIME = 12.49.41
JOB   6  ISSUING OPEN ACB MACRO REQUEST
JOB   6  ISSUING SETLOGON MACRO REQUEST
JOB   6 *02 Enter START to start a transaction or CLOSE to close the ACB
                                  s appcappr
                                  JOB   7  IEF403I APPCAPPR - STARTED - TIME = 12.49.47
                                  JOB   7  ISSUING OPEN ACB MACRO REQUEST
                                  JOB   7  ISSUING SETLOGON MACRO REQUEST
                                  JOB   7 *03 Enter START to start a transaction or CLOSE to close the ACB
r 02,START
   IEE600I REPLY TO 02 IS: START
JOB   6  ISSUING CONTROL=OPRCNTL,QUALIFY=CNOS MACRO REQUEST
                                  JOB   7  ENTERING ATTN ACB EXIT ROUTINE
                                  JOB   7  ATTN EXIT DRIVEN FOR CNOS
                                  JOB   7  EXITING ATTN ACB EXIT ROUTINE
JOB   6  ISSUING CONTROL=ALLOC,QUALIFY=ALLOCD MACRO REQUEST
JOB   6 *04 Enter START to start a transaction or CLOSE to close the ACB
JOB   6  ENTERING ALLOC RPL EXIT ROUTINE
JOB   6  ENTERING SEND REQUEST ROUTINE
JOB   6 *05 JEFFERY  - ENTER message, DEALLOCATE, or blanks to receive
r 05,Hello Jaime this is Jeffery
   IEE600I REPLY TO 05 IS: HELLO JAIME THIS IS JEFFERY
JOB   6  ISSUING CONTROL=SEND QUALIFY=DATA  MACRO REQUEST
JOB   6  EXITING  SEND REQUEST ROUTINE
JOB   6  ENTERING SEND RPL EXIT ROUTINE
JOB   6  ENTERING SEND REQUEST ROUTINE
JOB   6 *06 JEFFERY  - ENTER message, DEALLOCATE, or blanks to receive
r 06,
   IEE600I REPLY TO 06 IS:
JOB   6  ISSUING CONTROL=SEND QUALIFY=CONFIRM MACRO REQUEST
JOB   6  EXITING  SEND REQUEST ROUTINE
                                  JOB   7  ENTERING ATTN ACB EXIT ROUTINE
                                  JOB   7  ATTN EXIT DRIVEN FOR FMH-5 RECEIVED
                                  JOB   7  ISSUING CONTROL=RCVFMH5 MACRO REQUEST
                                  JOB   7  EXITING ATTN ACB EXIT ROUTINE
                                  JOB   7  ENTERING RCVFMH5 RPL EXIT ROUTINE
                                  JOB   7  ENTERING RECEIVE REQUEST ROUTINE
                                  JOB   7  ISSUING CONTROL=RECEIVE QUALIFY=SPEC MACRO REQUEST
                                  JOB   7  EXITING RECEIVE REQUEST ROUTINE
                                  JOB   7  ENTERING RECEIVE RPL EXIT ROUTINE
                                  JOB   7  JAIME   - HELLO JAIME THIS IS JEFFERY
                                  JOB   7  ISSUING CONTROL=SEND QUALIFY=CONFRMD MACRO REQUEST
JOB   6  ENTERING SEND RPL EXIT ROUTINE
JOB   6  ENTERING RECEIVE REQUEST ROUTINE
JOB   6  ISSUING CONTROL=RECEIVE QUALIFY=SPEC MACRO REQUEST
JOB   6  EXITING RECEIVE REQUEST ROUTINE
                                  JOB   7  EXITING RECEIVE RPL EXIT ROUTINE
                                  JOB   7  ENTERING SEND RPL EXIT ROUTINE
                                  JOB   7  ENTERING SEND REQUEST ROUTINE
                                  JOB   7 *07 JAIME   - ENTER message, DEALLOCATE, or blanks to receive
                                  r 07,Hi Jeff.
                                  IEE600I REPLY TO 07 IS: HI JEFF.
                                  JOB   7  ISSUING CONTROL=SEND QUALIFY=DATA  MACRO REQUEST
                                  JOB   7  EXITING  SEND REQUEST ROUTINE
                                  JOB   7  ENTERING SEND RPL EXIT ROUTINE
                                  JOB   7  ENTERING SEND REQUEST ROUTINE
                                  JOB   7 *08 JAIME   - ENTER message, DEALLOCATE, or blanks to receive
                                  r 8,
                                  IEE600I REPLY TO 08 IS:
                                  JOB   7  ISSUING CONTROL=SEND QUALIFY=CONFIRM MACRO REQUEST
```

Figure 37. Console log part 1 of 7

```
JOB   6  ENTERING RECEIVE RPL EXIT ROUTINE
JOB   6  JEFFERY  - HI JEFF.
JOB   6  ISSUING CONTROL=SEND QUALIFY=CONFRMD MACRO REQUEST
JOB   6  EXITING RECEIVE RPL EXIT ROUTINE
JOB   6  ENTERING SEND RPL EXIT ROUTINE
JOB   6  ENTERING SEND REQUEST ROUTINE
JOB   6 *09 JEFFERY  - ENTER message, DEALLOCATE, or blanks to receive
                                  JOB   7  EXITING  SEND REQUEST ROUTINE
                                  JOB   7  ENTERING SEND RPL EXIT ROUTINE
                                  JOB   7  ENTERING RECEIVE REQUEST ROUTINE
                                  JOB   7  ISSUING CONTROL=RECEIVE QUALIFY=SPEC MACRO REQUEST
                                  JOB   7  EXITING RECEIVE REQUEST ROUTINE
r 09,We won the soccer game today!!
   IEE600I REPLY TO 09 IS: WE WON THE SOCCER GAME TODAY!!
JOB   6  ISSUING CONTROL=SEND QUALIFY=DATA  MACRO REQUEST
JOB   6  EXITING  SEND REQUEST ROUTINE
JOB   6  ENTERING SEND RPL EXIT ROUTINE
JOB   6  ENTERING SEND REQUEST ROUTINE
JOB   6 *10 JEFFERY  - ENTER message, DEALLOCATE, or blanks to receive
r 10,
   IEE600I REPLY TO 10 IS:
JOB   6  ISSUING CONTROL=SEND QUALIFY=CONFIRM MACRO REQUEST
JOB   6  EXITING  SEND REQUEST ROUTINE
                                  JOB   7  ENTERING RECEIVE RPL EXIT ROUTINE
                                  JOB   7  JAIME    - WE WON THE SOCCER GAME TODAY!!
                                  JOB   7  ISSUING CONTROL=SEND QUALIFY=CONFRMD MACRO REQUEST
JOB   6  ENTERING SEND RPL EXIT ROUTINE
JOB   6  ENTERING RECEIVE REQUEST ROUTINE
JOB   6  ISSUING CONTROL=RECEIVE QUALIFY=SPEC MACRO REQUEST
JOB   6  EXITING RECEIVE REQUEST ROUTINE
                                  JOB   7  EXITING RECEIVE RPL EXIT ROUTINE
                                  JOB   7  ENTERING SEND RPL EXIT ROUTINE
                                  JOB   7  ENTERING SEND REQUEST ROUTINE
                                  JOB   7 *11 JAIME    - ENTER message, DEALLOCATE, or blanks to receive
                                  r 11,Thats great!!
