SC38-0256-0
File No. S370-30

Advanced Communications Function for VTAM (ACF/VTAM)

Program Product

Macro Language Guide

Program Numbers 5746-RC3 (DOS/VS)
5735-RC2 (OS/VS)

IBM

First Edition (July 1977)

This edition applies to the initial DOS/VS version of ACF/VTAM (Program Number 5746-RC3) and to the initial OS/VS version of ACF/VTAM for OS/VS1, OS/VS2 SVS, and OS/VS2 MVS (Program Number 5735-RC2). Information about the optional Multisystem Networking Facility of ACF/VTAM is included. The information in this publication should be used for planning purposes until ACF/VTAM becomes available for your operating system.

Preface

This book describes how to write application programs for the Advanced Communications Function for VTAM (ACF/VTAM) and for the Multisystem Networking Feature that is available with ACF/VTAM. This book is a guide to using ACF/VTAM macro instructions in the data communication part of an ACF/VTAM application program. As such, this book is a companion for and is to be used with *ACF/VTAM Macro Language Reference,* SC38-0261.

Who This Book Is For
This book is for any programmer, whether an application programmer or a system programmer, who must write a program that uses ACF/VTAM macro instructions. The reader is assumed to be familiar with Basic Assembler Language.

How This Book Is Organized
The body of this book describes the use of record-mode macro instructions to communicate with Systems Network Architecture (SNA) devices. Use of basic-mode macro instructions to communicate with BSC devices, start-stop devices, and local non-SNA devices are described in Appendix A.

This book contains three parts:

Part 1, "ACF/VTAM Application Program Concepts and Language," should be read as an introduction. It contains chapters on:

What an ACF/VTAM Application Program Is. This chapter describes on an introductory level how an ACF/VTAM application program is part of a data communication system and generally how an ACF/VTAM application program is organized.

The ACF/VTAM Language. This chapter summarizes the ACF/VTAM application program macro instructions and control blocks and relates them to each other.

Part 2, "Writing an ACF/VTAM Application Program," describes ACF/VTAM application program facilities in detail and with examples. It contains chapters on:

Organizing a Program

Opening and Closing a Program

Connecting and Disconnecting Logical Units

Communicating with Logical Units

Using Exit Routines

Manipulating Control Blocks

Handling Errors and Special Conditions

Debugging a Program

Part 3, "Sample Programs," contains the general logic of two sample programs:

A synchronous program

An asynchronous program that communicates with 3600 and non-SNA terminals

Appendix A includes information on using ACF/VTAM to communicate with BSC devices, start-stop devices, and local non-SNA devices. Appendix B summarizes the indicators and commands that can be used to control the exchange of messages. Appendix C contains data-flow diagrams that show sequences in which messages, commands, and responses are exchanged in various types of operations. Appendix D contains a coded example of a primary application program, based on Sample Program 1 in Part 3. Appendix E contains a coded example of an ACF/VTAM application program that uses authorized path. Appendix F contains a coded example of a primary application program and secondary application program that work together.

How to Use This Book
Read Part 1 for an introduction to ACF/VTAM application program concepts and to the ACF/VTAM macro instructions. Use Parts 2 and 3 and the appendixes in conjunction with *ACF/VTAM Macro Language Reference* when designing and coding a program.

Related Publications
These publications are related to this manual:

ACF/VTAM Concepts and Planning, GC38-0282. This manual describes the concepts and tasks involved in designing, defining, and using an ACF/VTAM data communication network, including the writing of ACF/VTAM application programs. This book is useful for understanding how ACF/VTAM application programs fit into an installation's teleprocessing system.

ACF/VTAM Macro Language Reference, SC38-0261. This manual, which describes ACF/VTAM macro instructions and operands in detail, must be used to write an ACF/VTAM application program. *ACF/VTAM Macro Language Guide* is a companion to *ACF/VTAM Macro Language Reference.*

ACF/VTAM Program Operator Guide, SC38-0257. This manual describes how to write a program-operator application program for use with ACF/VTAM. This information supplements *ACF/VTAM Macro Language Guide* and *ACF/VTAM Macro Langauge Reference*.

ACF/VTAM System Programmer's Guide, SC38-0268 (DOS/VS), SC38-0258 (OS/VS1), SC38-0267 (OS/VS2 SVS), and SC38-0262 (OS/VS2 MVS). This manual describes how to define, tailor, tune, and maintain an ACF/VTAM system. It includes information on the choice and specification of installation options. Information on definition of devices and programs can be found in this manual.

Systems Network Architecture; General Information, GA27-3102. This manual describes Systems Network Architecture concepts that might be useful when writing an ACF/VTAM application program.

Systems Network Architecture; Format and Protocol Reference Manual: Architecture Logic, SC30-3112. This manual is a very detailed description of the SNA architecture. It is intended for system programmers. It may be useful for persons who want an in-depth knowledge of the SNA protocols.

Contents

Figures

Part 1. ACF/VTAM Application Program Concepts and Language

Chapter 1. What an ACF/VTAM Application Program Is. This chapter introduces ACF/VTAM application program concepts and facilities, which are discussed in more detail, with examples, in Parts 2 and 3. This chapter gives the reader an overview of ACF/VTAM application program concepts by showing how an ACF/VTAM application program fits into an installation's teleprocessing system and by showing the principal elements in any ACF/VTAM application program. The facilities provided and major choices involved in writing an ACF/VTAM application program are summarized.

Chapter 2. The ACF/VTAM Language. This chapter summarizes ACF/VTAM macro instructions and discusses their general characteristics. It explains the relationship among control blocks defined by ACF/VTAM macro instructions and the executable ACF/VTAM macro instructions that use these control blocks. These relationships are discussed in the context of three things that every ACF/VTAM application program must do: open and close the program, connect and disconnect logical units, and communicate with logical units.

Chapter 1. What an ACF/VTAM Application Program Is

This chapter provides an overview of ACF/VTAM application programs. It describes what an ACF/VTAM application program is by:

Showing an ACF/VTAM application program as part of an ACF/VTAM data communication system

Showing an ACF/VTAM application program's relationship to the concepts of Systems Network Architecture (SNA)

Showing the major programming elements in an ACF/VTAM application program

An ACF/VTAM Application Program as Part of an ACF/VTAM System

Figure 1-1 shows an ACF/VTAM application program as part of an ACF/VTAM data communication system. The circled numbers in Figure 1-1 refer to major parts of the system; those circled numbers appear beside headings below and tie the discussion below to the related parts of the figure. An ACF/VTAM application program can communicate with synchronous data link control (SDLC) logical units, with binary synchronous communication (BSC) and start-stop devices, with local SNA devices, with local non-SNA devices, and with other ACF/VTAM application programs. The body of this book describes communication with local SNA devices, remote SNA devices, and certain non-SNA (BSC and local) devices. Appendix A describes communication with BSC devices, start-stop devices, and local 3270s not treated as SNA devices.

The ACF/VTAM Application Program ①

An ACF/VTAM application program can contain two types of instructions: communication instructions and processing instructions. The application program always contains communication instructions, which are the instructions that send and receive messages and control other aspects of communication between the program and other elements in the network. The program usually also contains processing instructions, which are the instructions that manipulate the data before it is sent or after it is received, but the program does not have to contain processing instructions.

If an ACF/VTAM application program does contain processing instructions and if the program is small, those instructions may be interleaved with communication instructions. More commonly, however, the processing instructions are written separately, with an interface defined between processing parts of the program and the communication part of the program. This separation of function allows each part to be created separately and means that changes or additions to one part will not affect other parts. The processing part of an ACF/VTAM application program can be written in a higher-level language, such as PL/I. The communication part, which uses ACF/VTAM macro instructions, is written in assembler language.

ACF/VTAM application programs can share the resources of the system; that is, the application programs can use the same communications controllers, cluster controllers, and communication lines to reach logical units. For example, in Figure 1-1, application programs A and B use the same communications controller (at 5) and the same SDLC link to reach terminals and logical units (at 6 and 7). The application programs, however, are not aware that they are sharing these resources because ACF/VTAM, the network control program (NCP), and other programming elements in the network handle communications in such a way that the programs do not know they are sharing the resources. One restriction on sharing resources is that only one session can exist between any two logical units in a network at the same time. A second restriction is that a logical unit cannot establish a session with itself.

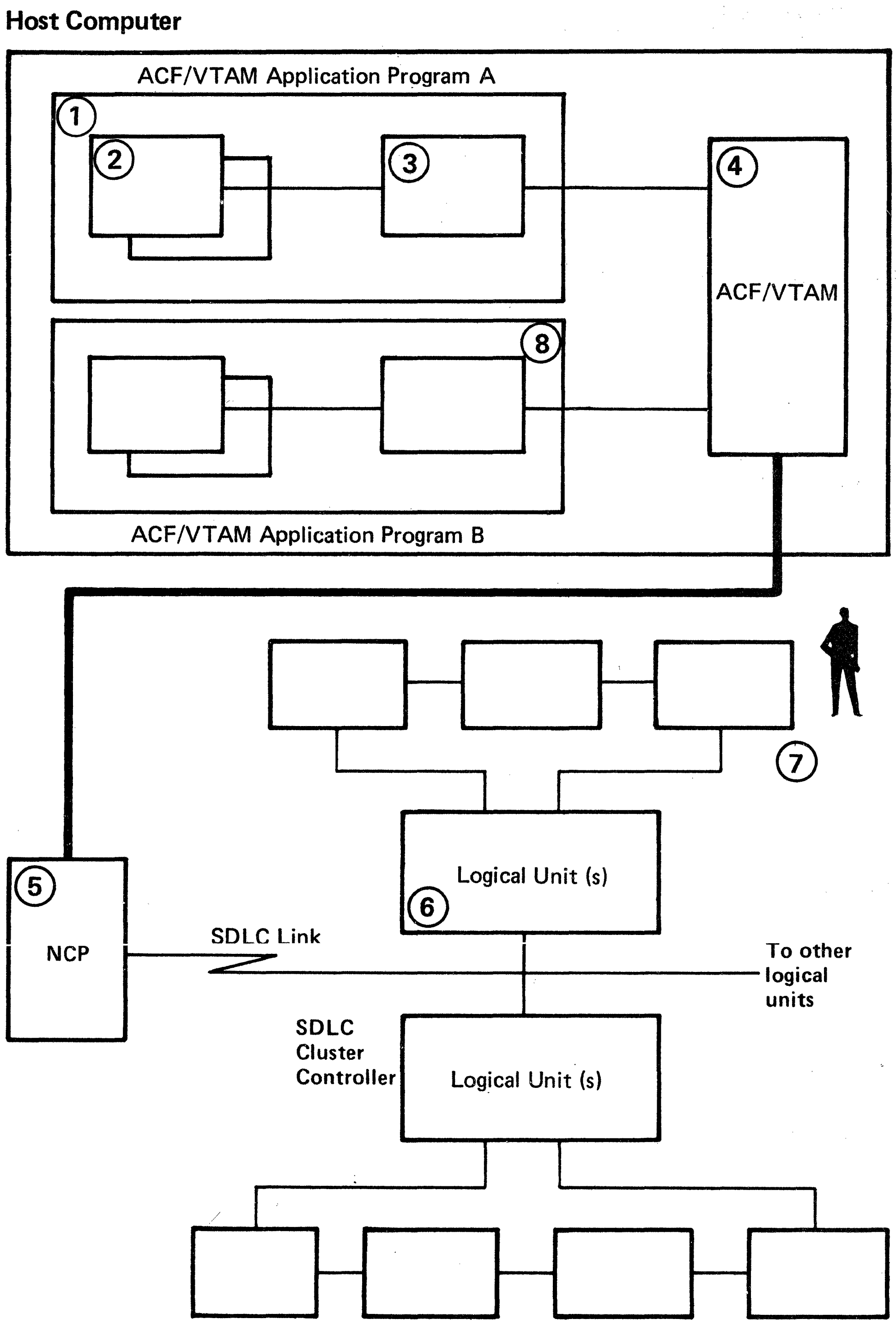


Figure 1-1. ACF/VTAM Application Programs in an ACF/VTAM Data Communication System

The Processing Part (2)

The instructions in the processing part of an application program can be written in assembler language or in a higher-level language, such as PL/I or COBOL. If written in assembler language, the instructions can be interleaved with the communication instructions in the program. But more commonly, as shown in Figure 1-1, the processing part is separate and requests data communication services by calling or branching to the communication part of the program. Many programs will contain several processing parts (routines or modules) that use a common communication part.

The Communication Part (3)

This part of the ACF/VTAM application program contains macro instructions and associated control blocks used to connect and communicate with logical units that have been defined to ACF/VTAM. The programming elements that make up the communication part are discussed further in this chapter under "The Major Programming Elements in an ACF/VTAM Application Program."

ACF/VTAM (4)

ACF/VTAM controls the data communication network. Logical units (including other application programs) are defined as part of the network during ACF/VTAM definition and are then activated by start procedures or network operator commands. The ACF/VTAM application program then requests connection (on its own initiative or as the result of a logon) to one or more active logical units. Once connected, the program requests ACF/VTAM to perform data-transfer operations with each logical unit. In addition to managing the network and building channel programs, ACF/VTAM performs such services as input and output data buffering, automatic scheduling of application program exit routines, and sequence numbering of outbound messages. ACF/VTAM requests the operating system to execute channel programs it has built; the channel programs result in communication with local or remote logical units through a local communications controller or cluster controller.

The Network Control Program (5)

On receiving the input or output requests and associated data, the network control program (NCP) in the 3704 or 3705 Communications Controller does what is required to communicate with logical units on data communication lines. Many functions previously performed by the access method (for example, BTAM) or application program are now performed by the communications controller; for example, the controller schedules line activity, retries operations after transmission errors, and collects error statistics.

The Logical Unit (6)

An ACF/VTAM application program communicates with logical units. A logical unit can be:

An SNA terminal used directly by a terminal operator, or a program in a terminal, a terminal control unit, or a cluster controller (such as a 3601 or 3791 controller)

A non-SNA 3270 device (local or remote) used in a record-mode session

Logical units controlled by TCAM or by another ACF/VTAM

Another ACF/VTAM application program

When the unqualified term *logical unit* is used in this book, it refers to any or all of the above. The term *device-type logical unit* refers to any logical unit other than another application program. To distinguish a logical unit that is another ACF/VTAM application program, it is called a *secondary application program.*

Though not shown in Figure 1-1, the ACF/VTAM application program can also communicate with certain BSC and start-stop devices. The set of macro instructions available for communicating with these devices is different from that used for communicating with logical units. The method of communicating with BSC terminals, start-stop terminals, and local non-SNA devices in basic mode is described in Appendix A.

In general, for programmable SNA devices, the user defines which processing functions will take place in a program in the programmable device (such as a cluster controller) and which in the ACF/VTAM application program in the host computer. The user must coordinate the cluster controller program and the ACF/VTAM application program so that they work together.

The Terminal Operator and the Batch Function ⑦

If the ACF/VTAM application program communicates with a cluster controller program rather than with a nonprogrammable terminal, the ACF/VTAM application program may not need to know about terminal operator actions. The logical unit will determine whether and how data received from a terminal operator will go to the ACF/VTAM application program and whether and how data received from the ACF/VTAM application program will go to the terminal operator.

An ACF/VTAM application program can also communicate with a batch-transmission or batch-reception program in a cluster controller (such as the 3791 batch function). To participate effectively in the batch transmissions, the ACF/VTAM application program does not need to know the original source of or the eventual disposition of data received from or sent to the subsystem batch program.

Another ACF/VTAM Application Program ⑧

An ACF/VTAM application program can also communicate with another ACF/VTAM application program. The two application programs can be in the same host computer or in different host computers. In this kind of communication in a particular session, one application program adheres to a set of defined primary protocols and is known as the *primary* application program. The other application program adheres to a set of secondary protocols and is known as the *secondary* application program.

An ACF/VTAM application program can be both a primary and a secondary application program at the same time. It can be a primary application program in its sessions with logical units while it is also functioning as a secondary application program in sessions with other ACF/VTAM application programs.

Systems Network Architecture (SNA) Concepts in ACF/VTAM

ACF/VTAM follows SNA concepts and uses SNA protocols to connect and communicate with elements in a data communication network. Several SNA concepts provide helpful background information for the programmer who writes an ACF/VTAM application program. Those concepts are:

Network addressable units

Primary and secondary logical units

Sessions

Domains

The SNA Concept of Network Addressable Units

Each element in a network to which a data or control message can be sent is assigned a network address by ACF/VTAM. Each element with such an address is known as a *network addressable unit* (NAU). The network address uniquely identifies the element, regardless of whether the element is a device (such as a terminal or terminal control unit), a program (such as an application program in a cluster controller or terminal), or a portion of ACF/VTAM. For ACF/VTAM and other elements in the data communication network, the network address contains the information necessary to route a message to its destination.

Three types of network addressable units are defined by SNA and recognized by ACF/VTAM. The three types are: (1) system services control point (SSCP), (2) physical units (PUs), and (3) logical units (LUs). Figure 1-2 shows the location of these types of network addressable units in a simplified network.

The *system services control point* is a unit of coding in ACF/VTAM that manages the network and has primary control over communications. The SSCP performs functions such as bringing up the network and shutting it down, establishing and disestablishing connections (sessions) between units, and reacting to network problems (such as failure of a link or unit). To perform these functions, the SSCP must be able to communicate with physical units and logical units in the network under its control.

A *physical unit* is not literally a physical device in the network. Rather, a physical unit is a portion of a device (usually programming or circuitry, or both) that performs control functions for the device in which it is located and, in some cases, for other devices that are attached to the PU-containing device. For the devices under its control, the physical unit takes action during activation and deactivation, during error recovery and resynchronization, during testing, and during gathering of statistics on operation of the device. Each device in the network is associated with a physical unit.

The physical unit may exist either within the device or within an attached controlling device. The physical unit exists within a host computer, a communications controller, and a cluster controller. For a terminal, however, the physical unit may be within the terminal or it may be within the terminal control unit, cluster controller, or communications controller to which the terminal is attached.

A *logical unit* is a device or program by which an *end user* (a terminal operator or an input/output mechanism) gains access to the data communication network. To the network, a logical unit is the source of a message coming into the network. But the logical unit may or may not be the original source. The contents of the message or the information on which the message is based may have originated at a device controlled by the logical unit. (For example, in a 3601 cluster controller, the logical unit is a 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 it is the logical unit [the program] in the 3601 that uses the input to create a message and begin transmission of the message.) Similarly, the network sees a logical unit as the destination of a message, but the logical unit may actually pass the message on to a device for recording, printing, or displaying to a terminal operator. (For example, a message received by a logical unit [a program] in a 3601 may be passed on to a finance terminal to be displayed on the screen of that terminal.) In some cases, however, the logical unit is an intrinsic part of the device at which the message is displayed (for example, a 3767 terminal contains the logical unit and is the input/output device).

An ACF/VTAM application program is also a logical unit. ACF/VTAM sees it as an originator of and destination for messages. But there can be other programs in the host computer that interface with an ACF/VTAM application program and to which the contents of messages can be passed or from which the contents of messages can be received. Thus, although the ACF/VTAM application program is the logical unit, the messages it handles may be used by another program. In this case, the other program is the end user.

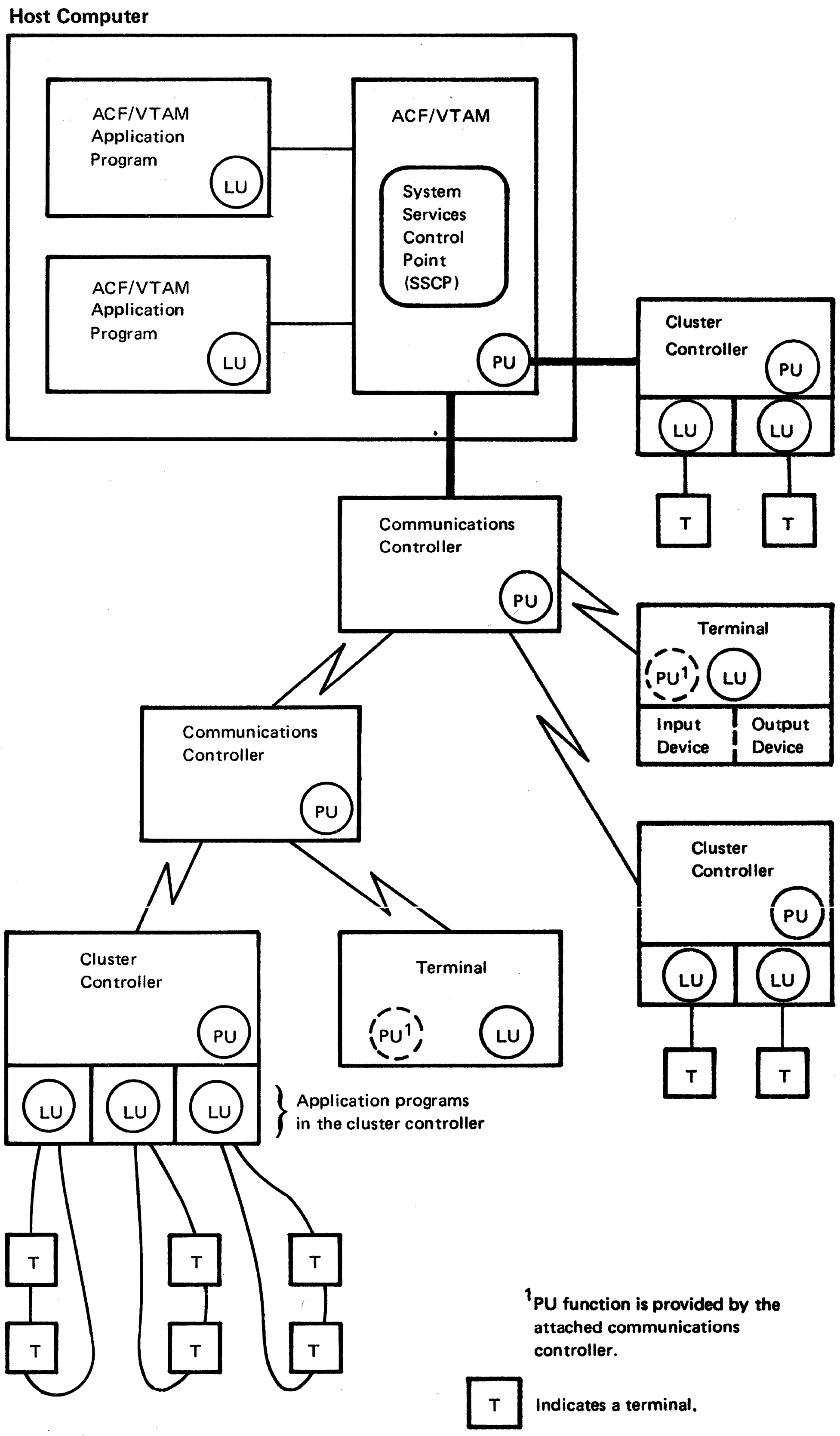

Host Computer
ACF/VTAM Application Program
LU
ACF/VTAM Application Program
LU
ACF/VTAM
System Services Control Point (SSCP)
PU
Cluster Controller
PU
LU
LU
T
T
Communications Controller
PU
Terminal
PU1
LU
Input Device
Output Device
Communications Controller
PU
Cluster Controller
PU
LU
LU
T
T
Cluster Controller
PU
Terminal
PU1
LU
LU
LU
LU
Application programs in the cluster controller
T
T
T
T
T
T
1PU function is provided by the attached communications controller.
T
Indicates a terminal.

Figure 1-2. The SSCP, Physical Units, and Logical Units in a Network

Of the three types of network addressable units, an ACF/VTAM application programmer is concerned only with logical units, and is not concerned with the SSCP and the physical units. An ACF/VTAM application programmer is concerned with logical units because his or her program (which itself is a logical unit) will communicate with other logical units (for example, terminals, programs in cluster controllers or terminals, and/or other ACF/VTAM application programs). An ACF/VTAM application program does not communicate directly with the SSCP or physical unit, but commands issued by the program may lead to actions by the SSCP or physical unit.

When an application program establishes connection with a logical unit (by issuing an OPNDST or OPNSEC macro instruction), a 32-bit communication identifier (CID) is returned in two control blocks used in making the connection (the request parameter list [RPL] and the node initialization block [NIB]). The CID identifies the two logical units involved in a communication session. Whenever the application program is to send a message to the logical unit, the CID must be in the RPL used to send the message. The programmer can move the CID out of and back into the RPL, but the programmer does not directly use and must not change the contents of the CID.

The SNA Concept of Primary and Secondary Logical Units

In communication between two logical units, one logical unit acts as the primary end of the session (by using primary protocols), the other as the secondary end (by using secondary protocols).

The same ACF/VTAM application program can be primary on some sessions and secondary on other sessions at the same time. For example, it can perform primary functions in its communications with terminals and logical units (including other ACF/VTAM application programs), and it can perform secondary functions in its communications with another application program.

For more information on what primary and secondary application programs can and cannot do, see "How a Secondary Application Program Requests Connection" in Chapter 5.

The SNA Concept of Sessions

Before two units in a network can communicate with each other, the units must be tied together in what is known as a *session.* In an SNA network, several different types of sessions are established, including SSCP-SSCP sessions, SSCP-PU sessions, SSCP-LU sessions, and LU-LU sessions.

When a network includes more than one host computer and therefore more than one ACF/VTAM (or ACF/VTAM in one or more hosts and ACF/TCAM in one or more other hosts), a session called an *SSCP-SSCP session* must be established between the SSCP in one ACF/VTAM and each other SSCP with which the first SSCP will communicate.

Within the machine configuration controlled by each SSCP, different kinds of sessions are established in stages. The SSCP must first establish an *SSCP-PU session* with each physical unit that is active in the configuration. Then, for each active logical unit associated with a physical unit, the SSCP must establish an *SSCP-LU session.* And finally, when a pair of logical units indicate that they want to communicate with each other, the SSCP must establish an *LU-LU session* between them. The paragraphs that follow describe the steps in establishing these sessions, with the circled numbers beside the headings serving as keys to the circled numbers in Figure 1-3.

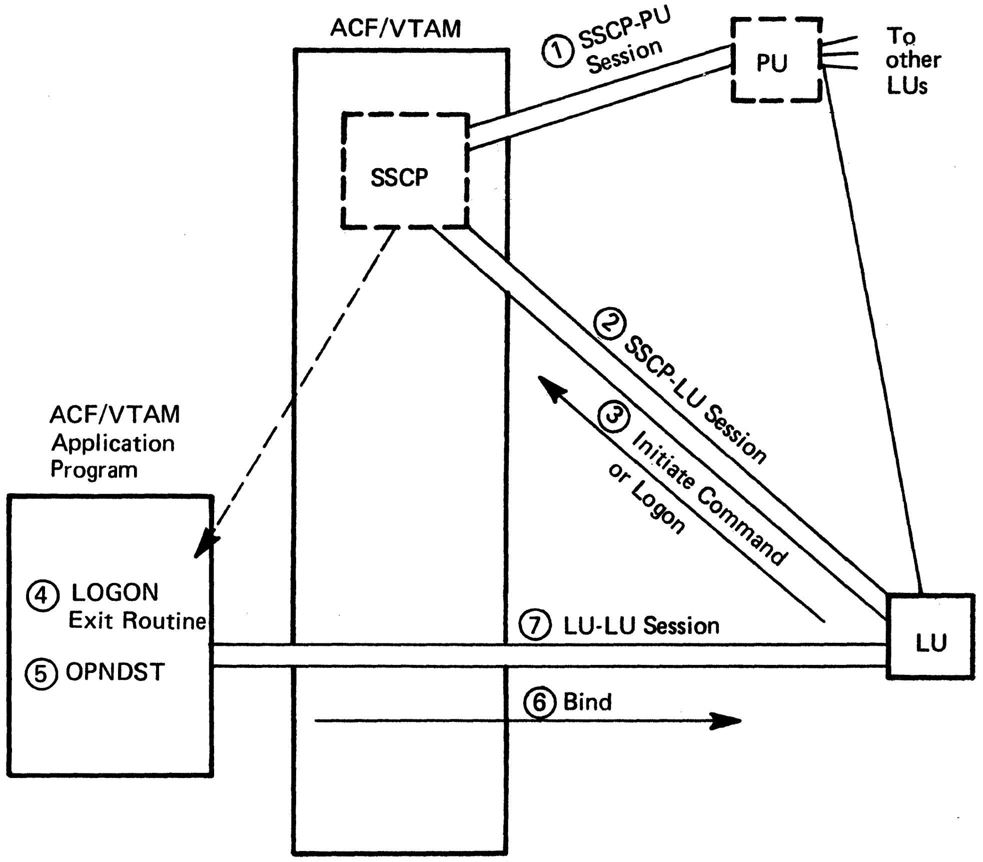


Figure 1-3. Establishing an LU-LU Session

The SSCP-PU Session ①

To get ready for an SSCP-LU session, the SSCP must first establish a session with the physical unit that controls the logical unit. This type of session is called an *SSCP-PU session.* This session is used to exchange messages and commands that pertain to startup and shutdown of the machine configuration or the individual physical unit and to the recovery of operations after a device or link failure. After the SSCP-PU session has been established, the SSCP can attempt to establish a session with any active logical unit associated with that physical unit. The SSCP-PU session is established on a nonswitched line as soon as the physical unit is activated. On a switched line, the session is established following a dial-in or dial-out operation. For local SNA devices, the session is established when physical connection is established. In ACF/VTAM, the SSCP-PU session is established by the SSCP; an ACF/VTAM application program does not itself take any direct action to establish that session.

The SSCP-LU Session ②

Once a session has been established between the SSCP and a physical unit, the SSCP can issue commands to establish a session between itself and any active logical unit associated with the physical unit. This type of session is called an *SSCP-LU session.* The establishment of this session allows SNA commands to flow back and forth between the logical unit and the SSCP. These commands pertain mainly to connection and disconnection. In ACF/VTAM, the SSCP-LU session is established by the SSCP; an ACF/VTAM application program does not itself take any direct action to establish the session.

Initiate Command or Logon (3)

After the SSCP has established a session with a logical unit, the logical unit can attempt to initiate a session with an ACF/VTAM application program. Action by the logical unit to start such a session is usually initiated when a terminal operator communicates with the logical unit and indicates that he or she wants to work with an application program in the host computer. The logical unit either uses the logon information entered by the terminal operator to create an Initiate command to be sent to the SSCP, or the logical unit passes the logon information from the terminal operator on to the SSCP in the form in which it was received from the operator.

In addition, after the SSCP-LU session has been established, a secondary ACF/VTAM application program can request that a session be established with a primary ACF/VTAM application program. In this case, the secondary application issues a special macro instruction (the REQSESS macro instruction). This macro instruction causes an Initiate command (containing logon information) to be sent to the SSCP.

In either case, when the logon information reaches the SSCP, the SSCP notifies the ACF/VTAM application program that the logon has been received and should be processed. The logon information includes session parameters and, optionally, a user logon message. The user logon message is particular data that the terminal operator or logical unit wants to be passed to the ACF/VTAM application program. When the application program is notified that the logon has been received, the session parameters and the user logon message are made available for inspection by the program. Session parameters are a set of codes that indicate the communication rules that the logging-on unit wants to use for the session that is about to be established. The parameters specify such things as whether chained or unchained messages will be sent, what kinds of responses will be requested, which logical unit will start and end brackets, and so on.

Logon Exit Routine (4)

In ACF/VTAM, the SSCP notifies the application program that the logon has been received by scheduling execution of the program's LOGON exit routine. The LOGON exit routine then either accepts or rejects the logon. During processing of the logon, the application program determines whether the session parameters suggested by the logical unit are the right ones for the session or whether a different set of session parameters should be used.

Opening the LU-LU Session (OPNDST Macro Instruction) (5)

If the application program decides that it wants to go into session with the logical unit, the application program issues an OPNDST macro instruction. As a result of the OPNDST macro instruction, ACF/VTAM builds a Bind command and sends it to the logical unit. If the application program decides to reject the request for a session, it issues a CLSDST macro instruction, and the session is not established.

The Bind Command (6)

The Bind command is the key item in establishing the LU-LU session. Besides indicating the application program's willingness to go into session, the Bind command contains the session parameters that the program decided should be used for the session (the parameters may be the same or different from those suggested in the logical unit's logon information). If the logical unit agrees with the session parameters and wishes to establish a session with the application program, the logical unit sends a positive response to the Bind. If the logical unit does not agree with the session parameters, it sends a negative response and the LU-LU session is not completed.

Completing the LU-LU Session (7)

When ACF/VTAM receives a positive response to the Bind command, it completes the LU-LU session and the logical units are ready to communicate. In some cases, the exchange of messages and responses cannot begin until a Start Data Traffic command is sent by the primary end of the session to the logical unit. The need for the Start Data Traffic command is determined by the transmission services profile in the session parameters.

The SNA Concept of Domains as Implemented by ACF/VTAM

A data communication network is divided into domains when the network contains more than one host computer and each host computer contains ACF/VTAM (or another data communication access method that supports networking). A *domain* is that portion of a total network that is controlled by a particular ACF/VTAM (or another access method). Figure 1-4 shows a network with two domains. The elements in domain A are controlled by the ACF/VTAM in host computer 1; the elements in domain B are controlled by the ACF/VTAM in host computer 2.

The domains are joined either (1) by a cross-domain link between two local communications controllers (as shown in Figure 1-4) or (2) by a communications controller that is channel-attached to more than one host computer.

When a network contains more than one domain, an ACF/VTAM application program in one domain can communicate with logical units in its own domain and in other domains. The application program can communicate across domain boundaries with SNA terminals and SNA logical units, with ACF/VTAM or ACF/TCAM application programs, and with BSC 3270 terminals that were defined with PU=YES. The application program cannot communicate with local, start-stop, or BSC terminals (except specially designated BSC 3270 terminals) in other domains.

In cross-domain communications, neither end of the session need be aware that the other end of the session is in another domain. By using a symbolic name to refer to the resource in another domain, each end of the session provides enough information for the resource to be identified and located. All addressing and routing of messages between the domains are handled automatically by ACF/VTAM and the network control programs (NCPs) in the local communications controllers.

The Major Programming Elements in an ACF/VTAM Application Program

Figure 1-5 shows the three major functions that any ACF/VTAM application program performs:

- Opening and closing the program (associating the program with and disassociating it from ACF/VTAM).
- Connecting to logical units that have already been defined to ACF/VTAM and made active either when ACF/VTAM was started or when the network operator issued a command. Disconnecting the logical units when communication is no longer necessary or possible.
- Communicating with logical units to which the program is connected.

Figure 1-6 shows these major functions in more detail in the approximate order in which the functions occur. Although every ACF/VTAM facility is not shown in Figure 1-6, the facilities that are shown give a general idea of an ACF/VTAM application program. The headings below correspond to the numbers in Figure 1-6.

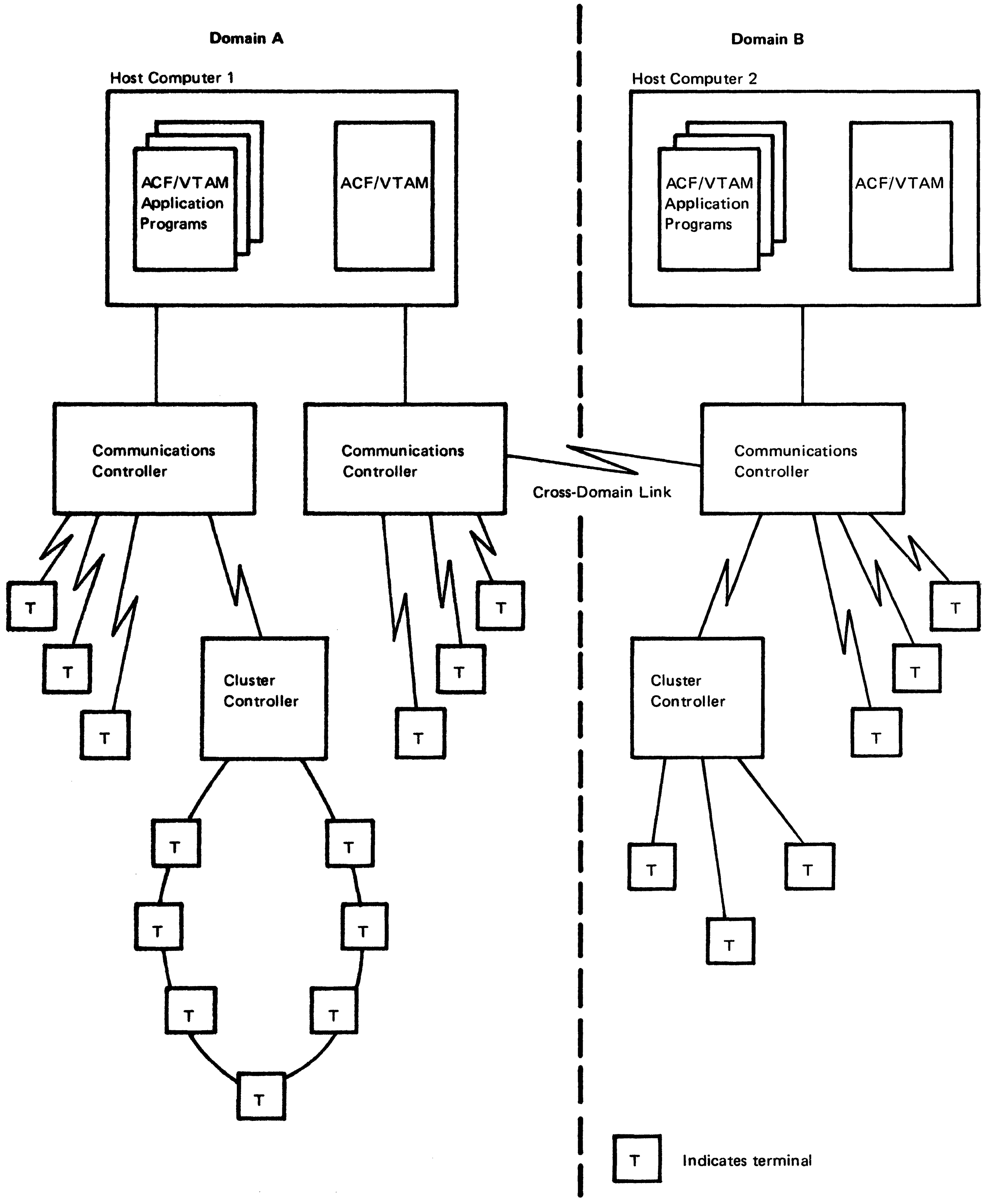


Figure 1-4. A Data Communication Network with Two Domains

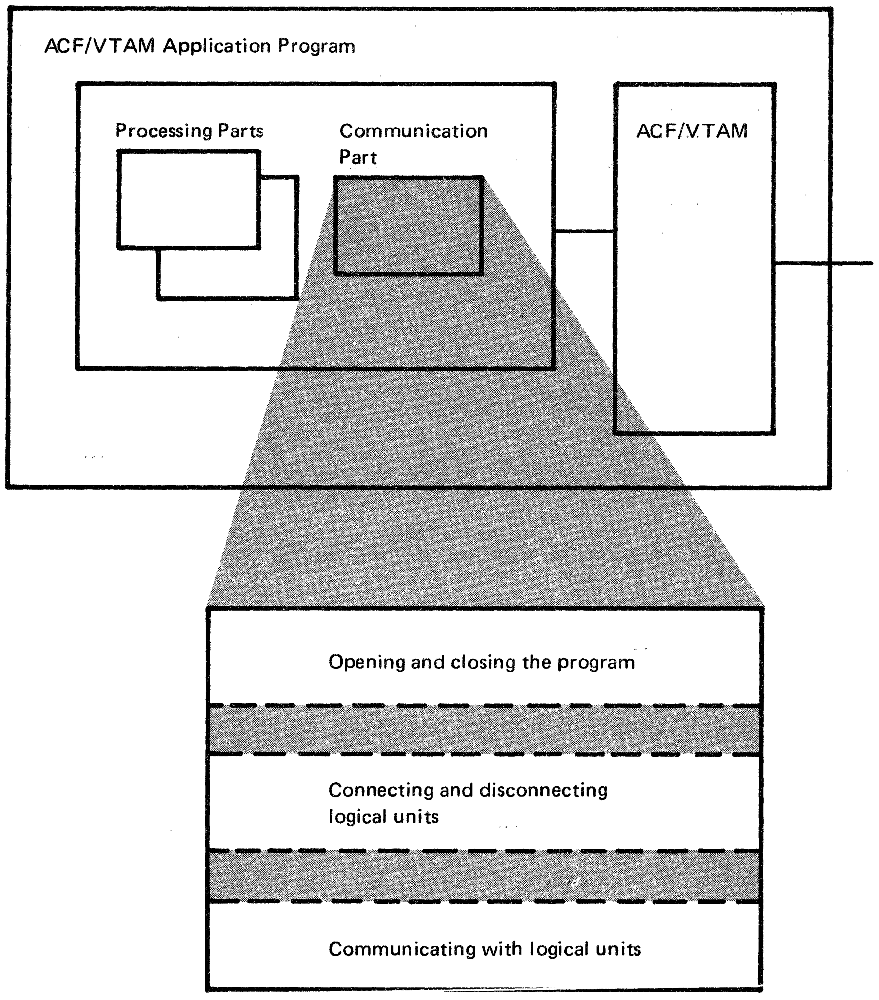

ACF/VTAM Application Program
Processing Parts
Communication Part
ACF/VTAM
Opening and closing the program
Connecting and disconnecting logical units
Communicating with logical units

Figure 1-5. The Major Functions of the Communication Part of an ACF/VTAM Application Program

Opening the Program ①

Assume that a program has been started. The program issues an OPEN macro instruction to open an access method control block (ACB). The ACB, in the constants area of the program, enables ACF/VTAM to relate the program to the name of the APPL statement that was used to define the application program to ACF/VTAM.

Connecting a Logical Unit ②

A common way to connect a logical unit is to have the logical unit send a logon, which requests connection to a particular program. An ACF/VTAM application program can have a LOGON exit routine that is automatically entered when a logon is received. The LOGON exit routine connects the logical unit by issuing an OPNDST macro instruction. The OPNDST points to a node initialization block (NIB). The NIB contains information that ACF/VTAM associates with the logical unit during its connection. When the OPNDST is completed, the program and the logical unit can exchange messages.

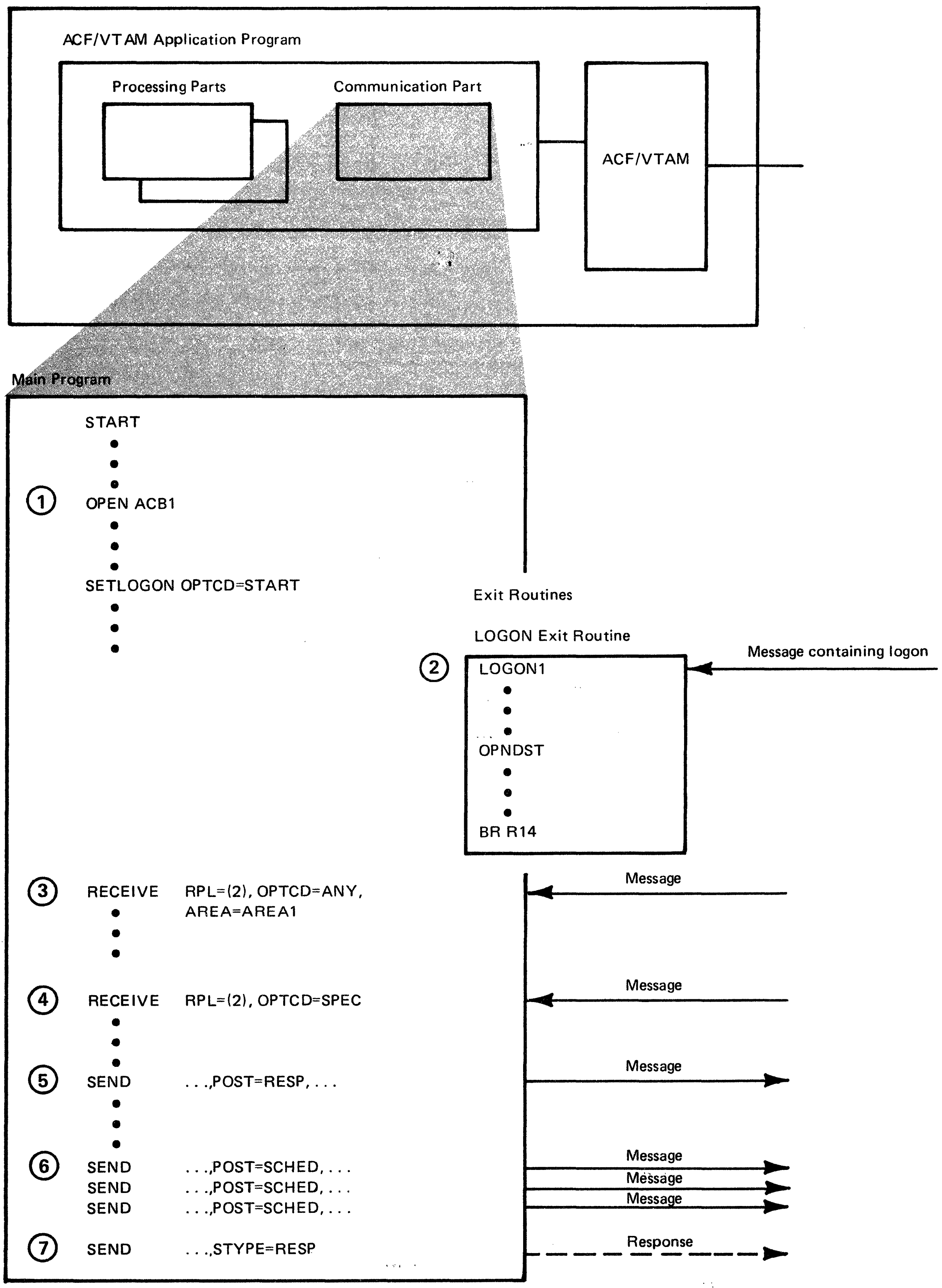


Figure 1-6 (Part 1 of 2). Major Programming Elements in the Communication Part of an ACF/VTAM Application Program

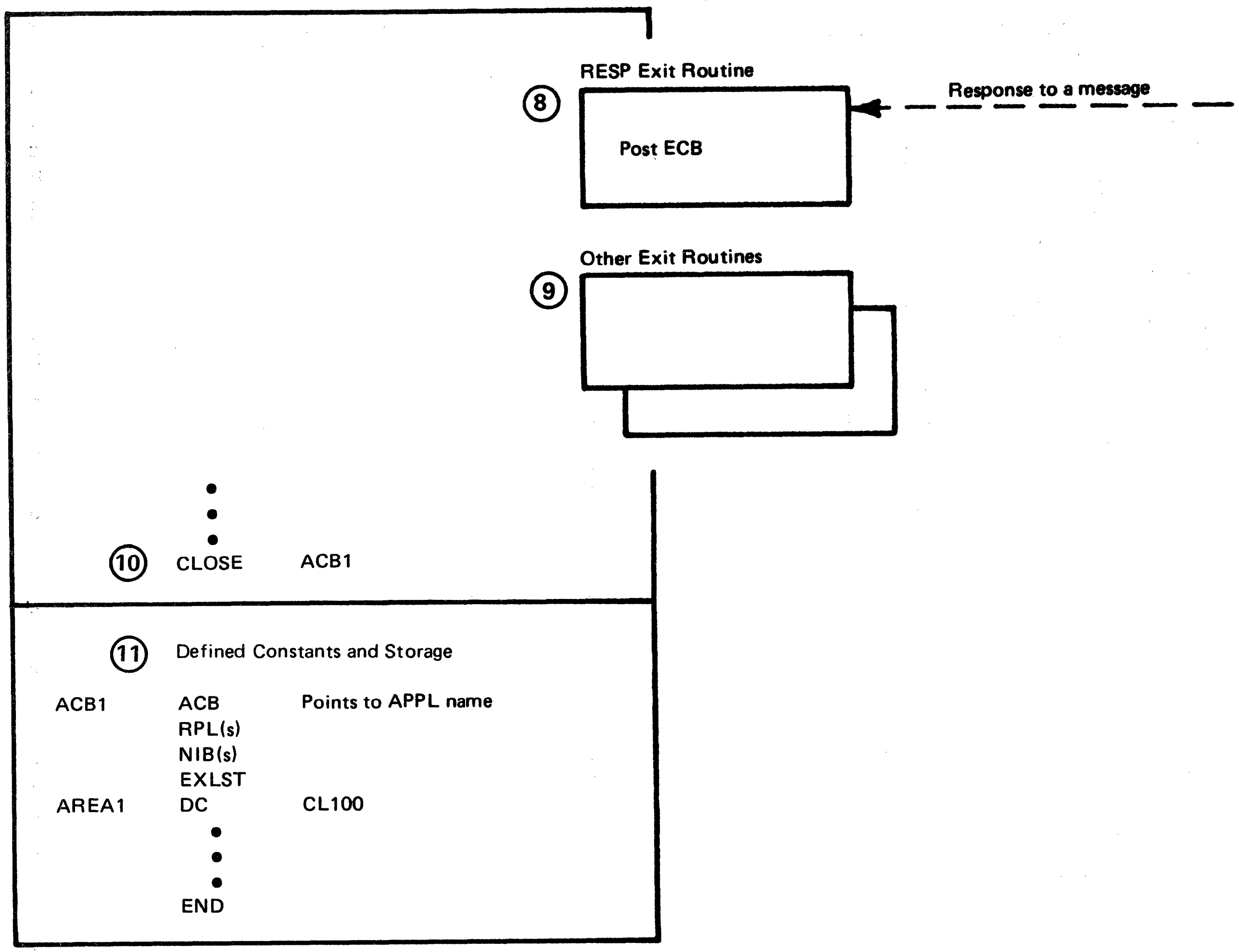


Figure 1-6 (Part 2 of 2). Major Programming Elements in the Communication Part of an ACF/VTAM Application Program

Receiving a Message from Any Logical Unit (3)

After one or more connections have been made, a request can be issued to receive input from a *specific* connected logical unit or from *any* connected logical unit. To receive a message sent from any connected logical unit, a RECEIVE with OPTCD=ANY is issued. Such a RECEIVE is completed if ACF/VTAM is already holding a message received from any connected logical unit or is completed when such a message is received. (Note that a message from a logical unit that has been switched to specific mode is not eligible to complete a RECEIVE with OTPCD=ANY.) As part of the RECEIVE operation, the data in the message is moved from ACF/VTAM to a designated area in the application program—for example, to AREA1 shown in the constants area. Note also that there is a separate and independent CA-CS setting for each type of input (DFSYN, DFASY, and RESP).