                                  IEE600I REPLY TO 11 IS: THATS GREAT!!
                                  JOB   7  ISSUING CONTROL=SEND QUALIFY=DATA  MACRO REQUEST
                                  JOB   7  EXITING  SEND REQUEST ROUTINE
                                  JOB   7  ENTERING SEND RPL EXIT ROUTINE
                                  JOB   7  ENTERING SEND REQUEST ROUTINE
                                  JOB   7 *12 JAIME    - ENTER message, DEALLOCATE, or blanks to receive
                                  r 12,
                                  IEE600I REPLY TO 12 IS:
                                  JOB   7  ISSUING CONTROL=SEND QUALIFY=CONFIRM MACRO REQUEST
JOB   6  ENTERING RECEIVE RPL EXIT ROUTINE
JOB   6  JEFFERY  - THATS GREAT!!
JOB   6  ISSUING CONTROL=SEND QUALIFY=CONFRMD MACRO REQUEST
JOB   6  EXITING RECEIVE RPL EXIT ROUTINE
JOB   6  ENTERING SEND RPL EXIT ROUTINE
JOB   6  ENTERING SEND REQUEST ROUTINE
JOB   6 *13 JEFFERY  - ENTER message, DEALLOCATE, or blanks to receive
                                  JOB   7  EXITING  SEND REQUEST ROUTINE
                                  JOB   7  ENTERING SEND RPL EXIT ROUTINE
                                  JOB   7  ENTERING RECEIVE REQUEST ROUTINE
                                  JOB   7  ISSUING CONTROL=RECEIVE QUALIFY=SPEC MACRO REQUEST
                                  JOB   7  EXITING RECEIVE REQUEST ROUTINE
```

Figure 38. Console log part 2 of 7

```
r 13,However Kimberly lost her basketball game
   IEE600I REPLY TO 13 IS: HOWEVER KIMBERLY LOST HER BASKETBALL GAME
JOB   6  ISSUING CONTROL=SEND QUALIFY=DATA  MACRO REQUEST
JOB   6  EXITING  SEND REQUEST ROUTINE
JOB   6  ENTERING SEND RPL EXIT ROUTINE
JOB   6  ENTERING SEND REQUEST ROUTINE
JOB   6 *14 JEFFERY  - ENTER message, DEALLOCATE, or blanks to receive
r 14,
   IEE600I REPLY TO 14 IS:
JOB   6  ISSUING CONTROL=SEND QUALIFY=CONFIRM MACRO REQUEST
JOB   6  EXITING  SEND REQUEST ROUTINE
                                  JOB   7  ENTERING RECEIVE RPL EXIT ROUTINE
                                  JOB   7  JAIME   - HOWEVER KIMBERLY LOST HER BASKETBALL GAME
                                  JOB   7  ISSUING CONTROL=SEND QUALIFY=CONFRMD MACRO REQUEST
JOB   6  ENTERING SEND RPL EXIT ROUTINE
JOB   6  ENTERING RECEIVE REQUEST ROUTINE
JOB   6  ISSUING CONTROL=RECEIVE QUALIFY=SPEC MACRO REQUEST
JOB   6  EXITING RECEIVE REQUEST ROUTINE
                                  JOB   7  EXITING RECEIVE RPL EXIT ROUTINE
                                  JOB   7  ENTERING SEND RPL EXIT ROUTINE
                                  JOB   7  ENTERING SEND REQUEST ROUTINE
                                  JOB   7 *15 JAIME   - ENTER message, DEALLOCATE, or blanks to receive
                                  r 15,I'll have Timothy call her on the second line
                                  IEE600I REPLY TO 15 IS: I'LL HAVE TIMOTHY CALL HER ON THE SECOND LI
                                  JOB   7  ISSUING CONTROL=SEND QUALIFY=DATA  MACRO REQUEST
                                  JOB   7  EXITING  SEND REQUEST ROUTINE
                                  JOB   7  ENTERING SEND RPL EXIT ROUTINE
                                  JOB   7  ENTERING SEND REQUEST ROUTINE
                                  JOB   7 *16 JAIME   - ENTER message, DEALLOCATE, or blanks to receive
                                  r 16,to cheer her up
                                  IEE600I REPLY TO 16 IS: TO CHEER HER UP
                                  JOB   7  ISSUING CONTROL=SEND QUALIFY=DATA  MACRO REQUEST
                                  JOB   7  EXITING  SEND REQUEST ROUTINE
                                  JOB   7  ENTERING SEND RPL EXIT ROUTINE
                                  JOB   7  ENTERING SEND REQUEST ROUTINE
                                  JOB   7 *17 JAIME   - ENTER message, DEALLOCATE, or blanks to receive
                                  r 3,START
                                  IEE600I REPLY TO 03 IS: START
                                  r 17,
                                  IEE600I REPLY TO 17 IS:
                                 JOB   7  ISSUING CONTROL=SEND QUALIFY=CONFIRM MACRO REQUEST
JOB   6  ENTERING RECEIVE RPL EXIT ROUTINE
JOB   6  JEFFERY  - I'LL HAVE TIMOTHY CALL HER ON THE SECOND LINE
JOB   6  ENTERING RECEIVE REQUEST ROUTINE
JOB   6  ISSUING CONTROL=RECEIVE QUALIFY=SPEC MACRO REQUEST
JOB   6  EXITING RECEIVE REQUEST ROUTINE
JOB   6  ENTERING RECEIVE RPL EXIT ROUTINE
JOB   6  JEFFERY  - TO CHEER HER UP
JOB   6  ISSUING CONTROL=SEND QUALIFY=CONFRMD MACRO REQUEST
JOB   6  EXITING RECEIVE RPL EXIT ROUTINE
JOB   6  ENTERING SEND RPL EXIT ROUTINE
JOB   6  ENTERING SEND REQUEST ROUTINE
JOB   6 *18 JEFFERY  - ENTER message, DEALLOCATE, or blanks to receive
```

Figure 39. Console log part 3 of 7

```
                                JOB    7  EXITING  SEND REQUEST ROUTINE
                                JOB    7  ENTERING SEND RPL EXIT ROUTINE
                                JOB    7  ENTERING RECEIVE REQUEST ROUTINE
                                JOB    7  ISSUING CONTROL=RECEIVE QUALIFY=SPEC MACRO REQUEST
                                JOB    7  EXITING RECEIVE REQUEST ROUTINE
                                JOB    7  ISSUING CONTROL=ALLOC,QUALIFY=ALLOCD MACRO REQUEST
                                JOB    7  ENTERING ALLOC RPL EXIT ROUTINE
                                JOB    7  ENTERING SEND REQUEST ROUTINE
                                JOB    7 *19 TIMOTHY  - ENTER message, DEALLOCATE, or blanks to receive
                                r 19,Hello Kimberly this is Timothy
                                IEE600I REPLY TO 19 IS: HELLO KIMBERLY THIS IS TIMOTHY
                                JOB    7  ISSUING CONTROL=SEND QUALIFY=DATA  MACRO REQUEST
                                JOB    7  EXITING  SEND REQUEST ROUTINE
                                JOB    7  ENTERING SEND RPL EXIT ROUTINE
                                JOB    7  ENTERING SEND REQUEST ROUTINE
                                JOB    7 *20 TIMOTHY  - ENTER message, DEALLOCATE, or blanks to receive
                                r 20,
                                IEE600I REPLY TO 20 IS:
                                JOB    7  ISSUING CONTROL=SEND QUALIFY=CONFIRM MACRO REQUEST
                                JOB    7  EXITING  SEND REQUEST ROUTINE
                                JOB    7 *21 Enter START to start a transaction or CLOSE to close the ACB
r 18,That would be a good idea.