The RECEIVE and most other ACF/VTAM application program requests must furnish the address of a request parameter list (RPL), shown in the constants area of Figure 1-6. Fields in the RPL contain parameters that tell ACF/VTAM exactly how to perform the requested operation. On completion of a requested operation, ACF/VTAM places feedback information in the RPL, where it can be checked by the application program.

Each message that is received or sent by the ACF/VTAM application program contains either or both of two kinds of information: data and commands. Data is information that is meaningful only to the processing portion of an application program. Commands are special signals that help direct the further exchange of messages between the application program and the logical unit. Whereas data is received in or sent from a defined input or output area in the program, commands are received in or sent from certain fields of the RPL that is specified in a RECEIVE or SEND macro instruction. (The term *commands* is used here and in other parts of this manual to cover SNA commands and indicators used for data-flow control and session control.)

Receiving a Message from a Specific Logical Unit (4)

An input request can also be issued in such a way that only a message from a specific logical unit will satisfy the request. To do this, a program issues a RECEIVE with OPTCD=SPEC and with the RPL indicating the logical unit from which the input is desired. A common practice is for a program to issue a RECEIVE with OPTCD=ANY to accept input from any logical unit. Then, when a message is received from a logical unit, the program specifies that the logical unit is to be switched into a mode (called *specific mode*) in which a message from it cannot satisfy a RECEIVE with OPTCD=ANY and can only complete a RECEIVE with OPTCD=SPEC. When this mechanism is used, a conversation with a logical unit consists of an initial RECEIVE with OPTCD=ANY followed by a series of RECEIVEs with OPTCD=SPEC and SEND macro instructions. (Note that a SEND macro instruction always specifies that the message is to be transmitted to a specific logical unit.) When the conversation is completed, the application program can switch the logical unit back to the ANY mode.

Completion of the RECEIVE with OPTCD=ANY might be followed by execution of a processing routine of the program and, subsequently, by the processing part of the program's calling the communication part with individual requests for input and output. The communication part issues the RECEIVE with OPTCD=SPEC at **4** or one or more SENDs at **5** or **6**.

As implied by the preceding description, a RECEIVE is not completed until ACF/VTAM receives a message from a logical unit and passes it to the application program.

Sending a Message (5)

In contrast to a RECEIVE macro instruction, a SEND macro instruction can be completed at either of two different times: when the message is *scheduled* (that is, when ACF/VTAM has accepted the request, moved the message to its own output area, and prepared everything for the transmission) or when the message has been *responded to* (that is, after ACF/VTAM has sent the message and received a response). To specify completion upon receipt of a response, the programmer uses a SEND with POST=RESP (as at **5**), meaning that the results of the operation will be immediately available in the RPL when the request is completed. With POST=RESP, the application program cannot reuse the RPL or message output buffer associated with the SEND until the operation is completed.

Scheduling the Sending of a Message (6)

As an alternative to POST=RESP, a message can be *scheduled* for output (SEND with POST=SCHED). On completion of the SEND, the data has been accepted by ACF/VTAM and the program can reuse the RPL and the message output buffer. The program itself must determine that a scheduled message actually arrived at its destination and was processed successfully. One way is to request the logical unit to send back a special message called a *response,* which is an indication of whether and how a message arrived and was processed, successfully or unsuccessfully. A response can be requested in each SEND that specifies POST=SCHED. When the response arrives, ACF/VTAM either completes a special RECEIVE that can receive responses (not shown) or enters an ACF/VTAM application program's RESP exit routine, such as the one at **8**.

Sending a Response (7)

The SEND requests in **5** and **6** above requested the sending of a message. The program may also want to send a response to a message. This is done by specifying STYPE=RESP instead of STYPE=REQ in the SEND macro instruction and by specifying other parameters to indicate the type of response (for example, positive or negative) to be sent. A response is sent because the logical unit requested it; it is a special kind of message that is identified as a response to a particular preceding message. Each message is given a sequence number by the sender's access method (ACF/VTAM or the logical unit); the receiver puts the same sequence number in a response, thus indicating within the response which message is being responded to.

Receiving a Response (8)

When a program uses SEND with POST=SCHED macro instructions, the program can take direct action to receive each response; that is, for each response, it can issue a RECEIVE with RTYPE=RESP. More frequently, however, a program will contain a RESP exit routine (at 8), which is scheduled each time a response is received. The RESP exit routine can notify the main program of receipt of the response, perhaps by posting an ECB. The main program must then correlate the response with the send operation that produced it.

Other Exit Routines (9)

In addition to the LOGON and RESP exit routines, ACF/VTAM provides automatic scheduling of other special-purpose exit routines. LERAD and SYNAD exit routines can be coded which ACF/VTAM schedules when the program issues a macro instruction that uses an RPL (such as SEND or RECEIVE) and an error or a special condition occurs. The presence and addresses of these special-purpose exit routines are identified to ACF/VTAM in an EXLST macro instruction (in the constants area). The EXLST is pointed to by the ACB or NIB, or both.

Although not shown, ACF/VTAM also provides another general kind of exit routine, the RPL-specified exit routine. An RPL exit routine is identified in the EXIT operand of an RPL-based macro instruction (any macro instruction that uses an RPL). A different RPL exit routine can be identified in each RPL-based macro instruction, or some macro instructions can use the same exit routine. When the operation requested in the macro instruction is completed, control is automatically given to the exit routine specified in the EXIT operand. Having an RPL exit routine scheduled upon completion of a request is an alternative to having ACF/VTAM post an ECB upon completion.

Disconnecting a Logical Unit

The connection with a logical unit is terminated when the application program issues a CLSDST macro instruction. Disconnecting the logical unit allows it to be connected to another application program.

Closing the Program (10)

An application program can be closed when the program determines it should be or when the ACF/VTAM network operator requests it. To close an application program, the program issues a CLOSE macro instruction. This disconnects from the program any logical units that are still connected to it and disassociates the program itself from ACF/VTAM.

Constants and Control Blocks (11)

In addition to message buffers (data areas) required for input and output messages and in addition to other areas such as status flags for logical units, each ACF/VTAM application program must define (or generate dynamically) these control blocks:

One ACB to define several facts about the program itself.

One EXLST (list of exit routine names) if any exit routines are to be written. Although not required, certain exit routines, such as TPEND, NSEXIT, and LOSTERM are recommended.

At least one RPL for each request that can be pending concurrently with other requests.

At least one NIB for each logical unit that must be in the connection process concurrently. It is possible to use only one NIB if logical units are connected one at a time.

Manipulative Macro Instructions

ACF/VTAM provides macro instructions that allow control blocks to be created and initialized during program execution (the GENCB macro instruction) and that allow the control block fields to be changed and tested (the MODCB, SHOWCB, and TESTCB macro instructions). In addition, macro instructions are provided that generate DSECTs for the control blocks. This allows control block fields to be located and tested with assembler language instructions. ACF/VTAM application program macro instructions and control blocks are discussed in more detail in Chapter 2.

Chapter 2. The ACF/VTAM Language

The ACF/VTAM application program uses ACF/VTAM macro instructions to request the operations discussed in Chapter 1. ACF/VTAM provides assembler language macro instructions to:

Associate an application program with or disassociate it from ACF/VTAM

Connect the program to or disconnect it from specific logical units

Communicate with logical units

Control an ACF/VTAM network

Build and initialize control blocks used when requesting connection, communication, or other services from ACF/VTAM

Manipulate a control block; for example, to test the value of a field in a control block

Characteristics of the ACF/VTAM Language

These are some characteristics of the ACF/VTAM language:

Keyword operands: The operands in ACF/VTAM macro instructions, with the exception of OPEN and CLOSE, are keyword operands rather than positional operands. Keyword operands make the coding easier to read. The keywords themselves identify control block fields. Some keyword operands must be specified, but most are optional.

Manipulative macro instructions: These macro instructions provide an easy way to gain access to particular control block fields, usually to test or display values after a requested operation. Fields are specified symbolically; field displacements do not have to be known.

Exit routines: The ACF/VTAM application program can specify that ACF/VTAM is to automatically schedule special-purpose exit routines. These routines are written to handle conditions such as receiving a request for connection from a logical unit or receiving a certain type of command from a logical unit. In addition, an ACF/VTAM application program can request that ACF/VTAM complete a particular request by scheduling an RPL-specified exit routine instead of by posting an ECB. RPL exit routines provide additional programming flexibility and convenience and give greater priority to the handling of an event's completion than does the posting of an ECB.

Complements VSAM: In general, the ACF/VTAM language complements the VSAM language. Both ACF/VTAM and VSAM use ACB, EXLST, and RPL control blocks (although the formats of these control blocks differ in the two access methods). Both ACF/VTAM and VSAM have macro instructions (GENCB, MODCB, TESTCB, and SHOWCB) that are used to manipulate these control blocks, and both provide the ability to code and specify the scheduling of exit routines.

A Summary of Macro Instructions

This section summarizes the ACF/VTAM application program macro instructions. For a complete description of each macro instruction, see *ACF/VTAM Macro Language Reference.*

The Connection Macro Instructions

These macro instructions tell ACF/VTAM that a particular ACF/VTAM application program is in operation and, subsequently, request ACF/VTAM to connect the application to one or more logical units. The macro instructions also request ACF/VTAM to disconnect the program from one or more logical units and to disconnect the program from the ACF/VTAM system.

OPEN: Identifies an application program to ACF/VTAM. Once the program is identified, ACF/VTAM can schedule exit routines associated with the program.

CLOSE: Indicates to ACF/VTAM that an application program is terminating its association with ACF/VTAM.

OPNDST: Requests ACF/VTAM to connect the application program to a designated logical unit or to a list of logical units. Connection must be made before communication macro instructions can be used to transfer data to or from the logical unit or units.

CLSDST: Requests ACF/VTAM to terminate the connection between the application program and a designated logical unit.

REQSESS: Requests ACF/VTAM to inform a primary application program that a secondary application program wants to start a session with the primary program.

OPNSEC: Informs ACF/VTAM that the secondary application program is satisfied with the session parameters transmitted to it in a Bind command and that ACF/VTAM should complete the connection to a primary application program.

TERMSESS: Requests ACF/VTAM to terminate the session (unconditional termination) or inform a primary application program that a secondary application program wants to end the session.

The Communication Macro Instructions

ACF/VTAM provides two types of communication macro instructions: *basic mode* and *record mode.* In general, basic-mode macro instructions are used to communicate with non-SNA devices; record-mode macro instructions are used to communicate with SNA devices. A program can communicate with local non-SNA 3270 and BSC 3270 terminals in either mode, but to communicate with these terminals in basic mode, the terminals must have been defined with PU=NO. This section describes the record-mode macro instructions. Basic-mode macro instructions are described in Appendix A. Here are brief descriptions of the record-mode communication macro instructions:

RECEIVE: Requests ACF/VTAM to transfer a message, command, or response, when received from a specific logical unit or any one of a group of logical units, to the application program's data area (if the input is data) and/or to appropriate fields of the RPL (if the input is command or response information).

SEND: Requests ACF/VTAM to transmit a message, command, or response to a specific logical unit. Data in a message is transferred from an output area in the application program; commands in a message and responses to messages are specified symbolically in the SEND macro instruction.

SESSIONC: Used by a primary application program to request ACF/VTAM to send to a logical unit commands that either (1) start or stop the exchanging of messages and responses with the SEND and RECEIVE macro instructions, (2) clear out all pending messages and responses for that session, or (3) assist in synchronizing message sequence numbers. Used by a secondary application program to (1) request the primary application program to begin message recovery action, (2) send a negative response to a connection request, (3) send a response to a sequence number request, and (4) send a response to a request to start (or resume) message and response exchange with SEND and RECEIVE macro instructions.

RESETSR: Changes the mode of receiving input from a particular logical unit. The modes are *continue-any mode* (have input from the logical unit satisfy an outstanding RECEIVE that will accept input from any logical unit) and *continue-specific mode* (have input satisfy an outstanding RECEIVE that specifies only that particular logical unit). RESETSR can also be used to cancel outstanding requests for input from the specified logical unit.

Network Control Macro Instructions

These macro instructions allow an authorized application program to issue ACF/VTAM network operator commands (except START and HALT) and the OS/VS REPLY command and to receive network operator messages from ACF/VTAM:

SENDCMD: Enters an ACF/VTAM network operator command or the OS/VS REPLY command from an authorized application program, (called a *program operator*). All network operator commands can be entered except START and HALT.

RCVCMD: Receives an unsolicited network operator message, or receives replies to commands that were issued by a program operator.

Sample programs and the use of these macro instructions are presented in the *ACF/VTAM Program Operator Guide,* SC38-0257.

Note: *The REPLY command is used for communication between ACF/VTAM and the application program. For DOS/VS, ACF/VTAM supports the REPLY command (in the OS/VS format) only in response to a request from ACF/VTAM.*

The Control Block Macro Instructions

These macro instructions are used to build and initialize ACF/VTAM application program control blocks and manipulate these control blocks. The ACF/VTAM application program control blocks are the:

Access method control block (ACB)

Exit list (EXLST)

Node initialization block (NIB)

Request parameter list (RPL)

Each ACF/VTAM application program can have one or more of each type of control block. Ordinarily, a program will have one ACB, one or several exit lists, a number of NIBs, and a number of RPLs.

The Control- Block-Building Macro Instructions

The control blocks are built and initialized by coding a macro instruction for each control block. The operation codes of the macro instructions are identical to the names of the control blocks that they build and initialize. These are the control-block-building macro instructions:

ACB: Builds and initializes an ACB. An ACB contains information the application program provides ACF/VTAM about the application program in its entirety. Primarily, it names the application program and the list of exit routines associated with the program. The ACB contains information about the *application program.*

EXLST: Builds and initializes an exit list. An exit list contains the addresses of special exit routines that ACF/VTAM is to schedule when certain conditions occur (as, for example, when a logon is received from a logical unit). The EXLST contains the names of *exit routines.*

NIB: Builds and initializes a NIB. A NIB contains information the application program provides ACF/VTAM about general communication characteristics that are to exist between the application program and a particular logical unit. This information is provided to ACF/VTAM as part of a connection request; it remains in effect for the duration of a connection. The NIB contains information about a *logical unit.*

RPL: Builds and initializes an RPL. An RPL contains information (parameters) that an application program provides ACF/VTAM when requesting connection, communication, or other RPL-based action. On completion of the requested action, the RPL contains information that ACF/VTAM has put there for the application program. The RPL contains information about a *request.*

An ACB, EXLST, NIB, or RPL control block can be assembled in the application program by using the appropriate control-block-building macro instruction described above, or the control block can be created and initialized during program execution by using the GENCB macro instruction described below.

The Control-Block Manipulating Macro Instructions

ACF/VTAM provides a group of macro instructions that manipulate control-block fields. These macro instructions provide a more convenient way to do this than by using assembler-language instructions. They refer to fields symbolically rather than by specific control-block location. By using these macro instructions rather than the control-block-building macro instructions listed above, a program can be written to be unaffected by control-block changes in future releases of ACF/VTAM. In addition, the macro instructions may be used to code a reenterable application program. The manipulative macro instructions are:

GENCB: Builds an ACB, EXLST, NIB, or RPL during program execution and can initialize designated fields with specified values. Multiple copies of one control block can be built with one GENCB macro instruction.

SHOWCB: Obtains the value or values from one or more fields of a control block and places them in an area in the application program where they can be examined. In addition to fields that are set by the application program's use of macro instruction keyword operands, a number of control block fields can be shown that are set by ACF/VTAM but that cannot be directly modified by the application program.

TESTCB: Tests the contents of a field against a value and sets the condition code in the program status word (PSW).

MODCB: Changes the contents of one or more fields by inserting specified values in the fields.

There are several different forms of the manipulative macro instructions. In addition to the standard form, there is a list form, a remote list form, a generate form, and an execute form. The nonstandard forms can be used for programs that must be reenterable or that are sharing with other programs the parameter lists that are assembled when the macro instructions are expanded.

Rather than using the manipulative macro instructions, the program can include IBM-supplied macro instructions that generate DSECTs for each kind of control block. Each DSECT shows the field names and possible values that each field can contain. These names and values can be used in assembler instructions to set and test designated fields.

Supporting Macro Instructions

These additional macro instructions are provided:

CHECK: Checks and, if necessary, awaits completion of a previously requested RPL-based operation; marks as inactive the RPL associated with the request (thus freeing it for further use); and, if a logical or other error or special condition is detected and a LERAD or SYNAD exit routine exists, causes the appropriate routine to be entered.

EXECRPL: Reissues a specified request. One use of this macro instruction is to reexecute a request without changing any field in the RPL. This is done, for example, in a SYNAD exit routine when the return code from the first attempt to perform the operation indicates that a retry is possible (return code 8 in register 0). Another use of the EXECRPL macro instruction is to request that the operation be performed again and to specify that, before the operation is retried, one or more fields in the RPL are to be changed or to be reset to their original values.

The EXECRPL can also be used in place of an RPL-based macro instruction, such as OPNDST, SEND, or RECEIVE. Prior to issuing an EXECRPL, the operation to be performed must be set in the RPL; this requires the use of the IBM-supplied RPL DSECT. Other parameters may either be set in the RPL or specified with keyword operands when the EXECRPL macro is issued. While less convenient to code, using EXECRPL results in execution of fewer instructions.

Note that EXECRPL cannot be used to issue or reissue a CHECK request that has failed, since CHECK does not alter the operation field of the RPL.

INQUIRE: Obtains certain information that the application program may need and places it in a specified area of the program. The information that can be requested using INQUIRE includes: the user logon message associated with a logon; the session parameters associated with a particular logon mode name or with a logical unit that is logging on; the number of logical units currently connected to, or queued for, the application program; and whether another application program is active or inactive and whether it is accepting logons.

INTRPRET: Provides a means of gaining access to a user-defined table. For example, INTRPRET can be used to obtain the real symbolic name of an application program when the program is identified with an alias in a logon message. INTRPRET can be used by special programs written to receive logon messages and then reconnect logical units to the appropriate application program.

SETLOGON: Used by a primary application program to tell ACF/VTAM to begin queuing and scheduling logons for an application program's LOGON exit routine. The user can also temporarily halt the queuing of logons until more logical units can be handled or can permanently halt the queuing of logons in preparation for a close-of-day operation. Used by a secondary application program to enable itself to issue REQSESS macro instructions and enable its SCIP exit routine to receive session parameters.

SIMLOGON: Allows the applicaton program itself to initiate a logon on behalf of one or more logical units to which the program is to be connected.

How the Executable Macro Instructions and the Control Blocks Are Related

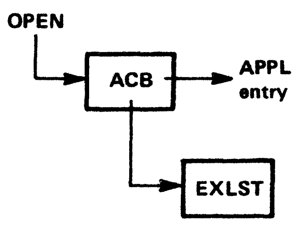
The relationship of the ACF/VTAM control blocks to each other and to the macro instructions that refer to them can be described in the context in which they are used. To establish that context, the following sections describe the relationships and use of the control blocks in terms of the operations that every application program must perform:

Opening the application program—that is, identifying itself to ACF/VTAM as operational

Connecting to logical units with which it will communicate

Communicating with connected logical units

Opening the Application Program

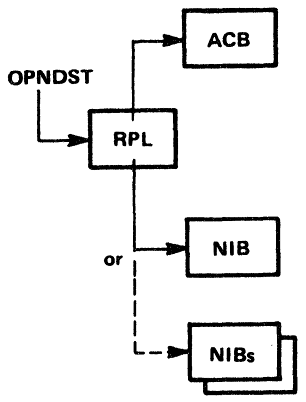


The OPEN macro instruction associates an active application program with ACF/VTAM so the application program can use ACF/VTAM facilities. The OPEN macro specifies an ACB; the ACB in turn points to a location in the program that contains the name of the application program as defined in an APPL statement during ACF/VTAM definition. The ACB may also point to an EXLST control block containing the names of exit routines that are to be associated with the application program. (An EXLST can also be pointed to when a logical unit is connected; see "Connecting Logical Units," below.) When the open process is completed, any exit routines that have been specified are eligible for scheduling by ACF/VTAM.

More than one ACB can be opened by a single OPEN macro instruction. This means that a program that performs related functions (for example, communicating with both logical units and terminals) may be defined so that it is viewed by ACF/VTAM as more than one application program. Many ACF/VTAM users will find it satisfactory to open only one ACB for each program.

The CLOSE macro instruction notifies ACF/VTAM that an application program is detaching itself from ACF/VTAM. As a result of a CLOSE macro, any logical units still connected to the program are disconnected.

Connecting Logical Units



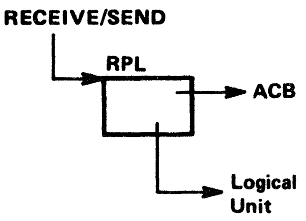
Before communicating with a logical unit, an application program must be connected to the logical unit. Connection can be initiated by the logical unit, the network operator, ACF/VTAM, or an application program. But, regardless of who initiates the connection request, it is the application program that will adhere to primary protocols that formally completes connection by issuing an OPNDST macro instruction. The OPNDST macro instruction specifies an RPL that is associated with the request. The RPL contains the address of a NIB. The NIB contains information that applies to subsequent communication with the logical unit. If necessary, the address of a unique storage area to be associated with the logical unit can be specified in the NIB. This area could include an I/O area and a place for flags that keep track of communication with the logical unit. If a number of logical units are to be connected by an application program, a single SIMLOGON or OPNDST with OPTCD=ACQUIRE can be used, and the RPL points to a list of NIBs instead of to a single NIB.

Optionally, for certain types of exit routines (DFASY, RESP, and SCIP), a NIB can point to a list of exit-routine names in an EXLST control block. For the logical unit being connected, these exit routines are used in preference to the corresponding exit routines identified for the entire application program when the ACB was opened.

When a logical unit is connected as the result of an OPNDST macro instruction, ACF/VTAM returns information about the logical unit in the RPL and the NIB. In both the RPL and the NIB, ACF/VTAM places a communication identifier (CID) that it has assigned to the session with the logical unit. On all subsequent I/O requests for the logical unit, the application program must be sure that this CID is present in the RPL. In addition to the CID, ACF/VTAM also places the logical unit name (for an OPNDST ACCEPT ANY) and other information in the NIB; if desired, the application program can use this information to determine how to communicate with the logical unit.

Once a NIB has been used to connect one logical unit, it can be reinitialized and reused to connect another logical unit.

Communicating with Logical Units



Having opened the application program's ACB and having connected one or more logical units to the program, the program can communicate with each connected logical unit by issuing SEND and RECEIVE macro instructions. ACF/VTAM obtains the name of the application program that made the request and the identity of the logical unit (if a specific logical unit is being addressed) from the RPL. The communication macro instruction specifies an RPL; the RPL contains the address of an ACB and the identity of the logical unit.

The SEND and RECEIVE macro instructions write and read messages and responses. A message contains data and/or control commands and indicators. A response contains information that tells whether a message requiring a response arrived and was processed successfully or unsuccessfully. Certain messages, called *normal-flow messages*, are received or sent in sequence with other serially queued messages; other messages, called *expedited-flow messages*, are received or sent immediately, ahead of other queued messages.

Only data is written from or read into an application program data area. Control commands and responses are sent by being specified symbolically in a SEND macro instruction or its associated RPL. Commands and responses that are received are not read into a data area but are detected by analyzing fields in the RPL associated with a RECEIVE macro instruction or in an RPL associated with the scheduling of a special exit routine that handles the receipt of commands or responses.

Disconnecting Logical Units

Once a series of communications between the application program and a logical unit is completed, the program disconnects the logical unit by issuing the CLSDST macro instruction. If the program is terminating and all logical units are to be disconnected at the same time, the program can issue a single CLOSE macro instruction, which closes the ACB, instead of issuing a series of CLSDST macro instructions for the logical units. As a result of the CLOSE macro instruction, ACF/VTAM issues a CLSDST macro instruction for each logical unit. Although it requires more coding, issuance of separate CLSDST macro instructions in the application program may result in faster execution than using the CLOSE macro instruction and having ACF/VTAM disconnect the units.

Exit Routines

ACF/VTAM allows use of exit routines by which an ACF/VTAM application program can gain control to handle certain conditions. An exit routine is written to handle a specific event (for example, a SYNAD routine which is written to process RPL-based errors or special conditions other than logical errors). When the event occurs, ACF/VTAM gives the exit routine control as soon as possible. With the exception of SYNAD and LERAD, exit routines need not be reenterable, since only one exit routine will be invoked at a time. If multitasking is used and each task opens an ACB, more than one exit routine can be invoked at the same time (one for each task that opened an ACB). The exit routines still need not be reenterable as long as they are not shared between the tasks (that is, as long as two tasks do not open ACBs that use the same exit routines). If an exit routine is shared in this way, it must be reenterable.

There are two kinds of exit routines:

Exit-list exit routines: These are special-purpose exit routines that ACF/VTAM schedules when an event they are designed to handle occurs, such as receipt of a logon. The exit-routine addresses (entry points) are specified in an exit list created with the EXLST macro instruction. A program can have more than one exit list. An exit list (a set of exit routines defined with the EXLST macro) can be specified in an ACB and thus be used by ACF/VTAM when an exit-routine event occurs for any logical unit connected to the program or, for certain exit routines—DFASY, RESP, and SCIP—an exit list can be specified in a NIB and be used by ACF/VTAM only when an exit-routine event occurs for the logical unit associated with the NIB.

RPL-specified exit routines: These are exit routines that contain instructions to be executed when particular requests are completed. In any individual connection, communication, or other RPL-based request, if an RPL exit-routine address is specified, the exit routine is scheduled as an alternative to ACF/VTAM's posting an ECB when the requested action is completed. A program can use a mixture of ECB-posting and RPL exit routines, or it can use all one or the other.

The names of the special-purpose exit routines and the events that cause them to be entered are summarized in Figure 2-1.

The use of exit routines is described in detail in Chapter 7 and illustrated in the sample programs in Part 3 and Appendixes D and F.

Register Conventions

These general register conventions apply in writing an ACF/VTAM application program:

- Before issuing an executable macro instruction (such as SEND), the address of an 18-word save area must be in register 13.
- When issuing an executable macro instruction, register notation can be used. However, register notation cannot be used to initialize a value in an ACB, RPL, NIB, or EXLST macro instruction.
- Registers 2-12 (and only those registers) can be used when issuing ACF/VTAM macro instructions. The one exception is that register 1 can be used to supply an RPL address for any RPL-based macro instruction.

Exit Routine Name	Event
ATTN	A start-stop terminal has caused an attention interruption.
DFASY	An expedited-flow command has been received from a logical unit.
LERAD	A logical error has occurred following an application program request.
LOGON	A request for connection has been received from a logical unit.
LOSTERM	Connection with a terminal or logical unit has been temporarily interrupted or permanently lost; the logical unit has requested that the session be terminated; or an event has occurred that may affect future operation of the session.
NSEXIT	A network services request unit has arrived for the application program, indicating either that (1) a session with a logical unit has been broken because of a session outage, or (2) a partially complete session establishment procedure will not be completed.
RELREQ	Another application program has requested connection to a logical unit that is presently connected to this program.
RESP	A response has been received from a logical unit for which no RECEIVE with RTYPE=RESP is outstanding.
SCIP	One of the following commands has been received by the program: Clear Start Data Traffic (SDT) Request Recovery (RQR) Set and Test Sequence Numbers (STSN) Bind Unbind
SYNAD	An error other than a logical error or a special condition has occurred following an application program request.
TPEND	The network operator is shutting down the network, or an abend of VTAM has occurred.

Figure 2-1. Special-Purpose Exit Routines and the Events That Cause Them to Be Scheduled

- On regaining control after issuing a macro instruction:
  - Register 15 contains a return code.
  - Register 1 contains the address of the RPL associated with the macro instruction.

Part 2. Writing an ACF/VTAM Application Program

Chapter 3. Organizing a Program: Describes major program organization alternatives that are available when constructing a program and discusses the advantages of each alternative. The facilities related to each alternative are discussed in more detail in subsequent chapters in this part.

Chapter 4. Opening and Closing a Program: Describes how a program is opened and closed and what happens as a result of these actions.

Chapter 5. Connecting and Disconnecting Logical Units: Describes alternative techniques for connecting logical units to a program and for disconnecting them.

Chapter 6. Communicating with Logical Units: Describes the detailed concepts and language involved in exchanging messages and responses with logical units, the flow of messages in the network that results from ACF/VTAM communication macro instructions, and the facilities provided for communication, such as sequence numbers.

Chapter 7. Using Exit Routines: Describes how exit routines are used in an ACF/VTAM application program and lists the rules that must be followed in using them.

Chapter 8. Manipulating Control Blocks: Describes some ways to manipulate control blocks. It shows examples of using the manipulative macro instructions: GENCB, MODCB, SHOWCB, and TESTCB. It lists and discusses in general the macro instructions that supply DSECTs, so that ACF/VTAM application program control blocks can be manipulated using assembler instructions.

Chapter 9. Handling Errors and Special Conditions: Describes in general how to organize and code special routines to analyze and handle the occurrence of errors and special conditions.

Chapter 10. Debugging a Program: Suggests how to minimize coding errors and how to determine the location of errors in an ACF/VTAM application program that is being developed.

Chapter 3. Organizing a Program

The organization of an ACF/VTAM application program affects how much storage it will use, how well it will perform, and how easy it will be to write. Before getting into the details about writing an ACF/VTAM application program, it may be helpful to understand the decisions that are made when writing the program and some of the ways in which a program can be organized. This chapter discusses these things:

- Data processing and data communication operations should be kept in different parts of an ACF/VTAM application program (this book is concerned primarily with the data communication part of the program).
- If new data processing routines are being written, it may be possible to move some work formerly performed in the host computer to the logical units with which the ACF/VTAM application program will communicate.
- Terminals supported by BTAM or by the basic-mode macro instructions of ACF/VTAM can be handled in the same ACF/VTAM program that communicates with logical units. See Appendix A.
- Different routines may be needed within an application program to handle logical units that have different session parameters. Chapters 5 and 6 discuss this subject.
- There are two major kinds of program organization: synchronous (single-thread) and asynchronous (multithread).
- Asynchronous (multithread) facilities include the scheduling of output and the receiving of input from any logical unit. It includes the continuance of conversation with a logical unit in a mode that excludes the logical unit from being read by a request for input from any logical unit. It includes a NIB that allows a unique storage area to be associated with a logical unit.
- A number of decisions must be made in writing a program. These are summarized in Figure 3-6.
- The control blocks and work areas required for each logical unit or for a group of logical units can be obtained and controlled in a number of different ways. Some examples are shown.

Single-Thread or Multithread Operations

An ACF/VTAM application program can be described as a *single-thread program*—that is, capable of processing the request of only one logical unit at a time—or as a *multithread program*—capable of processing the requests of many logical units concurrently. These terms are only generally descriptive of how the program works; in practice, many single-thread programs may do some overlapping of processing, and many multithread programs may do some processing that momentarily ties up the program for an action on behalf of only one logical unit. In general, a single-thread program requests synchronous operations and waits until each operation is completed before continuing. A multithread program requests asynchronous operations and continues processing on behalf of other logical units while waiting for an operation for a particular logical unit to be completed.

Deciding to Use a Single-Thread Program

A single-thread program is easier to design and code than a multithread program. Sample Program 1 in Part 3 is basically a single-thread program.

A single-thread design can be used for a program that never handles more than a few logical units at a time or, if more than a few are handled, where response time is not a consideration. A more likely use of a single-thread design is for a program that does nothing more than send a continuous series of messages to a logical unit (which might in turn forward the messages to a printer or to a data base on a disk) or receive a continuous series of messages from a logical unit (perhaps from the disk associated with a logical unit) and write them to a data base on disk storage at the host computer.

Deciding to Use a Multithread Program

In general, any ACF/VTAM application program that must communicate concurrently with a number of logical units must be organized in a manner that allows multithreading. This implies the use of asynchronous operations, determining completion of operations either by having ACF/VTAM post an ECB or by having it schedule an RPL exit routine.

In a multithread program, the control blocks for each logical unit must be managed efficiently. The control blocks reflect the status of the logical unit; for example, whether it has begun a conversation, what address is to be branched to when a requested output operation is completed, or whether the logical unit has sent in a logoff message. Sample Program 2 in Part 3 shows the general logic of a multithread program.

Multitasking may be used to transfer control between the communication and data processing parts of a program. It is also possible for the same routines to be shared among what ACF/VTAM perceives as more than one ACF/VTAM application program. This arrangement can be used for communicating with two different types of logical units. Two ACBs can be defined in a program, and one kind of logical unit connected with control blocks that point to one ACB while another kind of logical unit is to be associated with the other ACB. Since ACF/VTAM sees each ACB as an application program, each type of logical unit can have separate logic associated with it, including its own exit routines and its own I/O routines. Data processing parts of the program, a wait routine, and other routines can be shared.

How a Synchronous Operation Works

In a synchronous program, operations are performed serially. A request for a synchronous operation (for example, a SEND or RECEIVE with OPTCD=SYN) means that ACF/VTAM will not return control to the next sequential instruction in the program until after the requested operation is completed. Execution of the application program is halted until ACF/VTAM determines that the operation has been completed. The program must be willing to wait for the processing of one requested operation to be completed before going on to the next. Figure 3-1 illustrates a synchronous operation. (Note that while the program is waiting for the event to be completed, an asynchronous event such as a network operator HALT command could cause the program's TPEND exit routine to be entered. Only the main line of the program is suspended while waiting for completion of a synchronous operation. The exit routines associated with the program are scheduled and executed whether or not the main program logic is awaiting completion of a synchronous operation.)

When a synchronous operation is completed, the application program must determine whether the operation was successful or unsuccessful. The program does this by testing values in registers 15 and 0 and by examining fields in the RPL used for the operation. For more information on testing return codes from RPL-based macro instructions, see Chapter 9 in this manual and see Appendix C in *ACF/VTAM Macro Language Reference.*

In general, issuance of a synchronous request within an exit routine should be avoided, because that causes all execution under the task to be suspended until the operation is completed. Neither the main program, the exit routine in which the request was issued, nor another exit routine (except LERAD, SYNAD, or TPEND) can be executed until the operation is completed.

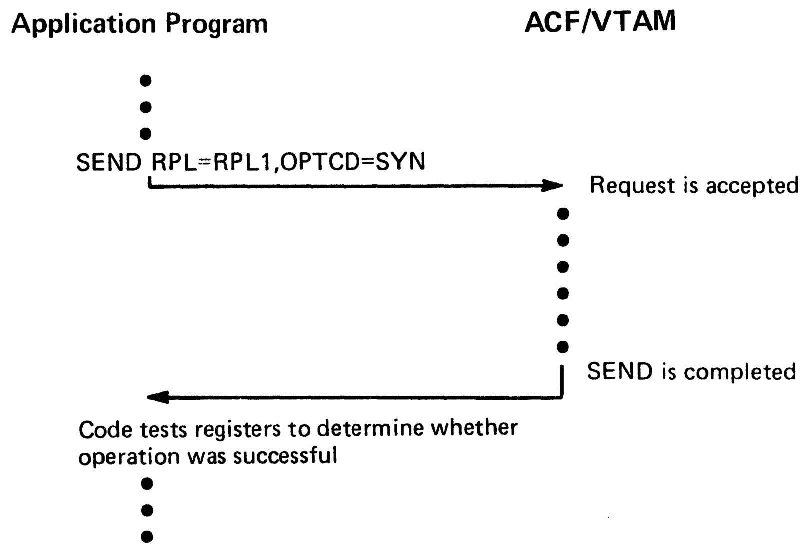


Figure 3-1. A Synchronous Operation

How an Asynchronous Operation Works

In an asynchronous operation, ACF/VTAM returns control to the next sequential instruction as soon as it has accepted the request, not when the requested operation has been completed. Accepting a request consists of screening the request for errors and scheduling the parts of ACF/VTAM that will eventually carry out the operation. While the operation is being performed, the application program is free to initiate other data-transfer operations or do other processing. For example, an application program can issue a RECEIVE macro instruction and indicate that the operation is to be handled asynchronously (OPTCD=ASY); while the input operation is being performed, the application program can begin to write to a direct-access storage device or receive input from another logical unit.

When an asynchronous operation is specified, there are two ways that ACF/VTAM can notify the application program that the requested operation has been completed. If the application program associates an event control block (ECB) with the request, ACF/VTAM posts the ECB when the operation is completed. Alternatively, the application program can designate that a particular RPL exit routine is to be executed as soon as the operation is completed. When the operation is completed, ACF/VTAM schedules the exit routine. The method of notification is controlled by the setting of the ECB operand or EXIT operand in the RPL used for the request. Figure 3-2 illustrates asynchronous processing in an application program using ECBs; Figure 3-3 illustrates the use of an RPL exit routine.

Regardless of whether a program waits on an ECB or uses an RPL exit routine, a CHECK macro instruction must be issued after an asynchronous operation to mark the RPL inactive and to make it available for another operation. The CHECK macro instruction also clears the ECB.

Using ECBs

By using ECBs, the application program can issue one WAIT macro instruction for a combination of ACF/VTAM requests and any non-ACF/VTAM requests that use ECBs.

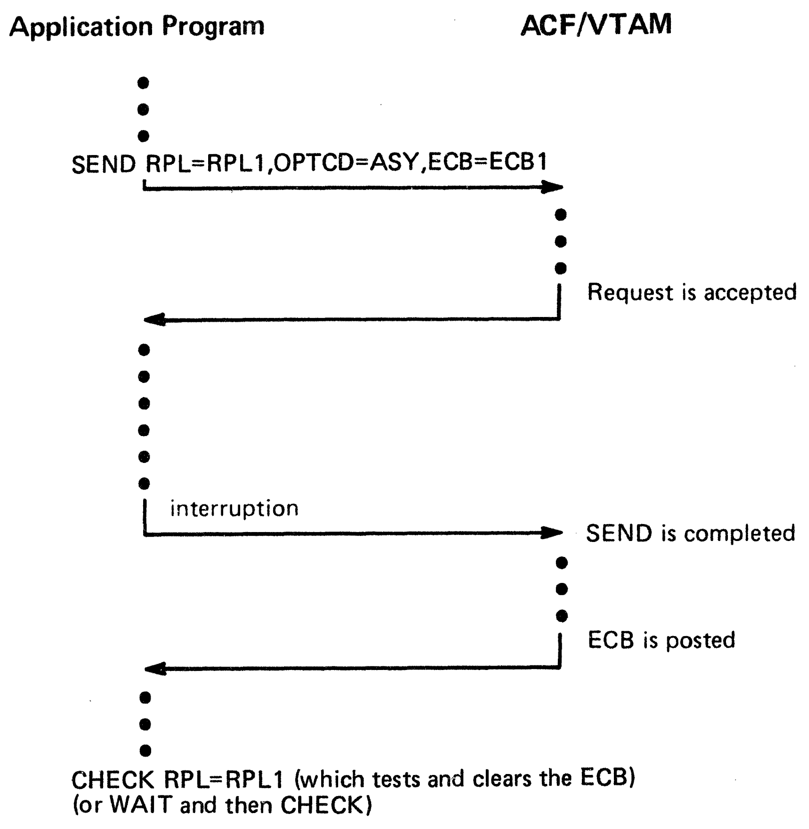


Figure 3-2. An Asynchronous Operation with an ECB Posted

For example, an application program can issue three VSAM requests and three ACF/VTAM requests; by issuing one WAIT for all six ECBs, the application program resumes processing when any one of the six operations is completed.

Using ECBs, the application program can test ECBs itself and continue to wait only if no ECB has been posted. The program can prioritize requested operations or logical units by testing some ECBs before testing others. The order of checking can be varied during program execution as circumstances change.

The distinction between ECBs and RPL exit routines rests primarily on the fact that the RPL exit routine is *automatically* scheduled when the requested operation is completed, thereby saving the application program the trouble of testing ECBs and branching to subroutines. The use of ECBs provides the program with greater control over the order in which events are to be handled.

If neither an ECB address nor an RPL exit-routine address is specified in the RPL-based macro instruction, ACF/VTAM uses the ECB-EXIT field of the RPL as an internal ECB, and ACF/VTAM (for synchronous operations) or the user (for asynchronous operations) checks and clears it. It can be set to point to an external ECB by using an RPL-based macro that specifies ECB=*ecb address.* Once set, it can be reset to an internal ECB by using an RPL-based macro instruction that specifies ECB=INTERNAL.

Using RPL Exit Routines

Instead of having ACF/VTAM post an ECB when a request for an asynchronous operation is completed, the program can have ACF/VTAM schedule and cause control to be given to an RPL-specified exit routine. The RPL exit routine can supply the logic that

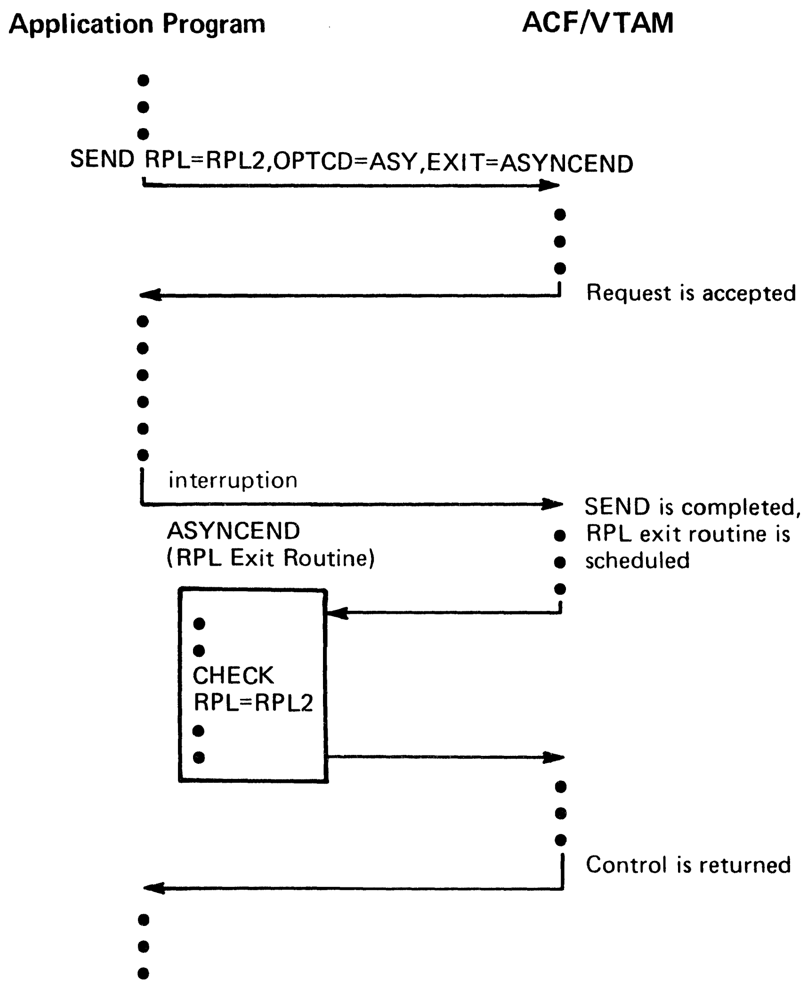


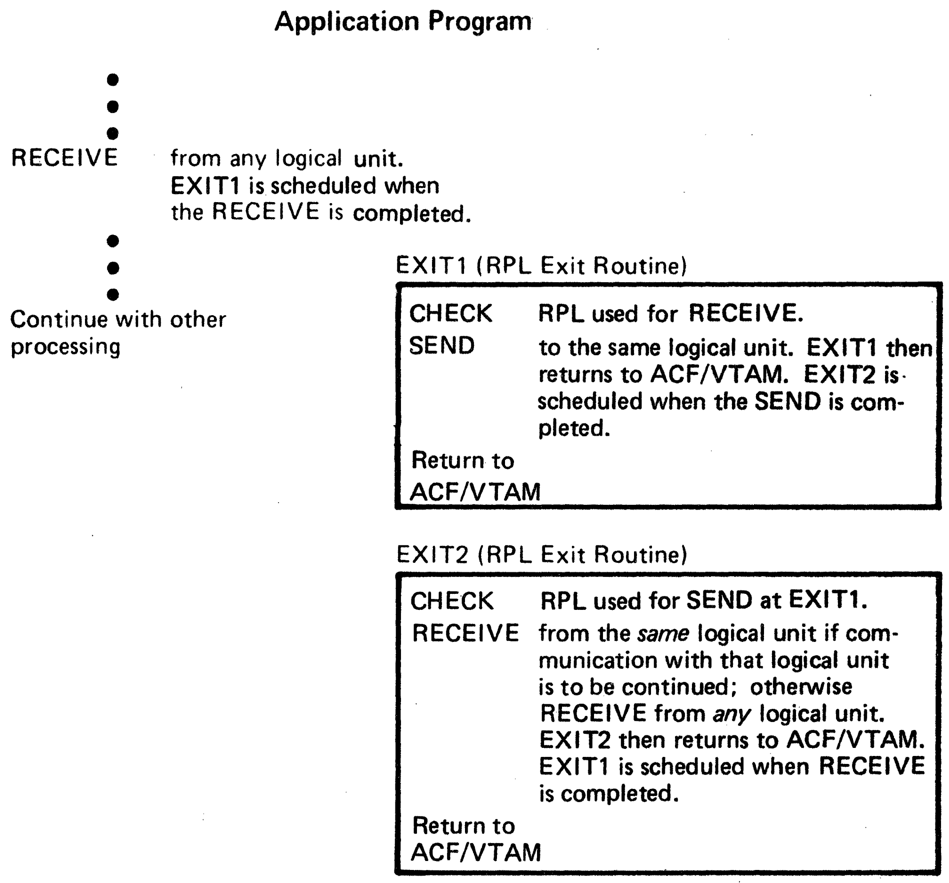
Figure 3-3. An Asynchronous Operation with an RPL Exit Routine Scheduled

would have been branched to by the main program after discovering a posted ECB. An RPL exit routine is any exit routine whose symbolic name has been provided in the EXIT operand of the macro instruction or the RPL used for the request.

One advantage to using an RPL exit routine instead of an ECB is that it is easier to code for that type of processing than it is to code the logic associated with discovering a posted ECB and relating the ECB to a branch address. The disadvantage of an RPL exit routine is that more system instructions must be executed to schedule an exit routine than must be executed to post an ECB. A program may use a combination of ECB-posting and RPL exit routines (see Sample Program 2 in Part 3).

An RPL exit routine may itself issue asynchronous requests, continue executing, and return to ACF/VTAM. The asynchronous request in an RPL exit routine may specify that upon completion of the request, an ECB is to be posted or an RPL exit routine is to be scheduled. If the RPL exit routine option is taken, the exit routine can be the same one in which the request was issued. (This is also shown in Sample Program 2 in Part 3.) Figure 3-4 shows a possible pattern of asynchronous requests within RPL exit routines.

An RPL exit routine must always eventually return control to ACF/VTAM. While one exit routine has control, no other exit routine (except LERAD, SYNAD, or TPEND) can be executed.



Application Program

●
●
●

RECEIVE from any logical unit. EXIT1 is scheduled when the RECEIVE is completed.

●
●
●

Continue with other processing

EXIT1 (RPL Exit Routine)

CHECK	RPL used for RECEIVE.
SEND	to the same logical unit. EXIT1 then returns to ACF/VTAM. EXIT2 is scheduled when the SEND is completed.
Return to ACF/VTAM	

EXIT2 (RPL Exit Routine)

CHECK	RPL used for SEND at EXIT1.
RECEIVE	from the *same* logical unit if communication with that logical unit is to be continued; otherwise RECEIVE from *any* logical unit. EXIT2 then returns to ACF/VTAM. EXIT1 is scheduled when RECEIVE is completed.
Return to ACF/VTAM	

Figure 3-4. A Possible Pattern of Requests in RPL Exit Routines for Asynchronous Operations

Advantages and Disadvantages of Different Forms of Operation

Figure 3-5 summarizes the advantages and disadvantages of synchronous operations and the two general forms of asynchronous operations, ECB-posting and RPL exit-routine scheduling.

Multithreading Facilities

In addition to the asynchronous handling of input and output requests, ACF/VTAM also provides the following facilities as aids to handling logical units in a multithread program:

A special field that can be used to associate a unique storage area with each logical unit

The ability to schedule the sending of a message

The ability to receive input from any session except those sessions that are specifically precluded

These facilities are discussed below in more detail.

Type of Request	Performance	Storage Requirements for RPLs and Data Areas	Programming Complexity
Synchronous (OPTCD=SYN)	Adequate for many batch-type (continuous input or output) programs or for programs serving few online logical units; poor for programs serving many active online logical units.	Small, since only one request can be outstanding at a time; can reuse RPL and data areas.	Simplest to program.
Asynchronous (OPTCD=ASY) • ECB-posting (ECB=address or INTERNAL)	Best; better than RPL exit routine which requires that more system instructions be executed than does posting of ECB.	May require more storage since many pending requests may be outstanding, tying up RPLs and data areas.	Most complex.
• RPL exit-routine scheduling (EXIT= address)	Not as good as ECB-posting. Some advantages if used to give priority of handling to a logical unit (for example, first input after logon).	About the same as ECB-posting.	Less complex than ECB-posting.