   IEE600I REPLY TO 18 IS: THAT WOULD BE A GOOD IDEA.
JOB    6  ISSUING CONTROL=SEND QUALIFY=DATA  MACRO REQUEST
JOB    6  EXITING  SEND REQUEST ROUTINE
JOB    6  ENTERING ATTN ACB EXIT ROUTINE
JOB    6  ATTN EXIT DRIVEN FOR FMH-5 RECEIVED
JOB    6  ISSUING CONTROL=RCVFMH5 MACRO REQUEST
JOB    6  EXITING ATTN ACB EXIT ROUTINE
JOB    6  ENTERING SEND RPL EXIT ROUTINE
JOB    6  ENTERING SEND REQUEST ROUTINE
JOB    6 *22 JEFFERY  - ENTER message, DEALLOCATE, or blanks to receive
r 22,
   IEE600I REPLY TO 22 IS:
JOB    6  ISSUING CONTROL=SEND QUALIFY=CONFIRM MACRO REQUEST
JOB    6  EXITING  SEND REQUEST ROUTINE
JOB    6  ENTERING RCVFMH5 RPL EXIT ROUTINE
JOB    6  ENTERING RECEIVE REQUEST ROUTINE
JOB    6  ISSUING CONTROL=RECEIVE QUALIFY=SPEC MACRO REQUEST
JOB    6  EXITING RECEIVE REQUEST ROUTINE
JOB    6  ENTERING RECEIVE RPL EXIT ROUTINE
JOB    6  KIMBERLY - HELLO KIMBERLY THIS IS TIMOTHY
JOB    6  ISSUING CONTROL=SEND QUALIFY=CONFRMD MACRO REQUEST
JOB    6  EXITING RECEIVE RPL EXIT ROUTINE
JOB    6  ENTERING SEND RPL EXIT ROUTINE
JOB    6  ENTERING SEND REQUEST ROUTINE
JOB    6 *23 KIMBERLY - ENTER message, DEALLOCATE, or blanks to receive
                                JOB    7  ENTERING RECEIVE RPL EXIT ROUTINE
                                JOB    7  JAIME    - THAT WOULD BE A GOOD IDEA.
                                JOB    7  ISSUING CONTROL=SEND QUALIFY=CONFRMD MACRO REQUEST
                                JOB    7  EXITING RECEIVE RPL EXIT ROUTINE
                                JOB    7  ENTERING SEND RPL EXIT ROUTINE
                                JOB    7  ENTERING RECEIVE REQUEST ROUTINE
                                JOB    7  ISSUING CONTROL=RECEIVE QUALIFY=SPEC MACRO REQUEST
                                JOB    7  EXITING RECEIVE REQUEST ROUTINE
                                JOB    7  ENTERING SEND RPL EXIT ROUTINE
                                JOB    7  ENTERING SEND REQUEST ROUTINE
                                JOB    7 *24 JAIME    - ENTER message, DEALLOCATE, or blanks to receive
```

Figure 40. Console log part 4 of 7

```
r 23,Hi Timothy, we lost our basketball game..
   IEE600I REPLY TO 23 IS: HI TIMOTHY, WE LOST OUR BASKETBALL GAME..
JOB   6  ISSUING CONTROL=SEND QUALIFY=DATA  MACRO REQUEST
JOB   6  EXITING  SEND REQUEST ROUTINE
JOB   6  ENTERING SEND RPL EXIT ROUTINE
JOB   6  ENTERING RECEIVE REQUEST ROUTINE
JOB   6  ISSUING CONTROL=RECEIVE QUALIFY=SPEC MACRO REQUEST
JOB   6  EXITING RECEIVE REQUEST ROUTINE
JOB   6  ENTERING SEND RPL EXIT ROUTINE
JOB   6  ENTERING SEND REQUEST ROUTINE
JOB   6 *25 KIMBERLY - ENTER message, DEALLOCATE, or blanks to receive
                                  r 24,I've got to go now
                                  IEE600I REPLY TO 24 IS: I'VE GOT TO GO NOW
                                  JOB   7  ISSUING CONTROL=SEND QUALIFY=DATA  MACRO REQUEST
                                  JOB   7  EXITING  SEND REQUEST ROUTINE
                                  JOB   7  ENTERING SEND RPL EXIT ROUTINE
                                  JOB   7  ENTERING SEND REQUEST ROUTINE
                                  JOB   7 *26 JAIME    - ENTER message, DEALLOCATE, or blanks to receive
                                  r 26,good bye!!
                                  IEE600I REPLY TO 26 IS: GOOD BYE!!
                                  JOB   7  ISSUING CONTROL=SEND QUALIFY=DATA  MACRO REQUEST
                                  JOB   7  EXITING  SEND REQUEST ROUTINE
                                  JOB   7  ENTERING SEND RPL EXIT ROUTINE
                                  JOB   7  ENTERING SEND REQUEST ROUTINE
                                  JOB   7 *27 JAIME    - ENTER message, DEALLOCATE, or blanks to receive
                                  r 27,DEALLOCATE
                                  IEE600I REPLY TO 27 IS: DEALLOCATE
                                  JOB   7  ISSUING CONTROL=DEALLOC QUALIFY=CONFIRM MACRO REQUEST
                                  JOB   7  EXITING  SEND REQUEST ROUTINE
                                  r 25,
                                  IEE600I REPLY TO 25 IS:
JOB   6  ISSUING CONTROL=SEND QUALIFY=CONFIRM MACRO REQUEST
JOB   6  EXITING  SEND REQUEST ROUTINE
JOB   6  ENTERING RECEIVE RPL EXIT ROUTINE
JOB   6  JEFFERY  - I'VE GOT TO GO NOW
JOB   6  ENTERING RECEIVE REQUEST ROUTINE
JOB   6  ISSUING CONTROL=RECEIVE QUALIFY=SPEC MACRO REQUEST
JOB   6  EXITING RECEIVE REQUEST ROUTINE
JOB   6  ENTERING RECEIVE RPL EXIT ROUTINE
JOB   6  JEFFERY  - GOOD BYE!!
JOB   6  ISSUING CONTROL=SEND QUALIFY=CONFRMD MACRO REQUEST
JOB   6  EXITING RECEIVE RPL EXIT ROUTINE
JOB   6  ENTERING SEND RPL EXIT ROUTINE
JOB   6  EXITING SEND RPL EXIT ROUTINE
```

Figure 41. Console Log Part 5 of 7

```
                                JOB   7  ENTERING RECEIVE RPL EXIT ROUTINE
                                JOB   7  TIMOTHY  - HI TIMOTHY, WE LOST OUR BASKETBALL GAME..
                                JOB   7  ISSUING CONTROL=SEND QUALIFY=CONFRMD MACRO REQUEST
JOB   6  ENTERING SEND RPL EXIT ROUTINE
JOB   6  ENTERING RECEIVE REQUEST ROUTINE
JOB   6  ISSUING CONTROL=RECEIVE QUALIFY=SPEC MACRO REQUEST
JOB   6  EXITING RECEIVE REQUEST ROUTINE
                                JOB   7  EXITING RECEIVE RPL EXIT ROUTINE
                                JOB   7  ENTERING SEND RPL EXIT ROUTINE
                                JOB   7  EXITING SEND RPL EXIT ROUTINE
                                JOB   7  ENTERING SEND RPL EXIT ROUTINE
                                JOB   7  ENTERING SEND REQUEST ROUTINE
                                JOB   7 *28 TIMOTHY  - ENTER message, DEALLOCATE, or blanks to receive
                                r 28,Maybe Bruce can help your team.
                                IEE600I REPLY TO 28 IS: MAYBE BRUCE CAN HELP YOUR TEAM.
                                JOB   7  ISSUING CONTROL=SEND QUALIFY=DATA  MACRO REQUEST
                                JOB   7  EXITING  SEND REQUEST ROUTINE
                                JOB   7  ENTERING SEND RPL EXIT ROUTINE
                                JOB   7  ENTERING SEND REQUEST ROUTINE
                                JOB   7 *29 TIMOTHY  - ENTER message, DEALLOCATE, or blanks to receive
                                r 29,
                                IEE600I REPLY TO 29 IS:
                                JOB   7  ISSUING CONTROL=SEND QUALIFY=CONFIRM MACRO REQUEST
JOB   6  ENTERING RECEIVE RPL EXIT ROUTINE
JOB   6  KIMBERLY - MAYBE BRUCE CAN HELP YOUR TEAM.
JOB   6  ISSUING CONTROL=SEND QUALIFY=CONFRMD MACRO REQUEST
JOB   6  EXITING RECEIVE RPL EXIT ROUTINE
JOB   6  ENTERING SEND RPL EXIT ROUTINE
JOB   6  ENTERING SEND REQUEST ROUTINE
JOB   6 *30 KIMBERLY - ENTER message, DEALLOCATE, or blanks to receive
                                JOB   7  EXITING  SEND REQUEST ROUTINE
                                JOB   7  ENTERING SEND RPL EXIT ROUTINE
                                JOB   7  ENTERING RECEIVE REQUEST ROUTINE
                                JOB   7  ISSUING CONTROL=RECEIVE QUALIFY=SPEC MACRO REQUEST
                                JOB   7  EXITING RECEIVE REQUEST ROUTINE
r 30,Maybe..
  IEE600I REPLY TO 30 IS: MAYBE..
JOB   6  ISSUING CONTROL=SEND QUALIFY=DATA  MACRO REQUEST
JOB   6  EXITING  SEND REQUEST ROUTINE
JOB   6  ENTERING SEND RPL EXIT ROUTINE
JOB   6  ENTERING SEND REQUEST ROUTINE
JOB   6 *31 KIMBERLY - ENTER message, DEALLOCATE, or blanks to receive
r 31,thanks for calling
  IEE600I REPLY TO 31 IS: THANKS FOR CALLING
JOB   6  ISSUING CONTROL=SEND QUALIFY=DATA  MACRO REQUEST
JOB   6  EXITING  SEND REQUEST ROUTINE
JOB   6  ENTERING SEND RPL EXIT ROUTINE
JOB   6  ENTERING SEND REQUEST ROUTINE
JOB   6 *32 KIMBERLY - ENTER message, DEALLOCATE, or blanks to receive
r 32,good bye
  IEE600I REPLY TO 32 IS: GOOD BYE
JOB   6  ISSUING CONTROL=SEND QUALIFY=DATA  MACRO REQUEST
JOB   6  EXITING  SEND REQUEST ROUTINE
JOB   6  ENTERING SEND RPL EXIT ROUTINE
JOB   6  ENTERING SEND REQUEST ROUTINE
JOB   6 *33 KIMBERLY - ENTER message, DEALLOCATE, or blanks to receive
r 33,DEALLOCATE
  IEE600I REPLY TO 33 IS: DEALLOCATE
JOB   6  ISSUING CONTROL=DEALLOC QUALIFY=CONFIRM MACRO REQUEST
JOB   6  EXITING  SEND REQUEST ROUTINE
```

Figure 42. Console Log part 6 of 7

```
                                JOB   7  ENTERING RECEIVE RPL EXIT ROUTINE
                                JOB   7  TIMOTHY  - MAYBE..
                                JOB   7  ENTERING RECEIVE REQUEST ROUTINE
                                JOB   7  ISSUING CONTROL=RECEIVE QUALIFY=SPEC MACRO REQUEST
                                JOB   7  EXITING RECEIVE REQUEST ROUTINE
                                JOB   7  ENTERING RECEIVE RPL EXIT ROUTINE
                                JOB   7  TIMOTHY  - THANKS FOR CALLING
                                JOB   7  ENTERING RECEIVE REQUEST ROUTINE
                                JOB   7  ISSUING CONTROL=RECEIVE QUALIFY=SPEC MACRO REQUEST
                                JOB   7  EXITING RECEIVE REQUEST ROUTINE
                                JOB   7  ENTERING RECEIVE RPL EXIT ROUTINE
                                JOB   7  TIMOTHY  - GOOD BYE
                                JOB   7  ISSUING CONTROL=SEND QUALIFY=CONFRMD MACRO REQUEST
JOB   6  ENTERING SEND RPL EXIT ROUTINE
JOB   6  EXITING SEND RPL EXIT ROUTINE
                                JOB   7  EXITING RECEIVE RPL EXIT ROUTINE
                                JOB   7  ENTERING SEND RPL EXIT ROUTINE
                                JOB   7  EXITING SEND RPL EXIT ROUTINE
                                r 21,CLOSE
                                IEE600I REPLY TO 21 IS: CLOSE
                                JOB   7  RESETTING SESSION LIMITS FOR LU62CONV MODE
JOB   6  ENTERING ATTN ACB EXIT ROUTINE
JOB   6  ATTN EXIT DRIVEN FOR CNOS
JOB   6  EXITING ATTN ACB EXIT ROUTINE
JOB   6  ENTERING ATTN ACB EXIT ROUTINE
JOB   6  ATTN EXIT DRIVEN FOR LOSS SESSION
JOB   6  EXITING ATTN ACB EXIT ROUTINE
                                JOB   7  ENTERING ATTN ACB EXIT ROUTINE
                                JOB   7  ATTN EXIT DRIVEN FOR LOSS SESSION
                                JOB   7  EXITING ATTN ACB EXIT ROUTINE
                                JOB   7  RESETTING SESSION LIMITS FOR SNASVCMG MODE
JOB   6  ENTERING ATTN ACB EXIT ROUTINE
JOB   6  ATTN EXIT DRIVEN FOR LOSS SESSION
JOB   6  EXITING ATTN ACB EXIT ROUTINE
                                JOB   7  ENTERING ATTN ACB EXIT ROUTINE
                                JOB   7  ATTN EXIT DRIVEN FOR LOSS SESSION
                                JOB   7  EXITING ATTN ACB EXIT ROUTINE
                                JOB   7  ISSUING CLOSE ACB MACRO REQUEST
                                JOB   7  IEF142I APPCAPPR APPCAPPR APPCAPPR STEP WAS EXECUTED-
                                         COND CODE 0000
                                JOB   7  IEF404I APPCAPPR - ENDED - TIME = 13.02.48
r 4,CLOSE

  IEE600I REPLY TO 4 IS: CLOSE
JOB   6  ISSUING CLOSE ACB MACRO REQUEST
JOB   6  IEF142I APPCAPPL APPCAPPL APPCAPPL STEP WAS EXECUTED - COND CODE 0000
JOB   6  IEF404I APPCAPPL - ENDED - TIME = 13.03.15
```

Figure 43. Console log part 7 of 7

Appendix D. Example of retrieving information for a mode and any restored sessions

The example in this appendix shows how to examine or process information returned by the APPCCMD CONTROL=OPRCNTL, QUALIFY=RESTORE macroinstruction. VTAM returns this information to the user in the data area provided for it.