Figure 3-5. Relative Advantages of Synchronous and Asynchronous Requests

The USERFLD Field of the NIB

In handling a series of input and output actions with a particular logical unit, the program may need some way of associating a particular piece of information with the logical unit. For example, the program might need to know:

Which city the logical unit is located in

What type of logical unit this particular one is

Which symbolic name this program is using for the logical unit

Which storage area contains the input buffer and other application-program-manipulated control information about operations with this logical unit

When such information is static in nature (for example, the city in which the logical unit is located), the information can be assembled into the program and always be available. More frequently, however, the information that the program wants to associate with the logical unit is not available until after the program starts execution or changes during program execution. For these dynamic types of information, the program needs a mechanism for associating the desired piece of information with the logical unit.

The mechanism provided by ACF/VTAM involves the USERFLD field of the NIB, which contains space for 4 bytes of information. Whatever information is in the USERFLD field of the NIB at the time the logical unit is connected is saved by ACF/VTAM, and whenever input is subsequently received from the logical unit, that information is provided in the USER field of the RPL used for the operation. This mechanism has many uses, including those described in the following paragraphs:

Identifying the logical unit from which input has been received. Each time the application program connects a logical unit, the program puts its own version of the symbolic name of the unit into the USERFLD field of the NIB before the OPNDST macro instruction is issued. Later, the program issues a RECEIVE OPTCD=ANY, which will accept an input message from any connected logical unit. When an input message is received, the program examines the USER field of the RPL to determine the logical unit from which the input message came.

Associating a storage area with a logical unit. For each logical unit, the application program may want to have a logical-unit-associated storage area that contains an RPL, possibly an ECB (if ECB-posting is used), a data area (to be used as a buffer for input and output messages), and a status information area. When the program connects a logical unit, it specifies in the USERFLD field of the NIB the address of the storage area it wants to be associated with that logical unit. ACF/VTAM saves this address and, when input is received from the logical unit, ACF/VTAM places the address in the USER field of the RPL. The program can use the USER field address to process the input rather than having to first identify the logical unit.

Scheduling Output

ACF/VTAM allows a program to request that a message be scheduled for sending and that the operation be considered complete as soon as the message has been *scheduled* for output rather than actually *sent with arrival confirmed by the receiving logical unit.* If the program wants to determine whether the message actually arrived, it can, as part of the output scheduling request, specify that a definite response be returned by the logical unit. On receiving the response, the program knows that the message arrived successfully or unsuccessfully. (For many logical units, the return of a positive response indicates not only that the message arrived successfully but also that it was processed successfully.) Since scheduling output usually takes relatively little time, a request to schedule the sending of a message may often be specified as a synchronous operation; it may also be specified as an asynchronous op eration with ECB-posting or RPL exit-routine scheduling specified.

In scheduling output, the program may choose not to require that a response be returned to every message; it may ask that a response be returned only to the last in a series of messages. Receipt of a positive response confirms successful arrival and processing of the message or series of messages, while receipt of a negative response indicates an error. Successful arrival and processing of a message can also be assumed if the resultant input message contains what the program expects or, in the event of an error, it can be assumed that a terminal operator will take the initiative in notifying the application program that he or she is waiting for a message that has not arrived.

By scheduling the sending of a message, the program reserves for itself the determination of whether confirmation of arrival and processing is necessary. When fewer responses are requested, greater message throughput is possible. The user, however, does not have a free hand entirely, since SNA protocols dictate when some responses must or may be requested.

Receiving Input from Any Connected Logical Unit Except Those Already in Conversation

ACF/VTAM provides a way of receiving input from any connected logical unit. To do this, a RECEIVE with OPTCD=ANY is issued. On completion, the identity (the CID) of the logical unit from which input has been received is in the ARG field of the RPL associated with the RECEIVE request. (The INQUIRE macro instruction with the CIDXLATE option can be used to translate the CID into the symbolic name of the logical unit.) Typically, a RECEIVE with OPTCD=ANY is issued to receive the initial input that will lead to a conversation with a logical unit.

Once a RECEIVE with OPTCD=ANY has been used to get initial input from a logical unit, that logical unit can be switched to another mode called *continue-specific mode.* When a logical unit is in this mode, a message from the logical unit will not satisfy a

RECEIVE with OPTCD=ANY; the message can only satisfy a RECEIVE with OPTCD=SPEC and whose RPL identifies the logical unit from which the message was received. While the logical unit is in continue-specific mode, the application program maintains specific control over each message sent to or received from the logical unit.

Thus, a program can consist of a single RECEIVE with OPTCD=ANY that is reissued each time it is completed and sets of specific RECEIVE and SEND macro instructions, with each set of specific macro instructions controlling the conversation with a particular logical unit. To obtain the continue-specific facility, OPTCD=CS is specified in the request at the point at which the logical unit is to be switched to continue-specific mode. For example, the RECEIVE that reads input from *any* logical unit (except those already in continue-specific mode) specifies OPTCD=(ANY,CS). This places the logical unit whose input satisfied the RECEIVE in continue-specific mode; the next issuance of the RECEIVE with OPTCD=ANY excludes this logical unit from being able to complete the RECEIVE. Sample Program 2 in Part 3 shows use of a RECEIVE with OPTCD=(ANY,CS).

Some Decisions That Affect Program Organization

Figure 3-6 lists some of the decisions that anyone designing and coding an ACF/VTAM application program must make. Some of the alternatives in managing the control blocks and work areas in the ACF/VTAM application program are discussed below.

Handling Control Blocks and Work Areas

The application program can handle control blocks and logical-unit-related work areas (data areas and status flags) in a number of ways. It can:

- Define RPLs, NIBs, or EXLSTs in the application program during assembly, or generate them during program execution by using the GENCB macro instruction
- Assign one RPL or NIB to a specific logical unit during assembly, or assemble or generate RPLs and NIBs that are to be available for any logical unit as the need arises
- Retain the RPL used in connecting the logical unit for all further communication with the logical unit
- Use one RPL for all connection requests and use another RPL or group of RPLs for all communication requests
- Define the RPLs, NIBs, and any other control blocks to be associated with logical units as a pool so that a limited amount of control block storage is not exceeded

In application programs that must handle many logical units concurrently, it may be useful to have a control block other than the RPL or NIB associated with a particular logical unit or logical unit conversation. ACF/VTAM provides a way of associating a storage area with a particular logical unit. The application program initially associates the storage with the logical unit by specifying the address of the area in the USERFLD of the NIB prior to issuing the OPNDST macro instruction; thereafter, whenever input is received from the logical unit, ACF/VTAM provides the specified address in the RPL's USER field.

Program Function	Decisions to Make
Opening and closing the program	• One ACB or more than one ACB? • Send final message to logical units before closing the program or simply disconnect them? • How is the program to terminate normally? By network operator closing down the network? (Use a TPEND exit routine.) By special message from one or more particular logical units? By some internal logic, such as time-of-day? By system operator message (for example, via WTOR)?
Connecting logical units	• Logon expected? (Use OPNDST with OPTCD=ACCEPT.) Analyze user logon message before connecting? (Use LOGON exit routine, INQUIRE to obtain user logon message, and OPNDST with OPTCD=ACCEPT to connect.) If not, can use OPNDST with OPTCD=ACCEPT in main program logic. • Who furnishes the logon? (The program logic may not have to be aware of this.) Automatic logon (user-defined logon that automatically schedules the LOGON exit routine for all or some logical units)? Logical unit (that is, a predefined logon message stored in and sent from the logical unit)? Terminal operator associated with a logical unit (the logical unit forwards a message from the terminal operator after perhaps modifying it in some way)? The application program itself, by issuing a SIMLOGON macro instruction? Another application program, by issuing a CLSDST macro instruction with the OPTCD=PASS option? A secondary application program, by issuing a REQSESS macro instruction? More than one of the above? (The LOGON exit routine may have to handle more than one of these kinds of logons.) • Are session parameters used? Will session parameters be supplied with the logon? – Is an INQUIRE macro needed to investigate the session parameters? – Will the application program ever have to modify the parameters supplied with the logon? Will the session parameters always be supplied solely by the primary application program? – Is a logon mode name to be provided in the LOGMODE field of the NIB? – Is a bind area address to be provided in the NIB? • Logon not expected? Identity of logical units known to program? (Use OPNDST or SIMLOGON to acquire the logical units.) – Acquire as many as are available? (Use OPNDST with OPTCD=(ACQUIRE,CONALL).) – Acquire any (single) one of them? (Use OPNDST with OPTCD=(ACQUIRE,CONANY).) – Simulate a logon so that all logical units are connected by the same logic? (Issue SIMLOGON and then issue OPNDST with OPTCD=ACCEPT in LOGON exit routine or in main program.) Identity of logical units known to program? (Use INQUIRE with OPTCD=TERMS to obtain the identities of the defined logical units, then issue OPNDST or SIMLOGON.) • Expect to share connected logical units with other programs? (Use RELREQ exit routine to handle requests from other programs for your logical units.)

Figure 3-6 (Part 1 of 3). Some Decisions That Affect Program Design and Coding

Program Function	Decisions to Make
Communicating with logical units	
• Receiving	• Identity of logical unit known (CID of logical unit in ARG field of RPL)? (Use SEND/ RECEIVE with OPTCD=SPEC.) • Identity of logical unit unknown? (Use RECEIVE with OPTCD=ANY to read input, then RECEIVE/SEND with OPTCD=SPEC.) • Exclude logical unit from having its input complete a request for any input while it is in the midst of a transaction? (Specify OPTCD=CS.) • Expect to receive responses to messages (using SENDs with other than POST=RESP)? Can use either: RECEIVE that includes RTYPE=RESP A RESP exit routine • Expect to receive expedited-flow control commands relating to quiescing or shutting down? Can use either: RECEIVE that includes RTYPE=DFASY A DFASY exit routine • Want to be able to receive session-control commands from a logical unit? (Use a SCIP exit routine.) • Can length of input message vary greatly? (Use PROC=KEEP in connecting logical unit and reissue RECEIVEs until entire message is read.) • Expect that logical unit may want to quiesce your program (temporarily stop it from sending)? Have logic to receive QEC and RELQ commands as result of DFASY input.
• Sending	• Send all of a message at once? Wait for the message to arrive at its destination before proceeding? (Use SEND with OPTCD=SYN, POST=RESP.) Start the message on its way and have ACF/VTAM post an ECB or schedule an RPL exit routine when ACF/VTAM receives a response to the message? (Use SEND with OPTCD= ASY, POST=RESP.) Have ACF/VTAM schedule the sending of the message and determine its arrival yourself? (Use SEND with OPTCD=SYN [or ASY] ,POST=SCHED,RESPOND=values.) – Have the logical unit return a definite response which will cause completion of a RECEIVE with RTYPE=RESP specified or cause entry to an RESP exit routine? (Specify RESPOND=NEX,FME,NRRN, or RESPOND=NEX,FME,RRN, or RESPOND=NEX,NFME,RRN, according to user preferences.) – Have the logical unit return only an exception (negative) response and either assume successful arrival or determine from the next received message that the input arrived successfully? (Specify RESPOND=EX,FME,RRN, or RESPOND=EX,FME,NRRN, or RESPOND=EX,NFME,RRN, according to user preferences.) – Have the logical unit return no response and determine successful arrival yourself? (Specify RESPOND=NEX,NFME,NRRN.) • Send an element in a chain of elements? (Specify SEND with POST=SCHED and CHAIN= FIRST, MIDDLE, or LAST with RESPOND=EX on all but the last SEND.) Request a definite response on the last SEND (to determine that the entire chain arrived successfully)? (Specify RESPOND=NEX,FME,RRN, or NEX,FME,NRRN, or NEX,NFME,RRN [according to user preferences] and receive the response with a RECEIVE (RTYPE=RESP) or with a RESP exit routine, or specify POST=RESP on the last element and the operation will not be posted complete until the response comes back.) Request only a negative response and assume by subsequent action of the receiver that the chain was received successfully? (Specify RESPOND=EX,FME,RRN, or EX,FME,NRRN, or EX,NFME,RRN according to user preferences.)

Figure 3-6 (Part 2 of 3). Some Decisions That Affect Program Design and Coding

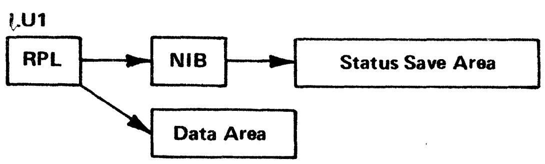
Program Function	Decisions to Make
Handling errors and special conditions	• What kind of information should be saved in a LERAD exit routine? Does the logic error affect only one logical unit or the entire program? • Which physical errors can be retried in a SYNAD exit routine? Which require that the logical unit be disconnected? Which require sending a system operator message? Which require that the program terminate? • What action should be taken in the NSEXIT and LOSTERM exit routines.? • What action should be taken when ACF/VTAM abnormally terminates while running under a user's task? Should the application program have a STXIT AB, STAE, or ESTAE exit routine to investigate and clean up its own files?
Handling control blocks and work areas	
• For connection	• Acquiring one in a list of known logical units? Decide whether to: Assemble a NIB or a list of NIBs and an RPL (for the OPNDST) into the program. Generate and initialize the NIB or list of NIBs and the RPL dynamically, using the GENCB macro instruction. Obtain the NIB or list of NIBs and the RPL from a pool, assembled or created dynamically, and initialize them. • Acquiring one or a list of unknown logical units? (Use INQUIRE with OPTCD=TERMS to create and initialize NIBs and use an assembled, generated, or pool-obtained RPL.) • Accepting logon? Decide whether to: Have a LOGON exit routine and reuse the same NIB and RPL for each connection request. Have one or more OPNDSTs in the main program, each of which will require an RPL and a NIB. • Do you need to create a BNDAREA for session parameters?
• For communication	• Simple program with synchronous requests? (Assemble RPLs and data area in program and reuse for each request.) • Asynchronous program? Assemble, generate, or obtain from an assembled or generated pool one RPL, one ECB (if an ECB will be posted), and one work area (data area and flags) for each active logical unit? Decide whether to use this storage for the duration of: – Connection to disconnection with the logical unit – RECEIVE and a related SEND – A series of RECEIVEs and SENDs Put address of logical-unit-related storage in USERFLD of NIB if storage obtained at connection?

Figure 3-6 (Part 3 of 3). Some Decisions That Affect Program Design and Coding

Techniques for Handling Control Blocks and Work Areas

Here are some techniques that can be used in handling control blocks and work areas:

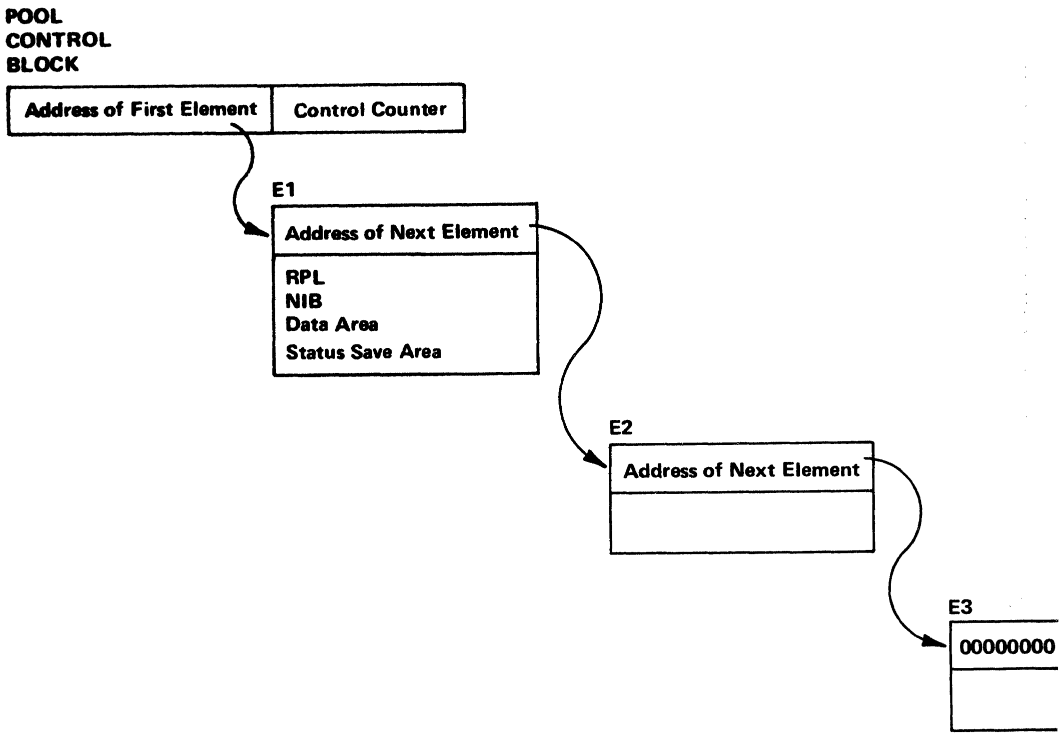
Element Per Logical Unit at Assembly: This method works well if a known set of logical units is to be connected to the program. A separate predefined RPL is used for each logical unit that is connected. The RPL points to a NIB and to a data area. The USERFLD field of the NIB can be set to point to a status save area for the logical unit.



To do this, a correspondence can be set up between each logical unit and its RPL. For example, a logical unit table can be constructed that matches a logical unit's symbolic name with its RPL. Since fixed connection is being used, a separate OPNDST macro can be coded for each logical unit; each OPNDST can be coded to use a specific RPL. When a data-transfer request is completed, the RPL's USER field points to the status save area. The RPL continues to point to the data area.

This method can be used if the program accepts logons from a known set of logical units. Whenever a logical unit logs on, the program is passed the logical unit's symbolic name. In the LOGON exit routine, this name is used as an index into a logical unit table to find the correct RPL. That RPL is used to connect the logical unit and to control data-transfer requests.

Element Per Logical Unit at Connection: This method can be used if the program accepts logons from logical units whose names are *not* known at assembly. A pool of RPLs, NIBs, data areas, and status save areas can be set up to be used as they are needed. The pool can consist of elements; each element contains one RPL, one NIB, one data area, and one status save area. As each logical unit logs on, the program selects an available element from the pool and uses some technique to associate the element with the logical unit. One technique is to put the address of the element in the USERFLD field of the NIB prior to issuing the OPNDST macro instruction. Subsequently, whenever execution of an RPL-based macro instruction is completed, the address is available in the USER field of the RPL. Here is a sample showing how the pool might be set up:



This coding could be used to construct the pool:

```
         DS    0D
PCB      DC    A(E1)                 POOL CONTROL BLOCK
         DC    A(0)                  CONTROL COUNTER
E1       EQU   *                     FIRST ELEMENT
         DC    A(E2)
         RPL   AM=VTAM,ACB=ACB1,NIB=NIB1,AREA=AREA1,
               OPTCD=(ACCEPT,ANY)
NIB1     NIB   USERFLD=A(SAREA1)
AREA1    DS    200C                  DATA AREA
SAREA1   DS    18C                   STATUS SAVE AREA
E2       EQU   *                     SECOND ELEMENT
         DC    A(E3)
         .
         .
         .
```

After selecting a pool element for a logical unit (when it logs on), that element can be used with the same logical unit for all subsequent requests. When the logical unit logs off or is disconnected, the element is returned to the pool. Here is an example to illustrate one method of managing the elements in the pool. Assume that the program is about to connect a logical unit for which a logon is queued. A pool element is needed for the logical unit. If no elements are available, the program issues a CLSDST, using a reserved RPL, to disconnect the logical unit and remove the queued logon. Or the program may want to connect the logical unit, write a "resources unavailable" message, and then disconnect it. Here is the sample coding:

```
*GET ELEMENT FROM POOL

START    LM    EREG,SREG,PCB       GET ADDR FIRST ELEM AND COUNTER
GLOOP    LTR   EREG,EREG           TEST IF POOL EMPTY
         BE    POOLMTY
         L     WREG1,0(EREG)       GET ADDR SECOND ELEMENT
         LR    WREG2,SREG          GET CONTROL COUNTER
         AL    WREG2,=A(1)         INCREMENT COUNTER
         CDS   EREG,WREG1,PCB      UPDATE PCB
         BNE   GLOOP               IF CDS DID NOT WORK

*EREG NOW HAS ADDRESS OF OBTAINED ELEMENT

EREQ     EQU   6
SREG     EQU   EREG+1
WREG1    EQU   10
WREG2    EQU   WREG1+1
         .
         .
         .
POOLMTY CLSDST . . .               CLSDST IF NO ELEMENTS AVAILABLE
         .
         .
         .
*PUT ELEMENT BACK IN POOL
*EREG HAS ADDRESS OF ELEMENT TO BE PUT BACK
         L     WREG1,PCB           GET ADDR FIRST ELEMENT
LOOP     ST    WREG1,0(EREG)       CHAIN ELEMENT TO PREVIOUS
         CS    WREG1,EREG,PCB      PUT ELEMENT IN POOL
         BNE   LOOP                IF UPDATE DID NOT WORK

*ELEMENT IS BACK IN POOL
```

Element Per Transaction: Here again, a pool is used for storage management. However, instead of assigning an element to a logical unit for the life of the connection, a new element is obtained for each transaction. A transaction is a two-way interchange: data goes both from and to the logical unit. An element is obtained for connection and returned when connection is made. A new element is obtained for each transaction and returned when the transaction is completed. Using this technique, a NIB does not have to be included in the elements used for the transactions.

Element Per Request: This is a more dynamic version of the "element per transaction" method. Here, a new element is used for each request. As each request is completed, the element is returned to the pool.

As a modification to this method, one NIB and one RPL can be used for all connection requests and additional pool elements obtained for subsequent data-transfer requests. The NIB and RPL for connection can be assembled in the program; they need not come from a pool.

To maintain a strict "element per request" technique, the data area portion of an element can be used to hold a NIB for connection and as a data area for data transfer. To do this, one NIB can be set up in the program. For each connection request, the contents of the NIB can be moved into the data area in the pool element. Or, a GENCB can be used to build a complete NIB in the data area. This reduces the size of the pool elements.

Combinations: These techniques can be combined to suit particular needs. Here are two ways to combine storage management techniques:

1. At assembly, establish a fixed status save area for each logical unit to be connected. Each status save area can contain a user identification to be compared to one contained in a logon, or the save area can be used to count the number of times the logical unit has logged on during the day. At connection, a pool element is obtained (containing RPL, NIB, and data area) for each request. The fixed save area provides a permanent place to keep logical unit information.
2. Assign one status save area per logical unit at connection. This is more dynamic than the method in 1, above, in that the programmer does not have to know at assembly which logical units will log on. One RPL and data area per request or per transaction can be used. Again, the status save area can be used to keep track of logical unit activity, but only between connection and disconnection. The RPL and data area, selected from a pool, allow dynamic data-transfer requests.

Using Multitasking

In addition to the multithreading facilities provided by ACF/VTAM, the operating system provides multitasking facilities that may be used when writing an ACF/VTAM application program that concurrently handles a large number of sessions. Multitasking can be used to separate communication activity from other activity such as disk I/O, to divide communication activity among several tasks, or to do both.

Using Multitasking to Separate Data Communication Activity from Other Activity

Multitasking can be used so that communication activity can occur while waiting for other activity such as disk I/O processing to be completed (see Figure 3-7). For example, an ACF/VTAM application program can be organized into a task that opens and closes the ACB and performs ACF/VTAM requests, and a task that performs disk I/O (VSAM) requests.

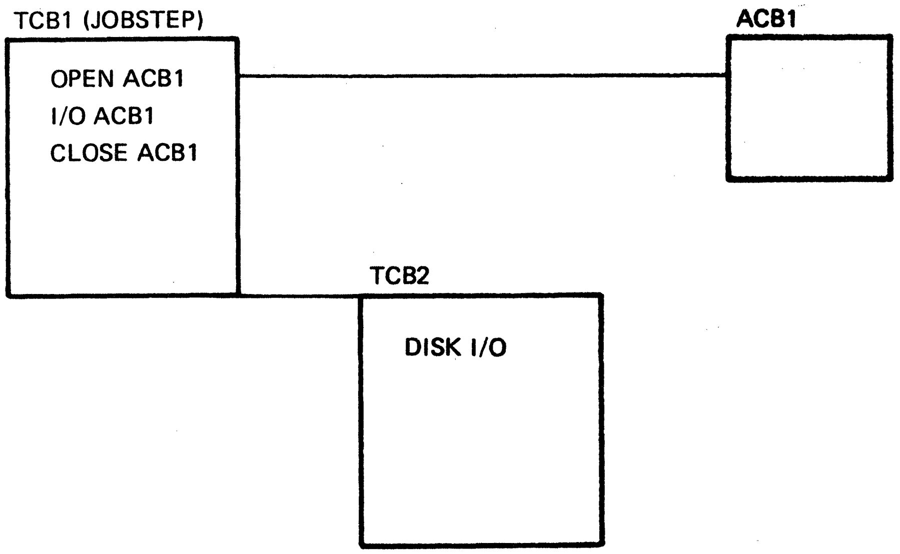


Figure 3-7. Multitasking a Program

In such a program, a page fault occurring during a request in the task that performs disk I/O requests does not prevent the task that performs communication requests from getting control during the time that the system is waiting for the required page to arrive in main storage. In a single-task ACF/VTAM application program, a page fault would require that the entire program wait.