The user can retrieve LU-mode table information (NOSESS), LU-mode table and session information (ALL), or no information (NONE). The application program specifies the type of information returned using the LIST keyword in the APPCCMD CONTROL=OPRCNTL, QUALIFY=RESTORE macroinstruction. Use of the LU name and logon mode parameters also affects the modes restored. For more information on restoring a mode, see “Restoring modes and any associated persistent LU-LU sessions” on page 49. For more information about the RESTORE APPCCMD, refer to z/OS Communications Server: SNA Programmer's LU 6.2 Reference. See “Retrieving information for a mode and sessions to be restored” on page 147 for a description of the RESTORE control block. Information is returned for each mode and session that is restored, if requested.

The data area can contain information for more than one LU-mode, as determined by the scope of the request and the size of the data area. The RESTORE structure (ISTSREST) for each LU-mode entry contains the following pointers:

- SRESLDAD points to the session limits data structure (ISTSLD) for that LU-mode.
- SRESESAD points to the session information structure (SRESESS) for the first session being restored for that LU-mode, if any.
- SRENXTAD points to the ISTSREST for the next LU-mode to be restored, if any.

Each session information structure points to the information for the next session being restored for that LU-mode, if any (SRESNXTA). Figure 44 on page 330 shows an example of how this structure might look when LIST=ALL is specified.

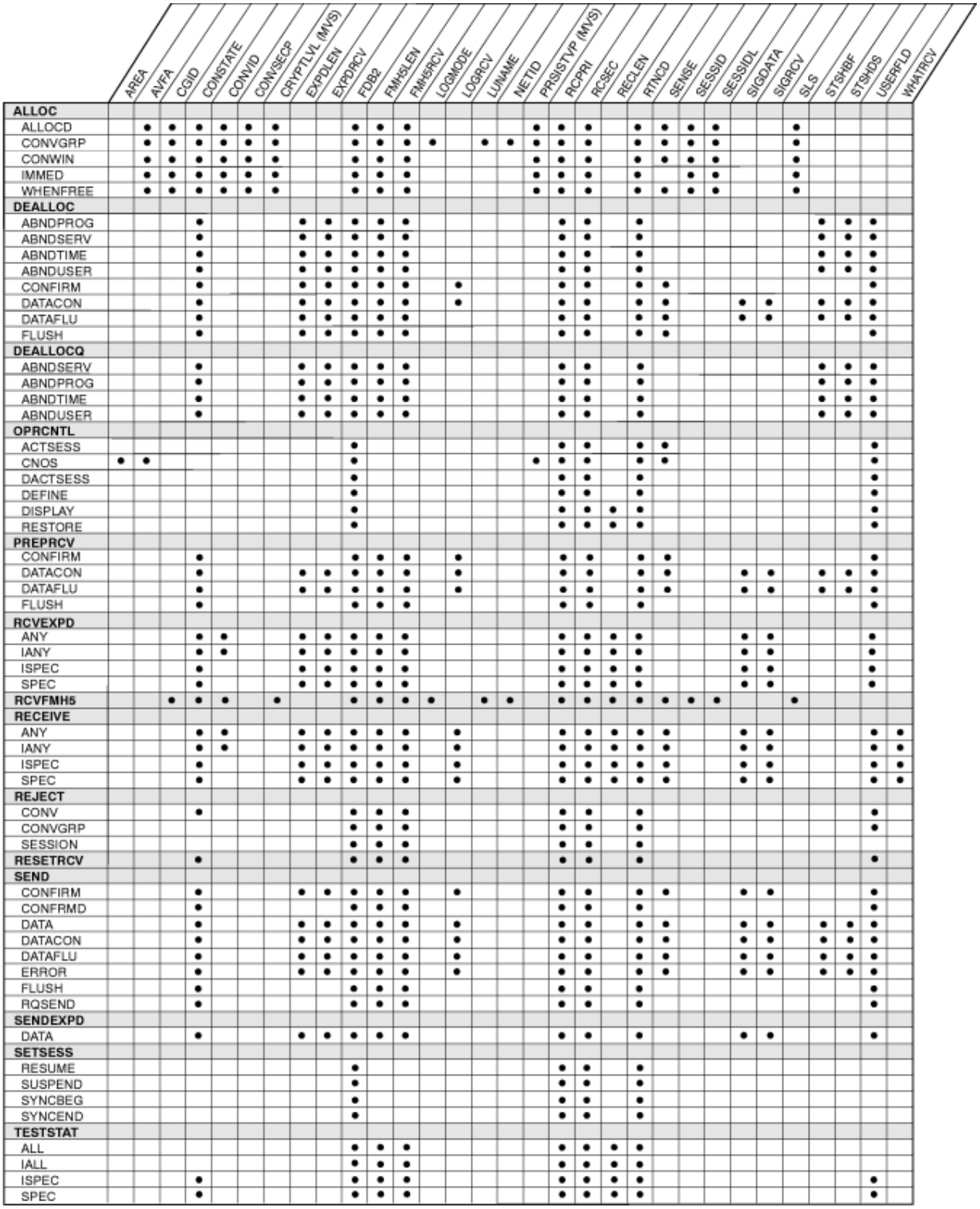


Figure 44. Information for three LU-modes

LU-mode 1 has two sessions associated with it, which are restored. LU-mode 2 has no sessions associated with it, and LU-mode 3 has at least one session associated with it. Only one session associated with LU-mode 3 has been restored. LU-mode 3 might have more sessions that need to be restored. If it does, the mode restored flag (SREMDRS) is set OFF.

If the data area does not have enough space to hold the information that is requested, any remaining sessions and modes are not restored until the application program issues another APPCCMD

CONTROL=OPRCNTL, QUALIFY=RESTORE macroinstruction. If LIST=NOSESS is specified, the session information is not returned. If LIST=NONE is specified, no information is returned.

Logic for retrieving restore information

Figure 45 on page 332 shows the logic for the code in the example program. This flowchart shows an example of how a routine might be written. It uses arbitrary names for things other than the DSECTs. This flowchart also shows how to set up basing for the ISTSREST, ISTSLD, and SRESESS DSECTs. You can use this logic to examine or process the information for each LU-mode contained in the RESTORE control block.

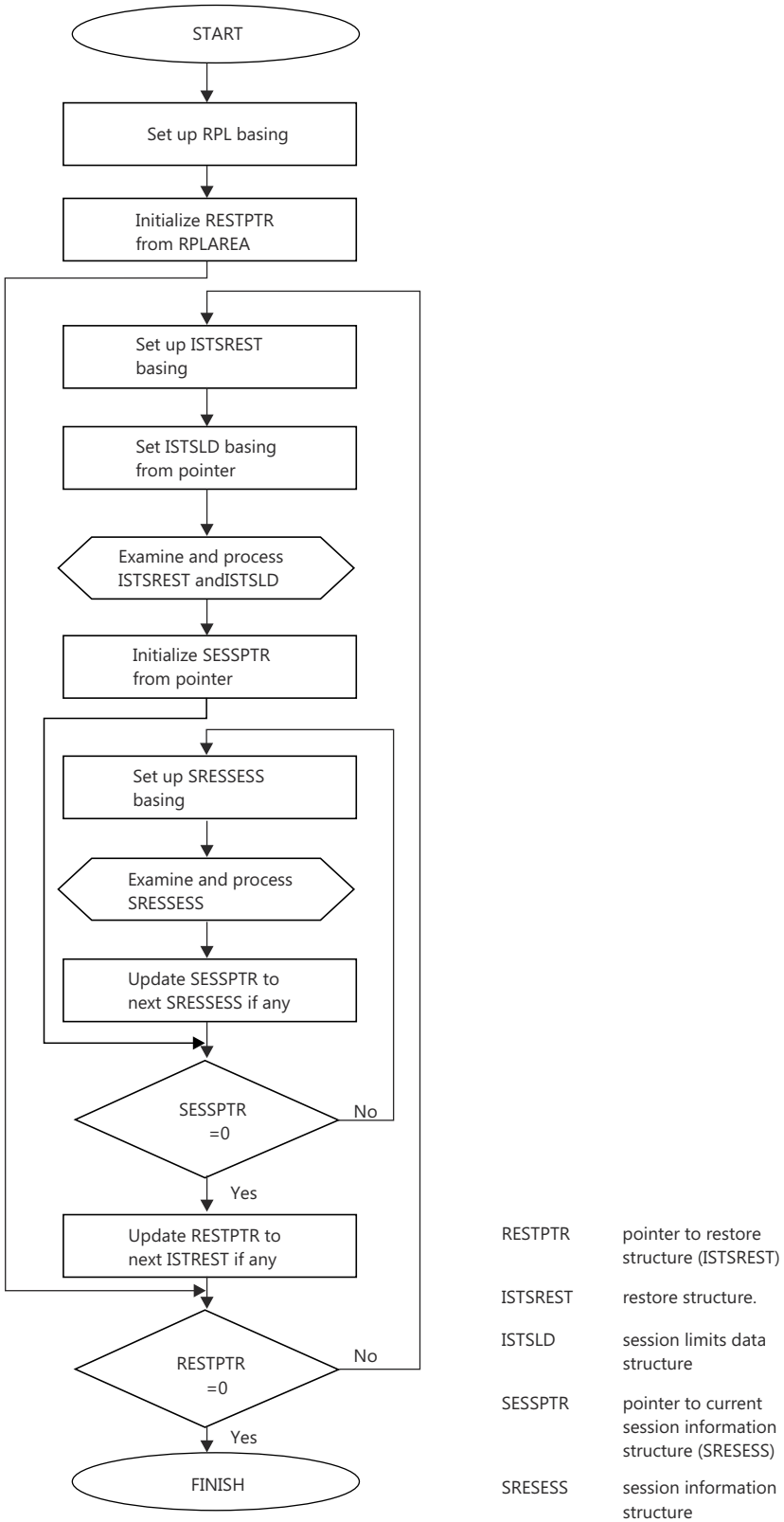


Figure 45. Logic for retrieving RESTORE information

Example program for retrieving restore information

The following example program shows how data can be retrieved when the application program has used either the ALL or NOSESS value for the LIST keyword in the APPCCMD CONTROL=OPRCNTL, QUALIFY=RESTORE macroinstruction.

```
*********************************************************************
*
*  FUNCTION => THIS EXAMPLE SETS UP AND BASES THE NECESSARY DSECTS, IN
*              AN APPROPRIATE LOOPING SCHEME, SO THAT THE RESTORE
*              INFORMATION IN THE USER'S AREA CAN BE EXAMINED AND/OR
*              PROCESSED.
*
*  ENTRY POINT => START
*
*  ASSUMPTION => THE USER ALREADY HAS THE ADDRESS OF ISTRPL, IT IS THE
*                SAME ISTRPL USED IN THE APPCCMD CONTROL=OPRCNTL,QUALIFY=
*                RESTORE FUNCTION CALL.
*
*                THIS CODE ASSUMES THAT THIS FUNCTION HAS BEEN CALLED
*                AND THE ADDRESS OF ISTRPL IS IN THE PTRRPL REGISTER
*
*  REGS USED => SEE THE DESCRIPTIONS BELOW - CONSTANTS AND EQUATES
*                6 = SLDPTR
*                7 = PTRRPL
*                8 = SESSPTR
*                9 = RESTPTR
*********************************************************************
ISTREADS START 0
         USING ISTREADS,15
         USING IFGRPL,PTRRPL            SET UP BASING OF THE RPL FROM
*                                       PTRRPL (THE SAME RPL USED IN THE
*                                       APPCCMD CONTROL=OPRCNTL,QUALIFY=
*                                       RESTORE)
*
         L     RESTPTR,RPLAREA          SET RESTPTR TO USER AREA START
*
*
         B     CHKSREST                 CHECK ADDRESS OF RESTORE STRUCTURE
*                                       BEFORE ATTEMPTING TO PROCESS IT
*
SREST    EQU   *                        RESTORE DATA STRUCTURE (ISTSREST
*                                       ROUTINE
*
*
         USING ISTSREST,RESTPTR         SET UP BASING FOR THE RESTORE
*                                       DATA STRUCTURE
*
         L     SLDPTR,SRESLDAD          OBTAIN SESSION LIMITS DATA
*                                       STRUCTURE (ISTSLD) ADDRESS
*
         USING ISTSLD,SLDPTR            SET UP BASING FOR THE SESSION
*                                       LIMITS DATA STRUCTURE
*        .
*        . { Insert logic to process Restore Data Structure
*        .   and the Session Limits Data Structure.)
*        .
*        .
*
         L     SESSPTR,SRESESAD         OBTAIN NEXT SESSION STRUCTURE
*
*
*
         B     CHKSESS                  CHECK SESSPTR ADDRESS BEFORE
*                                       ATTEMPTING TO PROCESS STRUCTURE
*
SESS     EQU   *                        SESSION INFORMATION STRUCTURE
*                                       ROUTINE
*
         USING SRESESS,SESSPTR          SET UP BASING FOR THE SESSION
*                                       INFORMATION STRUCTURE
*        .
*        .
*        . { Insert logic to process Session Information Structure }
*        .
*        .
*        .
*
*
```

```
         L     SESSPTR,SRESNXTA         GET NEXT SESSION INFORMATION
*                                       STRUCTURE ADDRESS
*
CHKSESS  EQU   *                        CHECK SESSION STRUCTURE ADDRESS
*
         LTR   SESSPTR,SESSPTR          HAVE ALL SESSION INFORMATION
*                                       STRUCTURES BEEN PROCESSED?