Using Multitasking to Divide Data Communication Activity among Several Tasks

Further efficiency may be possible by issuing ACF/VTAM requests in more than one task. For example, a program can use one task to open and close the ACB and to connect and disconnect logical units, and the program can use a number of other tasks, each containing a RECEIVE that specifies OPTCD=ANY and additional I/O requests. Whenever one ACF/VTAM I/O task has to wait, the system can schedule another ACF/VTAM I/O task.

There are two different ways to use multitasking to divide communication activity among several tasks: (1) a program can be written so that the first task attaches subtasks and all tasks use the same ACB, or (2) a program can be written so that the first task attaches subtasks and each task uses a separate ACB.

Multiple Tasks, Using the Same ACB

When multiple tasks in a job step use the same ACB (see Figure 3-8), the following considerations apply:

- The macro instructions (OPEN and CLOSE) that open and close the ACB must be in the same task.
- To multitask the same ACB in OS/VS1 or OS/VS2 SVS, a program must be privileged.
- Subtasks (non-ACB opened) should terminate only when no outstanding ACF/VTAM requests remain.
- The task that closes the ACB should ensure that other tasks refrain from issuing ACF/VTAM requests during and after CLOSE processing.
- In OS/VS1, OS/VS2 SVS, and OS/VS2 MVS, tasks that perform ACF/VTAM I/O requests must be lower than or equal in the task structure to the task that opens the ACB.

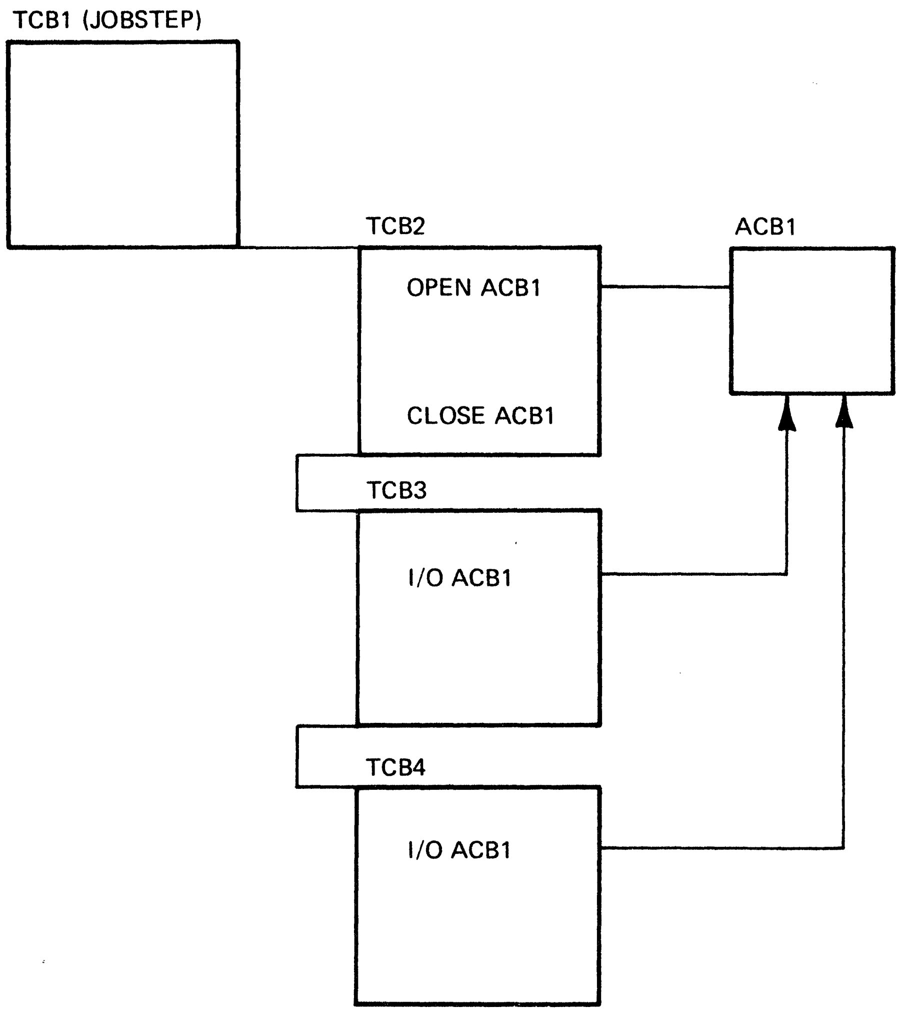


Figure 3-8. Multiple Tasks, Using the Same ACB

- All exit routines, both RPL-specified exit routines and EXLST exit routines, are scheduled to run as part of the task in which the ACB is opened. If any of the ACF/VTAM I/O tasks is dependent on information that may be detected in an exit routine (such as a response being received by a RESP exit routine), the exit routine must be able to communicate with that task (perhaps by posting an ECB located in a common area).
- In OS/VS1 and OS/VS2 SVS, any ABEND issued in an exit routine results in the abnormal termination of the task that opened the ACB as well as all of its subtasks.
- In DOS/VS, all ACF/VTAM processing occurs as part of the task in which the ACB was opened.

Multiple Tasks, Each with Its Own ACB

In an ACF/VTAM application program consisting of more than one task, each task can open its own ACB (see Figure 3-9). In such a structure, the following considerations apply:

- The macro instructions (OPEN and CLOSE) that open and close a particular ACB must be issued in the same task.

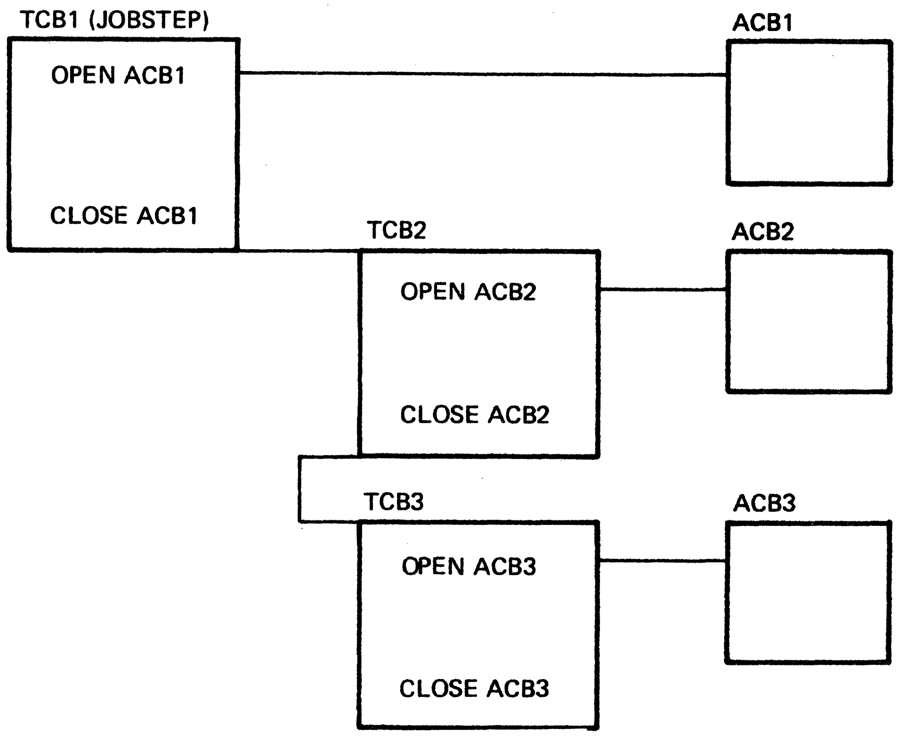


Figure 3-9. Multiple Tasks, Each with Its Own ACB

- In OS/VS1, OS/VS2 SVS, and OS/VS2 MVS, any task that opens an ACB can perform ACF/VTAM requests only on that ACB (not on any of the other ACBs that have been opened by related tasks).

Using Multiple ACBs within One Task

An ACF/VTAM application program that remains a single task can open more than one ACB (see Figure 3-10). By doing this, some ACB-specified exit routines, such as the TPEND exit routine, can be used in common by ACBs, while other exit routines can be associated with only one particular ACB. A possible use of multiple ACBs is to code one set of ACB-specified exit routines for one set of terminals (for example, all basic-mode terminals that may be connected) and a different set of exit routines for another set of terminals (for example, all record-mode terminals or logical units that may be connected).

Using Authorized Path in OS/VS2 MVS

In OS/VS2 MVS, an ACF/VTAM application program can specify that individual SEND, RECEIVE, RESETSR, and SESSIONC macro instructions be executed by ACF/VTAM in a path that requires fewer instructions. This faciltity, called *authorized path*, can be used to improve performance in an ACF/VTAM application program.

To use authorized path, the program must be authorized and in the supervisor state. *OS/VS2 System Programming Library: Supervisor*, GC28-0628, describes how to specify an authorized program. The MODESET macro instruction can be used to put the program into supervisor state.

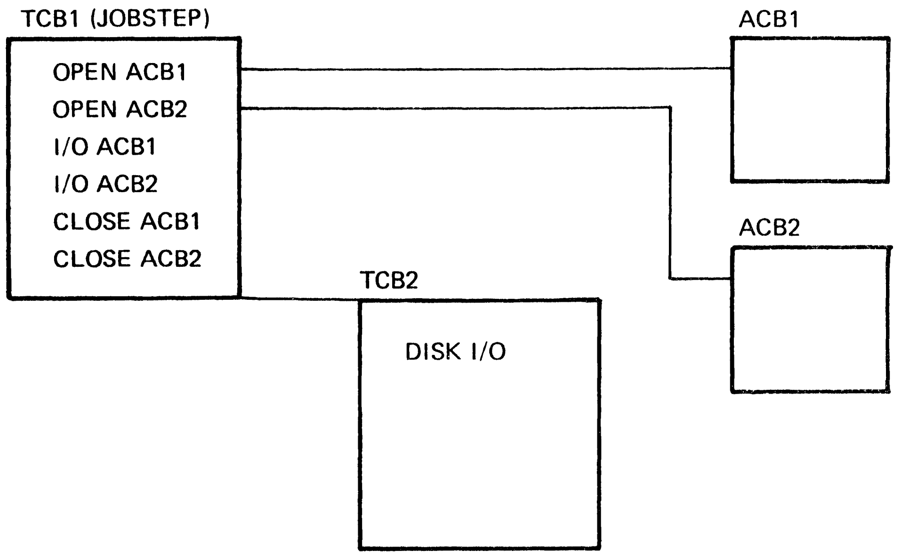


Figure 3-10. A Single Task with Multiple ACBs

The ACF/VTAM application program can use authorized path while running under a TCB or while running under an SRB. To use it while running under a TCB, the authorized program, having put itself into supervisor state, specifies BRANCH=YES on any SEND, RECEIVE, RESETSR, or SESSIONC macro that is to be executed using authorized path. (Subsequently, to issue any macro instruction that does not use authorized path and that uses the same RPL, the RPLBRANC flag in the RPL must be turned off either by (1) coding BRANCH=NO on a MODCB macro instruction, (2) referring to the field by using the IBM-supplied DSECT and turning it off with an assembler language instruction, or (3) by coding BRANCH=NO on the subsequent macro instruction that does not use authorized path.)

Authorized path is always used when SEND, RECEIVE, RESETSR, or SESSIONC is issued under control of an SRB. One way to use authorized path under an SRB is for the authorized program, while running under a TCB, to specify an RPL exit routine when issuing (in supervisor state) a SEND, RECEIVE, RESETSR, or SESSIONC macro that specifies BRANCH=YES. On entry to the RPL exit routine, the program will be running under an SRB. Any SEND, RECEIVE, RESETSR, or SESSIONC in this environment is automatically executed using the authorized path; BRANCH=YES need not be specified. An alternative way to create the SRB environment is to use the SCHEDULE macro instruction. No RPL-based macro instruction other than SEND, RECEIVE, RESETSR, SESSIONC, and CHECK should be issued while running under an SRB.

Figure 3-11 illustrates the basic logical requirements for using authorized path when running under a TCB and under an SRB. The program in Figure 3-11 is highly simplified. The program only connects and handles input from one logical unit, whereas an actual program would connect and handle input from several logical units. In addition, the logic associated with input/output requests would be more complex in an actual program. The following notes are keyed to the numbers in Figure 3-11.

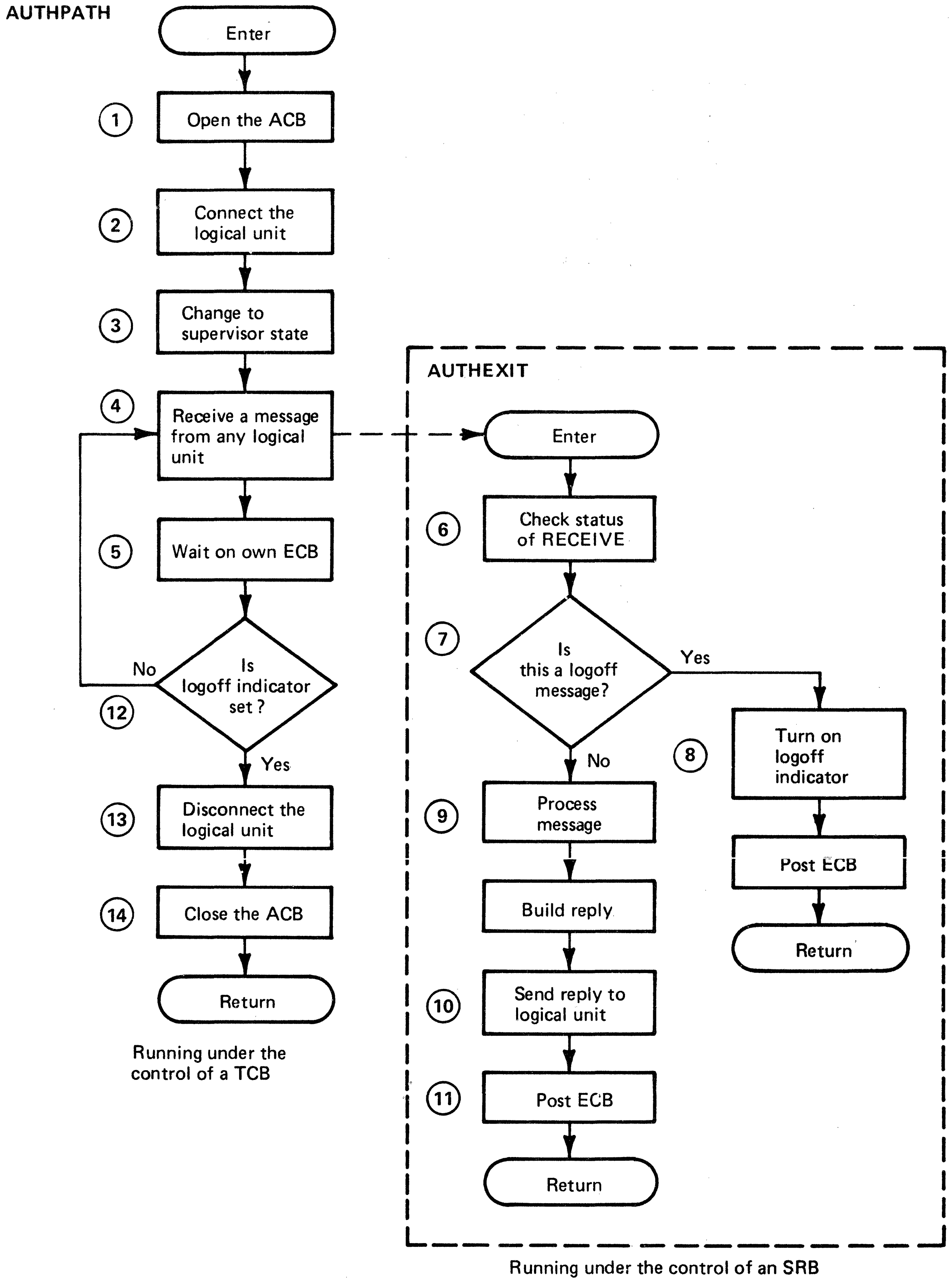


Figure 3-11. The Logical Requirements for Using Authorized Path (OS/VS2 MVS)

1 The application program begins processing as a task in OS/VS2 MVS, running under the control of a TCB. As part of normal ACF/VTAM processing, it issues an OPEN macro instruction to open an ACB. The OPEN might be coded like this:

```
OPEN        AUTHACB
```

In this sample program, AUTHACB contains:

```
AUTHACB   ACB        AM=VTAM,APPLID=APPL5ID,PASSWD=APPL5ID
```

2 Next, the application program issues an OPNDST macro instruction to connect the logical unit. The OPNDST might be coded:

```
OPNDST      RPL=AUTHRPL,OPTCD=SYN
```

The RPL, named AUTHRPL, contains the rest of the information needed for the OPNDST.

3 Now the application program uses the MODESET macro instruction to change into the supervisor mode. This is coded:

```
MODESET     MODE=SUP
```

4 The RECEIVE macro instruction conforms to the coding rules for authorized path running under the control of a TCB. The BRANCH=YES operand is specified. The RECEIVE macro instruction might be coded:

```
RECEIVE     RPL=AUTHRPL,RTYPE=DFSYN,AREA=INPUT00,
            AREALEN=100,OPTCD=(ASY,ANY,CS),EXIT=AUTHEXIT,
            BRANCH=YES
```

It is known that a message received from the logical unit will never exceed 100 bytes.

5 Because the RECEIVE was specified as an asynchronous operation (ASY in OPTCD), the main program AUTHPATH can continue execution until an input message from the logical unit completes the receive-any operation. In a more elaborate program, meaningful processing could be done here. But in AUTHPATH, the program immediately enters a wait state, waiting on its own ECB.

6 When a message is received from the logical unit, control goes to the RPL exit routine named AUTHEXIT. Note that this exit routine runs under the control of an SRB, and that the exit routine receives parameters that are different from those received by an RPL exit routine running under a TCB. On entry to AUTHEXIT:

- Register 1 contains the address of the RPL.
- Register 13 *does not* contain a save area address because no save area is provided. (This is also true of an RPL exit routine running under a TCB.)
- Register 14 contains the return address of the OS/VS2 MVS dispatcher.
- Register 15 contains the entry-point address of the exit routine.

The CHECK macro instruction frees the RPL for reuse and causes entry to a LERAD or SYNAD exit routine if necessary. The CHECK macro instruction is the only ACF/VTAM macro instruction other than SEND, RECEIVE, RESETSR, and SESSIONC that can be issued under control of an SRB. Any other ACF/VTAM macro instruction will fail. The CHECK macro instruction is coded:

```
CHECK       RPL=AUTHRPL
```

7 The exit routine then tests the input message to see if it is a logoff message (a message in a special format that indicates the logical unit wants to end communication with the program AUTHPATH).

8 If the message is a logoff message, the exit routine turns on a logoff indicator, posts the ECB, and returns control to AUTHPATH.

9 If the message is not a logoff message, the exit routine analyzes the message and builds a reply.

10 The exit routine is running under the control of an SRB because it is an exit routine entered from a macro instruction using authorized path. The SEND macro instruction therefore uses authorized path. The SEND looks like this:

```
SEND      RPL=AUTHRPL,OPTCD=(SYN,CA),CONTROL=DATA,
          STYPE=REQ,RTYPE=DFSYN,RECLEN=95,AREA=OUTPUT00,
          POST=SCHED,RESPOND=(NEX,NFME,NRRN)
```

The macro instruction specifies that the SEND operation is to be performed synchronously (SYN in OPTCD), meaning that the exit routine surrenders control until the SEND operation is scheduled. The macro instruction also specifies that no response is to be returned, which assumes that failure of the message to arrive will be detected by analyzing the next message entered by the terminal operator.

11 After the SEND operation has been scheduled, the exit routine posts the ECB on which the main program, AUTHPATH, has been waiting. The exit routine then returns control to AUTHPATH.

12 Because the ECB has been posted, the wait at 5 is satisfied and AUTHPATH continues execution. It tests to determine whether the logoff indicator has been set. If the indicator has not been set, it returns to 4 to reissue the RECEIVE macro instruction. Thus, execution continues through steps 4 through 12 for as long as normal input messages are received from the logical unit.

When the logoff indicator has been set (indicating that the message received from the logical unit was a logoff message), execution continues at 13.

13 The program disconnects the logical unit by using the CLSDST macro instruction. The CLSDST might be coded:

```
CLSDST    RPL=AUTHRPL,BRANCH=NO,OPTCD=SYN
```

The BRANCH=NO operand turns off the RPLBRANC flag that was turned on by the BRANCH=YES operand in the RECEIVE macro instruction. This must be done for the CLSDST macro instruction to be executed correctly. Because there is no authorized path for this macro instruction, the flag cannot be on when CLSDST is executed.

14 The CLOSE macro instruction closes the ACB.

Chapter 4. Opening and Closing a Program

Opening a Program

After an ACF/VTAM application program has been started, it must notify ACF/VTAM that it is to be recognized as an active element in the network. To do this, the program issues an OPEN macro instruction. On completion of the OPEN, ACF/VTAM has modified its control blocks and tables to indicate that the program is present in the network. ACF/VTAM can then accept requests from logical units for connection to this program. The program is also now able to make further requests of ACF/VTAM. Normally, the program remains "open" until a CLOSE macro instruction is issued when the program is terminating.

ACF/VTAM considers each open ACB to be a separate application program. Therefore, if an application program opens more than one ACB, VTAM sees each open ACB as a different program, even though the ACBs are related to the same program.

What Is Required to Open a Program

Two things are required to open an application program:

An ACB (defined with an ACB or GENCB macro instruction)

An OPEN macro instruction

Because an ACB can point to a list of exit routines, defined with an EXLST macro instruction, an EXLST macro may also be required.

The Access Method Control Block (ACB)

The access method control block contains information that describes the ACF/VTAM application program to ACF/VTAM. After an ACB has been opened, logical units that become connected to the program in reality become connected to the ACB. An ACB contains:

- The name of the access method to be used in opening the ACB (VTAM).
- The address of an application program identification. The application program identification must match a name that was specified on an APPL statement provided as part of the ACF/VTAM definition. When the program opens an ACB, ACF/VTAM searches an internal table. If it finds a match, the ACB is opened; if it does not find a match, the ACB is not opened.

  Note: *If no application program identification is available when an ACB is opened (that is, the APPLID operand was not specified in the ACB or in the OPEN macro instruction), ACF/VTAM uses the job-step name (the label of the EXEC statement) as the application program identification in OS/VS. If the programmer starts a job step that opens multiple ACBs, he or she must ensure that the application program identification is not missing from more than one ACB. In DOS/VS, if the programmer starts an application program in which no application program identification is provided in the ACB, the job name in the JOB statement is used as the application program identification.*

  The application program identification is put into a storage area with the other constants in the program. It must be left-adjusted, and can be no longer than 8 bytes. In the byte that precedes the identification, the length of the identification is coded:

  ```
  APID1    DC    AL1(L'NAME)
  NAME     DC    C'JOE'
  ```

  The address of the length byte (for example, APID1 above) is coded in the APPLID field of the ACB.

Alternatively, the length of the application program's identification can be specified:

```
APID1    DC    X'03'
         DC    C'JOE'
```

although this would require that both statements be changed if the name of the application program were changed.

- Optionally, the address of a password can be associated with the application program. When an ACB is opened, the password in the ACB is compared with a password defined in the APPL statement at network definition. ACF/VTAM keeps this password in an internal table. These passwords must match, or the ACB is not opened. If no password is specified on the APPL statement, no password need be specified on the ACB macro instruction.

  The password is put into a storage area in the program. It must be left-adjusted, and can be no longer than 8 bytes. In the preceding byte, the length of the password is coded:

  ```
  PSWD1    DC    AL1(L'AUTH01)
  AUTH01   DC    C'AUTH01'
  ```

  The address of the length byte (for example, PSWD1 in the preceding example) must be coded in the password operand of the ACB.

- The name of an exit list containing the names of exit routines written in the ACF/VTAM application program to handle specific events.

- An indication of whether ACF/VTAM is to queue logons directed to the application program identification specified in this ACB. If the programmer so specifies (MACRF=LOGON), ACF/VTAM queues logons on the ACB. Each queued logon causes the LOGON exit routine to be entered or a connection request in the main program to be completed. If the programmer specifies that logons are not to be queued (MACRF=NLOGON), no logons are saved if they cannot immediately be processed.

  If the ACB is being defined for an application program that will act as a secondary logical unit in any of its sessions, MACRF=LOGON must be coded. This is necessary for the secondary application program to be able to handle its end of the connection process properly.

Here is a sample ACB macro instruction used to build an access method control block.

```
ACB1       ACB    AM=VTAM,APPLID=APID1,
                  PASSWD=PSWD1,EXLST=EXIT,MACRF=LOGON
            .
            .
            .
APID1      DC     AL(L'AP1NAME)
AP1NAME    DC     C'MYPROG'
PSWD1      DC     AL1(L'PASSCHAR)
PASSCHAR   DC     C'JOE007'
EXIT       EXLST  AM=VTAM,LERAD=LGERRTN,SYNAD=PHYSERTN,etc.
```

where:

ACB1 is the symbolic name for this ACB; it will be included in the OPEN macro instruction that is used to open this ACB.

AM=VTAM tells the operating system that ACF/VTAM open processing will be used for this ACB.

APID1 is the address of the application program identification (MYPROG). When the ACB is opened, ACF/VTAM compares MYPROG to the entries in an internal table. Logon entries to this program are directed to MYPROG.

PSWD1 is the address of the password (JOE007). This must match the password coded in the appropriate entry in the internal table. If they match, or if no password was coded in the table, the ACB can be opened. If the passwords do not match, the ACB cannot be opened.

EXIT is the name of the exit list created by the EXLST macro instruction.

MACRF=LOGON specifies that (1) ACF/VTAM will queue logons for this ACB, and (2) ACF/VTAM will schedule the SCIP exit routine when a Bind command is received by the program. Neither action can occur, however, until the program has issued a SETLOGON with OPTCD=START.

An ACB can also be created when the program is being executed by issuing a GENCB macro instruction.

The OPEN Macro Instruction

Having created an ACB, the program opens it by issuing an OPEN macro instruction. For example:

```
OPENPROG     OPEN     ACB1
```

This macro instruction opens an ACB with the name ACB1. (Note that the OPEN and CLOSE macro instructions use a positional rather than a keyword operand.)

Using Multiple ACBs in an ACF/VTAM Application Program

Normally, an ACF/VTAM application program has only one ACB; the program is known to ACF/VTAM by only one APPL identification. However, a program can be known under two or more different APPL identifications, and each requires that a separate ACB be opened. One OPEN macro instruction can be used. For example:

```
OPENPROG     OPEN     ACB1,ACB2      (DOS/VS)
OPENPROG     OPEN     (ACB1,,ACB2)   (OS/VS)
```

Where the OPEN Can Be Issued

Normally, the OPEN macro instruction is issued in the communication part of the ACF/VTAM application program. The OPEN macro instruction cannot be issued from an exit routine. Information pertaining to the opening of the ACB, multiple ACBs, and multitasking ACBs can be found in Chapter 3.

Closing a Program

An ACF/VTAM application program closes itself by issuing a CLOSE macro instruction that specifies the program's ACB. The CLOSE macro instruction is used in the same way as the OPEN macro instruction. Normally, it should be issued in the communication part of the ACF/VTAM application program and must not be issued from an exit routine. The CLOSE request tells ACF/VTAM to mark the program as no longer present in the ACF/VTAM network. For example:

```
CLOSE     ACB1
```

When the program is closed, any logical units that have not previously been disconnected (with the CLSDST macro instruction) are disconnected. Any outstanding operations are posted complete. In addition, the program can no longer issue the SENDCMD or RCVCMD macro instructions.

In general, there are three ways that a program can learn that it should close its ACB.

- The program can determine itself that it should close (perhaps by determining the time-of-day).
- The program can receive a special text or data message, either from a logical unit or from the network operator, indicating that the program should close operations.
- The program's TPEND exit routine is entered, either because the network operator has issued a HALT command or because some abnormal event has caused ACF/VTAM to be terminated. There are three kinds of HALT commands: a standard HALT command (which contains neither the QUICK nor the CANCEL operand), a HALT QUICK command (which initiates a *quick closedown*), and a HALT CANCEL command (which initiates a *cancel closedown*). When all application programs running under ACF/VTAM are to be ended at the same time, the user must choose to end the programs by issuing a standard HALT command or a HALT QUICK command. Neither of these two commands, however, is completed (that is, ACF/VTAM is not halted) until all application programs have closed their ACBs. Particular actions to be taken by the TPEND exit routine in response to the different HALT commands are described in separate discussions below.

The Program Initiates Closing

The program can itself recognize that is has reached the end of its operations and should close itself. It might recognize this either as part of its normal processing or because it has encountered an error or special condition, such as the lack of a certain resource. For a normal end to operations, the program can send a final message to all logical units connected to it. For an error or special condition, it can send the final message as well as record information about the nature of the error. After taking any pre-closing action that it wants, the program issues a CLOSE macro instruction and then terminates itself.

The Program Receives a Closedown Message

The application program can close as the result of a special message from some element in the network. This occurs if closing the program depends on a situation remote from the host computer (and the network operator cannot be informed about the situation). For example, a terminal operator at a logical unit in Chicago knows that Chicago is always the last user of the program. When all terminal operators in Chicago finished using the program, a terminal operator sends a special message to the ACF/VTAM application program, telling it to close its operations. The ACF/VTAM application program then closes in an orderly fashion, notifying the network operator at the host computer.

The TPEND Exit Routine Is Entered

The TPEND exit routine is entered when the network operator issues a HALT command or when, because of an internal error or problem, ACF/VTAM is terminating itself or being abnormally terminated. When the TPEND exit routine is entered, register 1 contains the address of a 2-word parameter list in which:

Word 1 contains the address of the ACB of the application program being shut down.

Word 2 contains a code that indicates the reason for entry to the exit routine:

0	The network operator issued a standard HALT command.
4	The network operator issued a HALT QUICK command, or ACF/VTAM is halting itself in an orderly fashion because of an internal problem.
8	(OS/VS only) The network operator issued a HALT CANCEL command, or ACF/VTAM is being abnormally terminated.

For codes 0 and 4, the TPEND exit routine should take action as indicated in the following paragraphs. For code 8, the exit routine should immediately return control to its main program, where a CLOSE macro instruction should be issued.

Action for a Standard HALT Command

To an ACF/VTAM application program, notification that a standard HALT command (HALT NET without the QUICK or CANCEL operand) has been received represents a request from ACF/VTAM for the program to close its operations, but it also indicates that ACF/VTAM is willing to wait for the application program to do it in an orderly manner.

When ACF/VTAM receives a standard HALT command, it prevents any new application programs for associating themselves with ACF/VTAM (by opening their ACBs) and prevents application programs from making new connections with logical units. ACF/VTAM also stops queuing logons for application programs. But ACF/VTAM allows the programs to continue communications with connected logical units. For each application program, ACF/VTAM schedules the TPEND exit routine (if the program has one) and passes code 0 in the parameter list.

Under these conditions, the application program does not have to immediately close its ACB. The TPEND exit routine can inform other parts of the program that the standard HALT has been issued. It can do this by posting an ECB or by setting a switch that is checked by other parts of the program. The program can continue communications but should end them as soon as it can. It should then disconnect each logical unit and issue a CLOSE macro instruction. Note that the CLOSE macro instruction cannot be issued in an exit routine; it must be issued in the main program.

If the program has no TPEND exit routine and a standard HALT is issued by the network operator, the program has no immediate way of knowing that the HALT command has been issued. The program will continue communicating with logical units until the network operator cancels the program or until ACF/VTAM terminates (because the network operator has entered a HALT QUICK command).

The standard HALT command will not be completed until all application programs have issued a CLOSE macro instruction. If any application program has not closed its ACB after 45 seconds following receipt of the standard HALT command, ACF/VTAM sends the network operator a list of the application programs that are still open. The network operator can then cancel the programs (using the job name that is equivalent to the application program name), or the operator can notify each program to close its ACB by sending it a special message (if the programs are coded to recognize and act upon such a message).

Actions for a HALT QUICK Command or for an ACF/VTAM-Initiated Halt

The TPEND exit routine is also entered when the network operator issues a HALT QUICK command or when ACF/VTAM enters halt-quick processing because of an internal error. In either case, ACF/VTAM wants to close down the network rapidly.

After it receives a HALT QUICK command or after it enters halt-quick processing, ACF/VTAM will not allow any new application programs to associate themselves with ACF/VTAM (by opening their ACBs), nor will it allow application programs to make any new connections with logical units. For programs already in session with logical units, ACF/VTAM will not accept any new data-transfer requests. Any pending data-transfer request is marked complete, with a special flag set in the FDBK2 field of the RPL to indicate that the operation was canceled because of a quick closedown.

When an application program learns that a quick closedown is in progress, the program should close its ACB as soon as possible. The TPEND exit routine learns of the quick closedown by finding code 4 in the parameter list when it is entered. That exit routine should do a minimum of closedown processing and return control to ACF/VTAM as soon as possible so that the main program can issue the CLOSE macro instruction.

The user should be aware that, after the TPEND exit routine has returned control to ACF/VTAM, the halt-quick situation does not prevent ACF/VTAM from scheduling the program's other exit routines (such as the LOSTERM exit routine). Because of that, the TPEND exit routine should set a quick-halt-in-progress switch, which is tested at the beginning of each exit routine. When the switch is on, each exit routine should immediately return control to ACF/VTAM. The TPEND exit routine should also set a switch or post an ECB to signal the main program to disconnect logical units (if the exit routine did not do that) and close the ACB as soon as possible.

For a quick-halt situation, the TPEND exit routine or the main program should disconnect each logical unit with which it was communicating by issuing CLSDST or TERMSESS unconditional macro instructions. If the program closes its ACB without issuing the CLSDST and TERMSESS macros, the logical units will be disconnected serially, thus requiring more time to disconnect the logical units and slowing down the closedown operation.

If a program has no TPEND exit routine, it learns of the quick-halt situation by detecting a special return code when its next ACF/VTAM request is completed. The program should disconnect its logical units and issue a CLOSE macro instruction.

With the HALT QUICK command, as with the standard HALT command, a list of open application programs is sent to the network operator if any application program has not closed its ACB after 45 seconds.

Actions for a HALT CANCEL Command or for Abnormal Termination of ACF/VTAM (OS/VS Only)

In an OS/VS system, ACF/VTAM's receipt of a HALT CANCEL command or an ACF/VTAM abnormal termination also causes entry to the TPEND exit routine. For either event, ACF/VTAM interrupts any data-transfer operation and does not complete it (that is, the RPL is not marked as complete and no ECB is posted or RPL exit routine scheduled). ACF/VTAM will not accept any ACF/VTAM macro instruction except the CLOSE macro instruction. Therefore, when the TPEND exit routine detects code 8 in the parameter list it receives, the exit routine should set a switch or post an ECB to inform the main program that it should immediately issue the CLOSE macro instruction. The exit routine should then return control to ACF/VTAM so that control can be given to the main program.

If an application program does not have a TPEND exit routine or if that exit routine cannot be scheduled, the application program is abnormally terminated.

For more information on the TPEND exit routine, see Chapter 7.

Chapter 5. Connecting and Disconnecting Logical Units

Roles of Primary and Secondary Logical Units in Connection and Disconnection

Communication between two logical units cannot begin until a connection (that is, an LU-LU session) has been established between the logical units. As mentioned in Chapter 1, in any connection between logical units, one logical unit acts as the primary end of the session and the other logical unit acts as the secondary end of the session. The primary logical unit has more control over communications.

One distinction between the primary and secondary logical units involves their roles in connection. The distinction is this: The secondary logical unit can only ***request*** that a session be established; it cannot order the session to be established. It is the primary application program that, after receiving a request for a session, *orders* the session to be started (or rejects the request).

A similar relationship exists between a primary logical unit and a secondary logical unit in bringing an end to a session (that is, in causing disconnection of the logical units). Under normal conditions, the secondary logical unit *asks* that a session be ended by sending a Terminate command (an SNA command) to ACF/VTAM. ACF/VTAM informs the primary program that the Terminate command has been received. Then, either ACF/VTAM or the primary application program takes the action that actually breaks the connection.

In considering connections, the reader should be aware that a logical unit other than a secondary application program can be connected to only one primary application program at a time. In contrast, a secondary application program can have concurrent sessions with more than one primary application program.

The Concepts of Connection

The discussion above indicates that the process of connection and disconnection must be looked at from two viewpoints: from the viewpoint of the primary application program and from the viewpoint of the logical unit that is acting as the secondary end of the session. This chapter looks first at connection from the viewpoint of the primary application program. The role of a secondary application program is discussed later in the chapter.

A primary application program can establish connection in one of two ways: it can *accept* the logical unit or it can *acquire* the logical unit. (For information on requesting connection from a secondary application program, see "How a Secondary Application Program Requests Connection" later in this chapter.)

Acceptance by a Primary Application Program

When a primary application program accepts a logical unit, it does so because a logon was received from or received for the logical unit. A logon is a request from the logical unit to be connected, and the logon contains information needed by the primary application program to perform the next step in the connection process. The logon can come from the logical unit itself, or it can come from one of several other sources:

- From another application program which ended its connection with the logical unit by issuing a CLSDST macro instruction with OPTCD=PASS.

- From the network operator by means of a VARY LOGON command (although this cannot be used to generate a logon on behalf of a secondary application program.)
- From ACF/VTAM when the logical unit is activated. ACF/VTAM automatically generates a logon at that time when the LOGAPPL operand was specified in the definition statement for the logical unit. (This also cannot be used to generate a logon on behalf of a secondary application program.)

A logon can also be generated *within* the application program that the logical unit is to be logged on to by issuance of the SIMLOGON macro instruction. However, such logons, called *simulated logons*, are essentially a form of acquisition and are discussed under "Acquisition by a Primary Application Program" later in this chapter.

A primary application program accepts a connection by issuing the OPNDST macro instruction with OPTCD=ACCEPT.

Acceptance is suitable for primary application programs that do not require access to a specific logical unit or a specific set of logical units in order to function, but instead are designed to service various logical units that require access to the application program. If, for example, the user wants the logical units themselves to designate which application program they wish to use, the user can allow each logical unit to initiate logons so that the application program can accept the logical unit.

Queuing Logons: When ACF/VTAM receives a logon for an active application program and the logon cannot be passed immediately to the application program (for example, no SETLOGON START macro instruction has been issued), the logon is placed on a queue to await processing by the application program. When the logon is placed on the queue, the logical unit from which the logon was received or for which it was created is allocated to the application program if it is a logical unit other than a secondary application program. (A secondary application program is not allocated to the primary program.) As long as the logical unit is allocated (queued) to the application program, it is not available for connection to any other application program; it is available for connection only to the application program to which it is queued. (A secondary application program remains available for connection to other application programs.) The application program and its queued logical unit cannot communicate with each other until the connection is completed by the application program's acceptance of the logical unit. Because a queued logical unit (other than a secondary application program) is effectively eliminated from the system until accepted or disconnected by the application program, the user should ensure that application programs avoid leaving logical units on this queue any longer than necessary. Note especially that the queuing of a logon from a device-type logical unit makes that logical unit *unavailable* for acquisition, as discussed in later paragraphs.

When more than one logon is queued for the same application program, the logons are generally processed in the order in which they were received (that is, the first received is the first to be processed). There is, however, an exception to this: When the program disconnects the logical unit with a CLSDST macro instruction containing OPTCD=PASS (to pass the logical unit to a program that requested it), that logon is placed at the top of the queue and is processed first.

Accepting Logical Units with an Exit Routine: The application program can maintain a LOGON exit routine that ACF/VTAM schedules whenever a logon for the application program is received. ACF/VTAM provides the exit routine with the identity of the logical unit that issued the logon. The application program can either accept the logical unit (with an OPNDST macro instruction) or reject it (with a CLSDST macro instruction).

The application program does not have to use an exit routine to determine when a logon has been received. The application program can issue a connection request (OPNDST with OPTCD=ACCEPT,Q) that will remain outstanding (that is, will not be completed) until a logon is received from a specific logical unit or, optionally, from any logical unit. Although this method is simpler than using an exit routine, the application program does not have the opportunity to inspect the session parameters and to decline the logon. The application program also cannot ensure that the MODE field in the NIB is set properly. For example, it is possible to use a NIB with MODE=BASIC to successfully accept a logon from another application program. However, since all sessions between application programs must be in record mode, any attempt to send or receive messages would fail.

Preventing Logons: Logons cannot be directed to an application program until the application program notifies ACF/VTAM that it is ready to accept them. It controls this with the SETLOGON macro instruction. After a SETLOGON with OPTCD= START is used to start the acceptance of logons, ACF/VTAM schedules the program's LOGON exit routine for each logon that was queued and waiting or that subsequently is received, or it completes any connection requests (OPNDSTs with OPTCD=ACCEPT) that may have been issued outside the LOGON exit routine. Any time during its execution, the application program can notify ACF/VTAM that it is no longer accepting logons by issuing a SETLOGON with OPTCD=STOP or QUIESCE. (In a secondary application program, issuance of SETLOGON with OPTCD=START is necessary to enable the secondary program to perform its end of the connection procedure.)

Types of Acceptance: The application program can issue a connection request to accept a *specific* logical unit, or to accept *any* logical unit for which a logon has been issued.

To accept a specific logical unit, the application program must tell ACF/VTAM the identity of the logical unit; connection is not made until a logon has been issued for that logical unit. The application program can also accept a logon from any logical unit in the network. After connection is established, ACF/VTAM provides the identity of the logical unit.

Acquisition by a Primary Application Program

When the initiative for connection originates in the primary application program, the application program establishes connection by *acquiring* the logical unit. To acquire a logical unit, the application program need not and should not have received a logon from the logical unit. (If the application program has received a logon, it must either accept it or reject it.) When the acquisition request is issued, if the logical unit is active and available, the logical unit is connected (or queued for connection if the application program is simulating a logon on behalf of the logical unit). To be able to acquire logical units, an application program must have been authorized to use acquisition when the program was defined to ACF/VTAM; that is, the AUTH=ACQ operand must have been specified in the APPL definition statement.

A primary application program can acquire a logical unit in either of two ways: (1) by issuing an OPNDST macro instruction with OPTCD=ACQUIRE, or (2) by issuing a SIMLOGON macro instruction to create a *simulated logon* and then accepting the simulated logon.

Acquiring a Logical Unit with the OPNDST Macro Instruction

To acquire a logical unit, the primary application program can issue an OPNDST macro instruction with OPTCD=ACQUIRE at any point at which the application program wants to attempt to acquire connection with one or more logical units.

In using an OPNDST with OPTCD=ACQUIRE, the application program can specify that ACF/VTAM should attempt to acquire a connection (1) with a particular logical unit, (2) with the first logical unit that is available in a set of logical units, or (3) with some or all of a set of logical units. For the latter two possibilities, which involve a set of logical units, the application program defines the set by building a series of contiguous control blocks (NIBs), each containing the name of a logical unit. The series of contiguous NIBs is called a *NIB list*.

Acquiring Connection with a Particular Logical Unit: To acquire a connection with a specific logical unit, the application program issues an OPNDST with OPTCD= ACQUIRE that points to a single NIB (that is, LISTEND=YES was specified when the NIB was defined). As an alternative, the application program can use the SIMLOGON macro instruction as described below.

Acquiring Some or All Logical Units in a Set (CONALL): The OPNDST with OPTCD=ACQUIRE can also be used to attempt to acquire connection with some or all of the logical units in a NIB list. In this case, the macro instruction points to the first NIB in a NIB list, and the OPTCD operand contains the CONALL option as well as the ACQUIRE option. When such a macro instruction is issued, as many logical units as are available are connected. This type of acquisition can be used when the application program is willing to proceed with as many logical units as are *available*. (A secondary application program that is active is always *available* for connection. A device-type logical unit is *available* for connection if it is not already connected to or queued for connection to another application program.) After execution of the macro instruction, ACF/VTAM provides information so that the primary application program can determine which logical units were connected and which were not.

Acquiring the First Available Logical Unit in a Set (CONANY): Another variation of the OPNDST with OPTCD=ACQUIRE allows the application program to acquire any one logical unit of a specified set. To specify this type of acquisition, the application program issues an OPNDST with OPTCD=(ACQUIRE,CONANY), and the macro instruction points to the first NIB in a NIB list. When the macro instruction is executed, the first available logical unit in the set is connected. This type of acquisition is useful for application programs that require one of a set of logical units, but for which one logical unit is as good as another.

Acquiring a Logical Unit with the SIMLOGON Macro Instruction

An alternative method of acquiring a logical unit involves the use of the SIMLOGON macro instruction. In this method, the application program issues a SIMLOGON macro instruction, which causes ACF/VTAM to generate a logon for the logical unit and to pass that logon to the application program as though it had come from the logical unit itself. The application program then accepts the logon with an OPNDST OPTCD= ACCEPT, either in its LOGON exit routine or in its main line. The SIMLOGON macro instruction can be used to generate a logon for (1) a particular logical unit, (2) the first available logical unit in a set of logical units (the CONANY option), or (3) for all available logical units in a set (the CONALL option). For the SIMLOGON macro instruction (as for OPNDST OPTCD=ACQUIRE), a set of logical units consists of a series of consecutive NIBs called a *NIB list*. A logon that results from a SIMLOGON macro instruction is called a *simulated logon*.

The use of simulated logons is a form of acquisition because the initiative for the connection is taken within the application program itself; it does not come from the logical unit. And, like OPNDST OPTCD=ACQUIRE, use of the SIMLOGON macro instruction must have been authorized when the application program was defined to ACF/VTAM; that is, the AUTH=ACQ operand must have been specified in the APPL definition statement. (Note that AUTH=ACQ authorizes the program to use OPNDST OPTCD=ACQUIRE *and* SIMLOGON.)

Simulated logons might be used by an application program that employs one part of the program (for example, a LOGON exit routine) to ensure that adequate resources (such as storage or control blocks) are available for accepting a logical unit. If one part of the application program attempts to use a simulated logon to acquire a logical unit, the part that accepts logical units has a chance to determine whether the program can actually afford to establish connection with the logical unit.

A simulated logon might also be used by an application program that wants to ask the current owner (via the RELREQ exit routine) to release a particular logical unit. To cause the current owner to be notified that the logical unit is wanted, the OPTCD=(Q,RELRQ) operand must be included in the SIMLOGON macro instruction.

An application program that initially uses the SIMLOGON macro instruction to acquire logical units can be modified in the future to accept logons that originate at the logical units. If the application program is initially designed to acquire logical units with simulated logons, the modifications can be made more easily because coding to accept the logons already exists (either in the LOGON exit routine or as part of the mainline code).

Acquiring Connected Logical Units: A device-type logical unit (that is, a logical unit other than a secondary application program) cannot be connected to more than one application program at a time. Thus, if a primary application program attempts to acquire a device-type logical unit that is already connected to another application program, the requesting program cannot acquire the logical unit until the other program disconnects it. ACF/VTAM provides a means by which the owning application program can be notified that another application program wants one of the device-type logical units connected to the owning program.

The requesting application program can indicate whether its attempt to acquire a connected device-type logical unit should or should not cause the owning application program to be notified (SIMLOGON with OPTCD=RELRQ or NRELRQ). The requesting application program should request notification (that is, specify the RELRQ option) when it needs the device-type logical unit regardless of its connection status. Notification should not be indicated when the requesting application program needs the device-type logical unit only if it is unconnected.

The owning application program also controls whether it can be notified when another application program issues a connection request (that is, a SIMLOGON) to acquire one of its device-type logical units. To receive such notification, the owning application program must contain a RELREQ exit routine, and that exit routine must have been identified in an exit list pointed to by the ACB. Notification can only occur, therefore, when the requesting application program calls for notification with (OPTCD =Q,RELRQ) in the SIMLOGON and the owning application program contains the means for receiving the notification (a RELREQ exit routine).

Queuing Connection Requests for a Device-Type Logical Unit

An attempt to acquire a logical unit (with either an OPNDST with OPTCD=ACQUIRE or with a SIMLOGON macro instruction) always fails if the logical unit is inactive. However, if the requested logical unit is a device-type logical unit (as opposed to a secondary application program) and is active but unavailable, and the acquisition request is a SIMLOGON request, the request can be queued if the requesting program specifies such queuing. A queued request remains pending until the logical unit becomes available. (Note that an acquisition request entered with OPNDST OPTCD= ACQUIRE cannot be queued; if the requested logical unit is not available, the request immediately fails.)

The reader must be aware that there is a distinction between the meaning of "available" as it applies to acquisition and as it applies to acceptance. When an application program attempts to *accept* any type of logical unit, the logical unit is available if a logon for it has been directed at the application program. When an application program attempts to acquire an active device-type logical unit, the logical unit is available if it is not connected (or queued for connection as the result of a logon) to another application program.

The reader must also be aware that there is a distinction between the meaning of "available" as it applies to a secondary logical unit and as it applies to a device-type logical unit. Since a secondary application program can be the secondary end of multiple sessions, a secondary application program that is active is always available for connection. A device-type logical unit is available for connection only if it is active and is not connected to (or queued for connection to) another application program. The following discussion of queuing connection requests applies only to requests from or requests for device-type logical units.