*
         BNZ   SESS                     NO, PROCESS THE NEXT SESSION
*                                       INFORMATION STRUCTURE
*
*        L     RESTPTR,SRENXTAD         GET NEXT RESTORE STRUCTURE
*                                       ADDRESS
*
CHKSREST EQU   *                        CHECK RESTORE STRUCTURE ADDRESS
*
         LTR   RESTPTR,RESTPTR          HAVE ALL RESTORE STRUCTURES
*                                       BEEN PROCESSED?
*
         BNZ   SREST                    NO, PROCESS THE NEXT LU,MODE
*                                       RESTORE DATA STRUCTURE
FINISHED EQU   *
***
*               MAINLINE PROGRAM CONSTANTS AND EQUATES
***
*                                       ORDER BIT OF AN ADDRESS
*
SLDPTR   EQU   6                        SESSION LIMITS DATA STRUCTURE
*                                       (ISTSLD) BASE POINTER
*
PTRRPL   EQU   7                        ISTRPL BASE POINTER
*
*
SESSPTR  EQU   8                        SESSION INFORMATION STRUCTURE
*                                       (SRESESS) BASE POINTER
*
RESTPTR  EQU   9                        RESTORE DATA STRUCTURE (ISTSREST)
*                                       BASE POINTER
         LTORG
         EJECT
         IFGRPL AM=VTAM
         EJECT
         ISTSREST
         EJECT
         ISTSLD
         END
```

Appendix E. Architectural specifications

This appendix lists documents that provide architectural specifications for the SNA Protocol.

The APPN Implementers' Workshop (AIW) architecture documentation includes the following architectural specifications for SNA APPN and HPR:

- APPN Architecture Reference (SG30-3422-04)
- APPN Branch Extender Architecture Reference Version 1.1
- APPN Dependent LU Requester Architecture Reference Version 1.5
- APPN Extended Border Node Architecture Reference Version 1.0
- APPN High Performance Routing Architecture Reference Version 4.0
- SNA Formats (GA27-3136-20)
- SNA Technical Overview (GC30-3073-04)

The following RFC also contains SNA architectural specifications:

- RFC 2353 *APPN/HPR in IP Networks APPN Implementers' Workshop Closed Pages Document*

RFCs are available at http://www.rfc-editor.org/rfc.html.

Appendix F. Architectural specifications

This appendix lists documents that provide architectural specifications for the SNA Protocol.

The APPN Implementers' Workshop (AIW) architecture documentation includes the following architectural specifications for SNA APPN and HPR:

- APPN Architecture Reference (SG30-3422-04)
- APPN Branch Extender Architecture Reference Version 1.1
- APPN Dependent LU Requester Architecture Reference Version 1.5
- APPN Extended Border Node Architecture Reference Version 1.0
- APPN High Performance Routing Architecture Reference Version 4.0
- SNA Formats (GA27-3136-20)
- SNA Technical Overview (GC30-3073-04)

The following RFC also contains SNA architectural specifications:

- RFC 2353 *APPN/HPR in IP Networks APPN Implementers' Workshop Closed Pages Document*

RFCs are available at http://www.rfc-editor.org/rfc.html.

Appendix G. Accessibility

Accessible publications for this product are offered through IBM Documentation for z/OS.

If you experience difficulty with the accessibility of any z/OS documentation see How to Send Feedback to IBM to leave documentation feedback.

Notices

This information was developed for products and services that are offered in the USA or elsewhere.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing IBM Corporation North Castle Drive, MD-NC119 Armonk, NY 10504-1785 United States of America

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing Legal and Intellectual Property Law IBM Japan Ltd. 19-21, Nihonbashi-Hakozakicho, Chuo-ku Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

This information could include missing, incorrect, or broken hyperlinks. Hyperlinks are maintained in only the HTML plug-in output for the IBM Documentation. Use of hyperlinks in other output formats of this information is at your own risk.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation Site Counsel 2455 South Road Poughkeepsie, NY 12601-5400 USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Terms and conditions for product documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the IBM website.

Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

IBM Online Privacy Statement

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user, or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is set forth below.

Depending upon the configurations deployed, this Software Offering may use session cookies that collect each user's name, email address, phone number, or other personally identifiable information for purposes of enhanced user usability and single sign-on configuration. These cookies can be disabled, but disabling them will also eliminate the functionality they enable.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Policy at ibm.com®/privacy and IBM's Online Privacy Statement at ibm.com/privacy/details in the section entitled "Cookies, Web Beacons and Other Technologies," and the "IBM Software Products and Software-as-a-Service Privacy Statement" at ibm.com/software/info/product-privacy.

Policy for unsupported hardware

Various z/OS elements, such as DFSMS, JES2, JES3, and MVS, contain code that supports specific hardware servers or devices. In some cases, this device-related element support remains in the product even after the hardware devices pass their announced End of Service date. z/OS may continue to service element code; however, it will not provide service related to unsupported hardware devices. Software problems related to these devices will not be accepted for service, and current service activity will cease if a problem is determined to be associated with out-of-support devices. In such cases, fixes will not be issued.

Minimum supported hardware

The minimum supported hardware for z/OS releases identified in z/OS announcements can subsequently change when service for particular servers or devices is withdrawn. Likewise, the levels of other software products supported on a particular release of z/OS are subject to the service support lifecycle of those products. Therefore, z/OS and its product publications (for example, panels, samples, messages, and product documentation) can include references to hardware and software that is no longer supported.

- For information about software support lifecycle, see: IBM Lifecycle Support for z/OS (www.ibm.com/software/support/systemsz/lifecycle)
- For information about currently-supported IBM hardware, contact your IBM representative.

Programming interface information

This publication documents intended Programming Interfaces that allow the customer to write programs to obtain the services of z/OS Communications Server.

Policy for unsupported hardware

Various z/OS elements, such as DFSMS, HCD, JES2, JES3, and MVS, contain code that supports specific hardware servers or devices. In some cases, this device-related element support remains in the product even after the hardware devices pass their announced End of Service date. z/OS may continue to service element code; however, it will not provide service related to unsupported hardware devices. Software problems related to these devices will not be accepted for service, and current service activity will cease if a problem is determined to be associated with out-of-support devices. In such cases, fixes will not be issued.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at Copyright and trademark information at www.ibm.com/legal/copytrade.shtml.

Bibliography

This bibliography contains descriptions of the documents in the z/OS Communications Server library.

z/OS Communications Server documentation is available online at the z/OS Internet Library web page at http://www.ibm.com/systems/z/os/zos/library/bkserv/.

z/OS Communications Server library updates

Updates to documents are also available on RETAIN and in information APARs (info APARs). Go to https://www.ibm.com/mysupport to view information APARs.

- z/OS Communications Server V2R1 New Function APAR Summary
- z/OS Communications Server V2R2 New Function APAR Summary
- z/OS Communications Server V2R3 New Function APAR Summary
- z/OS Communications Server V2R4 New Function APAR Summary

z/OS Communications Server information

z/OS Communications Server product information is grouped by task in the following tables.

Planning

Title	Number	Description
z/OS Communications Server: New Function Summary	GC27-3664	This document is intended to help you plan for new IP or SNA functions, whether you are migrating from a previous version or installing z/OS for the first time. It summarizes what is new in the release and identifies the suggested and required modifications needed to use the enhanced functions.
z/OS Communications Server: IPv6 Network and Appl Design Guide	SC27-3663	This document is a high-level introduction to IPv6. It describes concepts of z/OS Communications Server's support of IPv6, coexistence with IPv4, and migration issues.

Resource definition, configuration, and tuning

Title	Number	Description
z/OS Communications Server: IP Configuration Guide	SC27-3650	This document describes the major concepts involved in understanding and configuring an IP network. Familiarity with the z/OS operating system, IP protocols, z/OS UNIX System Services, and IBM Time Sharing Option (TSO) is recommended. Use this document with the z/OS Communications Server: IP Configuration Reference.

Title	Number	Description
z/OS Communications Server: IP Configuration Reference	SC27-3651	This document presents information for people who want to administer and maintain IP. Use this document with the z/OS Communications Server: IP Configuration Guide. The information in this document includes: • TCP/IP configuration data sets • Configuration statements • Translation tables • Protocol number and port assignments
z/OS Communications Server: SNA Network Implementation Guide	SC27-3672	This document presents the major concepts involved in implementing an SNA network. Use this document with the z/OS Communications Server: SNA Resource Definition Reference.
z/OS Communications Server: SNA Resource Definition Reference	SC27-3675	This document describes each SNA definition statement, start option, and macroinstruction for user tables. It also describes NCP definition statements that affect SNA. Use this document with the z/OS Communications Server: SNA Network Implementation Guide.
z/OS Communications Server: SNA Resource Definition Samples	SC27-3676	This document contains sample definitions to help you implement SNA functions in your networks, and includes sample major node definitions.
z/OS Communications Server: IP Network Print Facility	SC27-3658	This document is for systems programmers and network administrators who need to prepare their network to route SNA, JES2, or JES3 printer output to remote printers using TCP/IP Services.

Operation

Title	Number	Description
z/OS Communications Server: IP User's Guide and Commands	SC27-3662	This document describes how to use TCP/IP applications. It contains requests with which a user can log on to a remote host using Telnet, transfer data sets using FTP, send electronic mail, print on remote printers, and authenticate network users.
z/OS Communications Server: IP System Administrator's Commands	SC27-3661	This document describes the functions and commands helpful in configuring or monitoring your system. It contains system administrator's commands, such as TSO NETSTAT, PING, TRACERTE and their UNIX counterparts. It also includes TSO and MVS commands commonly used during the IP configuration process.
z/OS Communications Server: SNA Operation	SC27-3673	This document serves as a reference for programmers and operators requiring detailed information about specific operator commands.
z/OS Communications Server: Quick Reference	SC27-3665	This document contains essential information about SNA and IP commands.

Customization

Title	Number	Description
z/OS Communications Server: SNA Customization	SC27-3666	This document enables you to customize SNA, and includes the following information: • Communication network management (CNM) routing table • Logon-interpret routine requirements • Logon manager installation-wide exit routine for the CLU search exit • TSO/SNA installation-wide exit routines • SNA installation-wide exit routines

Writing application programs

Title	Number	Description
z/OS Communications Server: IP Sockets Application Programming Interface Guide and Reference	SC27-3660	This document describes the syntax and semantics of program source code necessary to write your own application programming interface (API) into TCP/IP. You can use this interface as the communication base for writing your own client or server application. You can also use this document to adapt your existing applications to communicate with each other using sockets over TCP/IP.
z/OS Communications Server: IP CICS Sockets Guide	SC27-3649	This document is for programmers who want to set up, write application programs for, and diagnose problems with the socket interface for CICS® using z/OS TCP/IP.
z/OS Communications Server: IP IMS Sockets Guide	SC27-3653	This document is for programmers who want application programs that use the IMS TCP/IP application development services provided by the TCP/IP Services of IBM.
z/OS Communications Server: IP Programmer's Guide and Reference	SC27-3659	This document describes the syntax and semantics of a set of high-level application functions that you can use to program your own applications in a TCP/IP environment. These functions provide support for application facilities, such as user authentication, distributed databases, distributed processing, network management, and device sharing. Familiarity with the z/OS operating system, TCP/IP protocols, and IBM Time Sharing Option (TSO) is recommended.
z/OS Communications Server: SNA Programming	SC27-3674	This document describes how to use SNA macroinstructions to send data to and receive data from (1) a terminal in either the same or a different domain, or (2) another application program in either the same or a different domain.
z/OS Communications Server: SNA Programmer's LU 6.2 Guide	SC27-3669	This document describes how to use the SNA LU 6.2 application programming interface for host application programs. This document applies to programs that use only LU 6.2 sessions or that use LU 6.2 sessions along with other session types. (Only LU 6.2 sessions are covered in this document.)
z/OS Communications Server: SNA Programmer's LU 6.2 Reference	SC27-3670	This document provides reference material for the SNA LU 6.2 programming interface for host application programs.

Title	Number	Description
z/OS Communications Server: CSM Guide	SC27-3647	This document describes how applications use the communications storage manager.

Diagnosis

Title	Number	Description
z/OS Communications Server: IP Diagnosis Guide	GC27-3652	This document explains how to diagnose TCP/IP problems and how to determine whether a specific problem is in the TCP/IP product code. It explains how to gather information for and describe problems to the IBM Software Support Center.
z/OS Communications Server: ACF/TAP Trace Analysis Handbook	GC27-3645	This document explains how to gather the trace data that is collected and stored in the host processor. It also explains how to use the Advanced Communications Function/Trace Analysis Program (ACF/TAP) service aid to produce reports for analyzing the trace data information.
z/OS Communications Server: SNA Diagnosis Vol 1, Techniques and Procedures and z/OS Communications Server: SNA Diagnosis Vol 2, FFST Dumps and the VIT	GC27-3667 GC27-3668	These documents help you identify an SNA problem, classify it, and collect information about it before you call the IBM Support Center. The information collected includes traces, dumps, and other problem documentation.
z/OS Communications Server: SNA Data Areas Volume 1 and z/OS Communications Server: SNA Data Areas Volume 2	GC31-6852 GC31-6853	These documents describe SNA data areas and can be used to read an SNA dump. They are intended for IBM programming service representatives and customer personnel who are diagnosing problems with SNA.

Messages and codes

Title	Number	Description
z/OS Communications Server: SNA Messages	SC27-3671	This document describes the ELM, IKT, IST, IUT, IVT, and USS messages. Other information in this document includes: • Command and RU types in SNA messages • Node and ID types in SNA messages • Supplemental message-related information
z/OS Communications Server: IP Messages Volume 1 (EZA)	SC27-3654	This volume contains TCP/IP messages beginning with EZA.
z/OS Communications Server: IP Messages Volume 2 (EZB, EZD)	SC27-3655	This volume contains TCP/IP messages beginning with EZB or EZD.
z/OS Communications Server: IP Messages Volume 3 (EZY)	SC27-3656	This volume contains TCP/IP messages beginning with EZY.
z/OS Communications Server: IP Messages Volume 4 (EZZ, SNM)	SC27-3657	This volume contains TCP/IP messages beginning with EZZ and SNM.
z/OS Communications Server: IP and SNA Codes	SC27-3648	This document describes codes and other information that appear in z/OS Communications Server messages.

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

Product Number: 5655-ZOS

SC27-3669-70