A program should specify that an acquisition request is to be queued only if the application program does not require the device-type logical unit immediately. To indicate that the request is to be queued if the logical unit is not available, the application program uses a SIMLOGON with OPTCD=Q. In addition, the requesting application program can specify that the owning application program is to be notified of the request by adding RELRQ to the option codes, that is, by using SIMLOGON with OPTCD=(Q,RELRQ). The RELRQ option is effective only if the Q option has also been specified. Figure 5-1 lists the effects of queuing on the various types of connection requests. Note that queuing cannot be specified for an OPNDST with OPTCD=ACQUIRE.

When a program issues a SIMLOGON with OPTCD=(Q,RELRQ) for a device-type logical unit that is already owned, ACF/VTAM notifies the owning application program by scheduling an exit routine (RELREQ). The RELREQ exit routine is scheduled when the connection request occurs *while* the logical unit is already connected. If, *before* the logical unit is connected (that is, while the logical unit is still queued for connection), another application program issues a queued connection request for the logical unit, the RELREQ exit routine is not scheduled. Instead, ACF/VTAM sets a bit in the control block used for connection (the NIB) indicating that another application program has requested the logical unit.

The RELREQ exit routine is provided with the identity of the contested logical unit. The application program can elect to disconnect the logical unit immediately, disconnect it later, or ignore the request entirely. If the logical unit is disconnected, the previous owner can immediately attempt to acquire the logical unit from the new owner (using a queued connection request) so that the logical unit will be returned when it is no longer being used. When the logical unit is disconnected, it is

Type of Connection Request	Meaning When Request Specifies Queuing	Meaning When Request Does Not Specify Queuing
OPNDST ACCEPT		
a specific logical unit (SPEC)	Connect the specified logical unit if a logon has been received for it. Otherwise, connect the logical unit when a logon is received for it.	Connect the specified logical unit if a logon has been received for it. Otherwise, indicate failure in the return code.
any logical unit (ANY)	Connect any logical unit for which a logon has been received (if logons have been received for more than one logical unit, connect the logical unit that has waited the longest). Otherwise, wait until a logon is received from any logical unit and then connect that logical unit.	Connect any logical unit for which a logon has been received (if logons have been received for more than one logical unit, connect the logical unit that has waited the longest). Otherwise, indicate failure in the return code.
OPNDST ACQUIRE		
a set of one (CONANY)	(Cannot be queued)	Connect the logical unit if it is available[3]. Otherwise, indicate failure in the return code.
any one of a set (CONANY)	(Cannot be queued)	Connect the first logical unit in the set (NIB list) that is available. Otherwise, indicate failure in the return code.
as many as are available in a set (CONALL)	(Cannot be queued)	Connect all logical units in the set (NIB list) that are available. If none is available, indicate failure in the return code.
SIMLOGON		
one specific logical unit (CONANY or CONALL with a single NIB)	A. If the logical unit is active[1], live[2], and available[3], generate the logon and either pass it to the application program[4] or queue it for the application program. The return code for the SIMLOGON request indicates successful completion. B. If the logical unit is active but is not live or is not available, queue a session initiation request for the application program. Generate the logon when the terminal becomes live and available. The return code for the SIMLOGON request indicates successful completion. C. If the logical unit is inactive, indicate failure of the SIMLOGON request in the return code.	A. If the logical unit is active[1], live[2], and available[3], generate the logon and either pass it to the application program[4] or queue it for the application program. The return code for the SIMLOGON request indicates successful completion. B. If the logical unit is inactive, or is not live, or is not available, indicate failure of the SIMLOGON request in the return code.
any one of a set (CONANY)	Functionally equivalent to a series of SIMLOGONs, with one SIMLOGON attempted in sequence for each logical unit in the set (NIB list). For each SIMLOGON attempt for a logical unit in the list, items A and B above in this column apply to the attempt. The attempts stop when ACF/VTAM finds an available logical unit and generates the logon. If no logical unit in the NIB list is currently live and available, a session initiation request is queued for each logical unit in the list that is active. When one of those logical units becomes live and available, a logon is created for it, and all other queued session initiation request generated by the SIMLOGON are canceled. If no logical unit in the list is active, failure of the SIMLOGON is indicated in the return code.	Functionally equivalent to a series of SIMLOGONs, with one SIMLOGON attempted in sequence for each logical unit in the set (NIB list). A logon is created for the first logical unit in the list that is active, live, and available. If no logical unit in the list is active, live, and available, failure of the SIMLOGON is indicated in the return code.

Figure 5-1 (Part 1 of 2). Queued and Nonqueued Connection Requests

Type of Connection Request	Meaning When Request Specifies Queuing	Meaning When Request Does Not Specify Queuing
all in a set (CONALL)	Functionally equivalent to a series of SIMLOGONs, with one SIMLOGON attempted for each logical unit in the set (NIB list). For each logical unit that is immediately available, ACF/VTAM generates a logon and passes it to the application program or queues it for the application program. For each logical unit that is active, but not live or not available, ACF/ VTAM queues a session initiation request, which is converted to a logon when the logical unit becomes physically connected or available. If any lgoical unit in the set is inactive, failure of the SIMLOGON is indicated in the return code.	Functionally equivalent to a series of SIMLOGONs, with one SIMLOGON attempted for each logical unit in the set (NIB list). When all logical units in the set are active, live, and available, a logon is generated for each one and passed to or queued for the application program. If any logical unit in the set is not active, or not live or not available, the request fails and the failure is indicated in the return code. In this case, no logon is generated.

Notes:

1 "Active" means that the logical unit has been activated and additionally, for switched logical units that are dial-in only, that a dial connection has been established.

2 All active logical units are "live" except for dial-in start-stop and BSC terminals that have been activated but have not yet dialed in.

3 "Available" means that the logical unit is not connected to or queued for connection to another application program.

4 The logon is passed immediately to the application program if the program has issued an OPNDST ACCEPT for the specific logical unit or an OPNDST ACCEPT,ANY to accept any logical unit, or if the program's LOGON exit routine can be scheduled (the program has issued SETLOGON START). Otherwise, the logon remains pending, awaiting an OPNDST or the issuance of a SETLOGON START.

Figure 5-1 (Part 2 of 2). Queued and Nonqueued Connection Requests

reconnected to the acquiring application program that has waited the longest, which may not be the application program that was the previous owner of the logical unit.

By controlling which application programs release contested logical units and which do not, the user can cause some application programs to be able to obtain and keep logical units more readily than other application programs. Or, the user can establish a policy that all application programs release contested logical units that are not being used; this makes the logical units more generally available.

Disconnection by a Primary Application Program

A primary application program can disconnect a device-type logical unit in one of two ways: it can *release* the logical unit or it can *pass* the logical unit to another application program. The logical unit is released by disconnecting it without regard to which application program (if any) is to receive the logical unit. The logical unit is passed by disconnecting it and designating which application program is to receive the logical unit. Passing must be authorized when the application program is defined to ACF/VTAM (that is, AUTH=PASS must be specified in the APPL definition statement).

Passing and releasing are accomplished by using the PASS and RELEASE options of a CLSDST macro instruction.

When a device-type logical unit is released, ACF/VTAM connects the logical unit to any application program that has attempted to acquire the logical unit with a SIMLOGON macro instruction (and has indicated in the SIMLOGON that its connection request should be queued). If more than one application program has issued a SIMLOGON for the logical unit, ACF/VTAM connects the logical unit to the application program that first issued the connection request. If there are no queued requests to acquire the logical unit, ACF/VTAM generates an automatic logon for the logical unit (unless the automatic logon would be to the program that is releasing the logical unit). If no automatic logon has been specified by the user, the logical unit remains unconnected.

When a device-type logical unit is passed, ACF/VTAM generates a logon, directs the logon to the designated application program, and then disconnects the logical unit from the passing application program. The logical unit is not reconnected until the receiving application program accepts the logon.

A device-type logical unit should be passed only when it is imperative that it be connected to a specific application program and to no other. For example, a user might maintain several application programs, each of which requires the same information from the logical unit before it can be used. Although each application program could conduct its own interrogation, it might be simpler for one application program to obtain the initial information and then pass the logical unit to the appropriate application program.

When the application program passes a logical unit, it can also pass a logon message and session parameters (by using a logon mode name) to the receiving application program. In the example above, the application program might pass the results of the preliminary conversation in the logon message.

For information on disconnection by a secondary application program, see "How a Secondary Application Program Requests Disconnection" later in this chapter.

How a Primary Application Program Performs Connection

Performing connection in a primary application program requires three language elements:

A request parameter list (RPL)

A node initialization block (NIB)

An OPNDST macro instruction

The Request Parameter List (RPL)

The request parameter list, built with the RPL or GENCB macro instruction, contains information that describes a request for connection or data transfer. Either kind of request must identify an RPL. After the request has been completed and the event has been posted, the RPL may be used for another request.

When used for connection, an RPL contains information that describes a connection request. The data-transfer RPL describes a data-transfer request. However, the programmer can build an RPL that contains both kinds of information, and that RPL can be used for both kinds of requests. Here is a sample RPL for a connection request:

```
RPL1    RPL     AM=VTAM,ACB=ACB1,OPTCD=(ACCEPT,SPEC,ASY),
                NIB=NIB1,ECB=ECB1
```

where:

RPL1 is the label for the macro and serves as the name of the RPL.

AM=VTAM specifies the access method.

ACB=ACB1 specifies that the logical unit is to be connected to the ACB labeled ACB1.

OPTCD=(ACCEPT,SPEC,ASY), when used with an OPNDST macro, specifies that asynchronous processing is to be used to accept a logon from the logical unit identified in the NIB.

NIB=NIB1 specifies the address of the NIB containing the name of the logical unit to be connected.

ECB=ECB1 specifies that when the request defined by this RPL is completed, ECB1 is to be posted.

The Node Initialization Block (NIB)

A node initialization block (NIB) describes a logical unit that is to be connected to an ACF/VTAM application program. A NIB contains the symbolic name of the logical unit, user data that is to be associated with the logical unit, processing options to be used when the program communicates with the logical unit, and other items of information. After a logical unit is connected, ACF/VTAM adds the communication identifier (CID), which is ACF/VTAM's means of identifying the session. For BSC, start-stop, and local non-SNA devices, additional device information is provided.

A NIB is built with the NIB or GENCB macro instruction which can specify:

The symbolic name of the logical unit

The mode of communication (basic or record mode)

The processing options

The user data

The logon mode name

The Start Data Traffic indication

The address of a bind area in which the application program can construct a set of session parameters

The *symbolic name* of a logical unit is assigned at network definition. It is the name in the name field of the definition statement (LU statement, APPL statement, LOCAL statement, etc.) used to describe the logical unit to ACF/VTAM. This symbolic name is used only when the program connects a logical unit. After a connection is made, the application program uses the CID to communicate with the logical unit. For acquiring a logical unit, the symbolic name is placed in the NIB before the connection request (OPNDST or SIMLOGON) is issued. For accepting connection following a logon, a symbolic name is coded only if accepting connection from a specific logical unit. For accepting a logon from any requesting logical unit, a symbolic name is not specified in the NIB. ACF/VTAM connects the program to any logical unit that is logging on. When the logical unit is connected, ACF/VTAM puts its CID and symbolic name in the NIB.

The *mode* is either record or basic. Record mode is specified in a NIB used to connect a logical unit, and may optionally be specified if record mode is to be used to communicate with a BSC 3270 or local non-SNA 3270 terminal. Basic mode is specified for a stop-start or BSC terminal and may optionally be specified for a BSC 3270 or local 3270.

The *processing options* (PROC) determine certain characteristics to be assigned to the logical unit; for example, whether certain input from the logical unit will cause ACF/VTAM to schedule a DFASY or RESP exit routine. These options are fully described in *ACF/VTAM Macro Language Reference.*

The *user data* is in a 4-byte field (USERFLD) that allows some relevant data to be associated with the logical unit. A common use is to store the address of an area that contains an ECB, RPL, and work area that are to be associated with the logical unit. Having provided this address initially to ACF/VTAM at connection, ACF/VTAM supplies the address in the USER field of any RPL that receives input from the logical unit. More generally, however, whatever information is in the USERFLD field of the NIB at the time of connection is placed in the USER field of the RPL upon completion of each input operation from the logical unit.

The *logon mode name* is the name of an entry in a logon mode table. The entry contains the session parameters to be used for the connection.

The BNDAREA operand can be specified in the NIB macro to give the location of a *bind area* where the application program can predefine or dynamically construct a set of session parameters to be used for the connection. When the BNDAREA operand contains an address, the LOGMODE operand (logon mode name) is ignored. (The ISTDBIND DSECT can be used to set up session parameters in the bind area.)

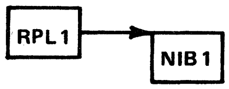
The *Start Data Traffic indication* (the SDT operand) specifies whether the primary application program will issue the Start Data Traffic command (SDT=APPL) or whether ACF/VTAM should issue that command automatically as part of the OPNDST processing (SDT=SYSTEM). The transmission services profile in the session parameters indicates whether or not the Start Data Traffic (SDT) command is to be used in the session. When use of the command is indicated, the SDT command must be sent during initial connection processing, after a Clear command has been sent, and after sequence number resynchronization to inform the secondary end of the session that the flow of messages and responses can begin. For many primary application programs, it is convenient and adequate to let ACF/VTAM issue the Start Data Traffic command during initial connection processing (that is, allow SDT=SYSTEM to take effect by default when defining the NIB). However, the primary application program must still issue an SDT command after a Clear command is sent.

For a secondary application program, the SDT indication is used to indicate a *response* to an SDT command. The secondary application program may return either a positive or a negative response.

A NIB is used in conjunction with each connection request. The RPL used for connection points to one or more NIBs which represent logical units to be connected. Depending on the type of request, there are several ways in which an RPL can point to a NIB (or NIBs). In all cases, the NIB operand of an RPL macro is used to specify the label of a NIB (or GENCB) macro instruction, not the name of the logical unit represented by the NIB.

Here are the ways in which an RPL can point to a NIB:

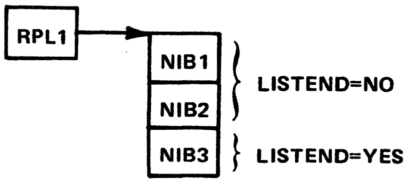
- The RPL can point to a specific NIB. To be sure the NIB is treated as a single NIB, the LISTEND=YES operand must have been included in the macro instruction that defined the NIB or that operand must have been allowed to take effect by default. The LISTEND=YES operand ensures that processing stops with that NIB.



This form is used by the OPNDST and SIMLOGON macros. The requested operation is performed only for the specified logical unit. The RPL and NIB might be coded:

```
RPL1   RPL    AM=VTAM,ACB=ACB1,NIB=NIB1
NIB1   NIB    NAME=LU1,MODE=RECORD,LISTEND=YES,
              USERFLD=A(LUTAB)
```

- The RPL can point to a list of NIBs. To define a NIB list, an application programmer defines a series of contiguous NIBs, either with NIB macro instructions or GENCB macro instructions or by using an IBM-provided DSECT. The last NIB in the list must have the LISTEND indicator (LISTEND=YES in the NIB macro instruction); the other NIBs in the list must have LISTEND=NO.



The coding might look like this:

```
RPL1   RPL    AM=VTAM,ACB=ACB1,NIB=NIB1,OPTCD=ACQUIRE
NIB1   NIB    NAME=LU1,LISTEND=NO,MODE=RECORD
NIB2   NIB    NAME=LU2,LISTEND=NO,MODE=RECORD
NIB3   NIB    NAME=LU3,LISTEND=YES,MODE=RECORD
```

If all NIBs in a program are in one list, the programmer may want to specify a working subset of the list for one operation. To do this, the RPL should point to any one NIB in the list. The subset will include all NIBs from (and including) the NIB to which the programmer has pointed, through (and including) the next NIB in which the LISTEND indicator is set (LISTEND=YES).

This list form can be used in the SIMLOGON and OPNDST (with OPTCD= ACQUIRE) macro instructions when the programmer wants to simulate logons or acquire connections with a set of logical units.

- The RPL can point to a NIB that contains neither a CID nor a symbolic name. This form can be used when the logical unit that will send a logon is not known. When the request is completed, ACF/VTAM fills in the NIB. This form is used by an OPNDST macro instruction with OPTCD=(ACCEPT,ANY). The coding might look like this:

```
RPL1   RPL    AM=VTAM,ACB=ACB1,NIB=NIB1,OPTCD=(ACCEPT,ANY)
NIB1   NIB    MODE=RECORD,PROC=(DFASYX,RESPX)
```

Acquiring Logical Units

To acquire a logical unit, a primary application program issues an OPNDST macro instruction containing OPTCD=ACQUIRE with the RPL pointing either to a single NIB or list of NIBs. Here is a simple example to illustrate the process of acquiring logical units.

Assume that the message-processing portion of a data communication program has been written, and that the program's identification is GOLDEN and its password is AU. GOLDEN is to be connected to three logical units in San Francisco (named SF1, SF2, and SF3) and to two logical units in Boston (named BOS1 and BOS2).

First, an ACB must be built to specify:

That the access method is ACF/VTAM

That the application program identification is contained in an area labeled APID

That the password is contained in an area labeled PSWD

That logons will not be accepted

```
ACB1   ACB    AM=VTAM,APPLID=APID,
              PASSWD=PSWD,MACRF=NLOGON
```

The application program identification is coded in an area labeled APID. The byte preceding the actual identification contains the length of the ID:

```
APID          DC     AL1(L'GOLDNAME)
GOLDNAME      DC     C'GOLDEN'
```

The password is coded in an area labeled PSWD. The first byte of PSWD contains the length of the password:

```
PSWD          DC     AL1(L'GOLDPASS)
GOLDPASS      DC     C'AU'
```

Next, GOLDEN defines the five logical units to which it is to be connected. One NIB is built for each logical unit:

```
SF1     NIB     NAME=SF1,MODE=RECORD
SF2     NIB     NAME=SF2,MODE=RECORD
SF3     NIB     NAME=SF3,MODE=RECORD
BOS1    NIB     NAME=BOS1,MODE=RECORD
BOS2    NIB     NAME=BOS2,MODE=RECORD
```

And GOLDEN has one RPL for each logical unit:

```
RPL1    RPL     AM=VTAM,ACB=ACB1,OPTCD=(ACQUIRE,ASY),
                NIB=BOS1,EXIT=ACQEX
RPL2    RPL     AM=VTAM,ACB=ACB1,OPTCD=(ACQUIRE,ASY),
                NIB=BOS2,EXIT=ACQEX
RPL3    RPL     AM=VTAM,ACB=ACB1,OPTCD=(ACQUIRE,ASY),
                NIB=SF1,EXIT=ACQEX
RPL4    RPL     AM=VTAM,ACB=ACB1,OPTCD=(ACQUIRE,ASY),
                NIB=SF2,EXIT=ACQEX
RPL5    RPL     AM=VTAM,ACB=ACB1,OPTCD=(ACQUIRE,ASY),
                NIB=SF3,EXIT=ACQEX
```

Each RPL is used to connect a specific logical unit using a specific NIB. Since the queuing of a connection request is not possible unless SIMLOGON is used, the connection request will not be queued if the logical unit is not available. The connection is performed asynchronously (OPTCD=ASY). When any connection request is completed, the RPL exit routine ACQEX is scheduled.

To request connection, GOLDEN opens the ACB, and issues five OPNDST macros in the main program:

```
OPEN      ACB1
OPNDST    RPL=RPL1
OPNDST    RPL=RPL2
OPNDST    RPL=RPL3
OPNDST    RPL=RPL4
OPNDST    RPL=RPL5
```

Since the OPNDST macros specify asynchronous operations, GOLDEN receives control again as each OPNDST is accepted by ACF/VTAM. As each OPNDST is completed, the ACQEX RPL exit routine is scheduled. On entry, register 1 contains the address of the RPL for the completed request. GOLDEN can issue a CHECK macro to test for errors, and it can issue a SEND macro to send a message to the logical unit telling the logical unit that it is now connected to the program.

```
ACQEX     BALR     3,0
          USING    *,3
          ST       14,SAVE1
          LA       13,SAVE2
          LR       2,1
          CHECK    RPL=(2)
          (Instructions to test return codes in registers)
          SEND     RPL=(2),AREA=CONNMSG,RECLEN=L'CONNMSG
          (Instructions to test return codes in registers)
          L        14,SAVE1
          BR       14
SAVE1     DS       F
SAVE2     DS       18F
CONNMSG   DC       22C'YOU MAY NOW USE GOLDEN'
```

This example assumes that it is known at assembly which logical units will be needed. In the next example, all that is known is that the program is to be connected to up to 10 active logical units defined during network definition by a PU statement and a series of LU statements. The PU statement is labeled GOLDPU. All available logical units are to be connected. (Some may not be available because they have already been acquired by another program.) No password is used, but a processing option is set for each logical unit.

First, GOLDEN builds an ACB, an RPL, an application program identification, and 10 NIBs (one for each logical unit that may be connected).

The ACB is:

```
ACB1   ACB   AM=VTAM,APPLID=APID,MACRF=NLOGON
APID   DC    X'08'
       DC    CL8'GOLDEN'
```

In the RPL, GOLDEN specifies the name of the NIB list (NIB1) and a NIB that contains the name of the PU statement (GOLDPU) for use with INQUIRE.

```
RPL1    RPL     AM=VTAM,ACB=ACB1,NIB=ADDR,AREA=NIB1,
                OPTCD=(ACQUIRE,SYN,CONALL)
         .
         .
         .
ADDR    NIB     NAME=GOLDPU
```

NIB1 is the address of an area that contains the 10 NIBs needed for connection. GOLDEN also sets PROC=(DFASYX,RESPX) and MODE=RECORD for each NIB so that appropriate input from the logical units can result in scheduling the ACB-specified DFASY and RESP exit routines in the program.

```
NIB1      NIB   PROC=(DFASYX,RESPX),MODE=RECORD
NIB2      NIB   PROC=(DFASYX,RESPX),MODE=RECORD
NIB3      NIB   PROC=(DFASYX,RESPX),MODE=RECORD
NIB4      NIB   PROC=(DFASYX,RESPX),MODE=RECORD
NIB5      NIB   PROC=(DFASYX,RESPX),MODE=RECORD
NIB6      NIB   PROC=(DFASYX,RESPX),MODE=RECORD
NIB7      NIB   PROC=(DFASYX,RESPX),MODE=RECORD
NIB8      NIB   PROC=(DFASYX,RESPX),MODE=RECORD
NIB9      NIB   PROC=(DFASYX,RESPX),MODE=RECORD
NIB10     NIB   PROC=(DFASYX,RESPX),MODE=RECORD
NIBEND    EQU   *
LISTLEN   DC    A(NIBEND-NIB1)
```

GOLDEN opens ACB1:

```
OPEN      ACB1
```

ACB1 is now recognized by and associated with ACF/VTAM. The NIBEND and LISTLEN values, following the NIB list, are used to compute the length of the NIB list for the INQUIRE macro. GOLDEN now uses INQUIRE to fill in the NIBs from GOLDPU:

```
L         6,LISTLEN
INQUIRE   RPL=RPL1,OPTCD=TERMS,AREALEN=(6)
```

The single INQUIRE with OPTCD=TERMS causes ACF/VTAM to fill in the 10 NIBs (one for each active logical unit) in the area starting at NIB1. Each NIB will contain the name of the logical unit, the device characteristics for that logical unit, and the system-assigned values for the remaining processing options. ACF/VTAM will insert a LISTEND=YES indication in the last NIB that is filled in.

Now that all the NIBs are ready, GOLDEN requests connection to all logical units in the list that are available. The RPL already specifies OPTCD=(ACQUIRE,CONALL). Because CONALL is specified, the NIB field of the RPL is set to specify the beginning of a NIB list (NIB1).

```
OPNDST    RPL=RPL1,NIB=NIB1
```

When the OPNDST is completed, GOLDEN is connected to as many logical units as are available. The CID generated for each connected logical unit is in the CID field of the respective NIB. A flag is set in the NIB indicating whether or not the logical unit was connected. If an OPNDST is completed without connecting any logical unit, an error return is set, and the LERAD or SYNAD exit routine in the program is scheduled.

Accepting Logons

A logon can come from any of the following sources: (1) a connection request to the program from a logical unit (including a secondary application program); (2) an automatic logon, as described in the *ACF/VTAM System Programmer's Guide;* (3) a simulated logon, discussed below in "Simulating Logons," (4) a network-operator-initiated logon (VARY LOGON), or (5) the passing of a logical unit from one application program to another (CLSDST OPTCD=PASS).

After a logical unit enters a logon, ACF/VTAM queues the logon for the application program. The program must accept the logon in order to complete the connection.

There are two methods of accepting logons. An OPNDST can be issued in the main program, and the OPNDST will not be completed until the logical unit logs on. Or the program can include a LOGON exit routine, which will be invoked whenever a logical unit issues a logon for the program.

Accepting Logons in the Main Program

To accept a logon in the main program, the main program issues an OPNDST macro instruction with OPTCD=ACCEPT, which is not completed until some logical unit logs on and the logon has been queued on an ACB in the program. When the OPNDST is completed, the logical unit is connected to the program; the RPL specified in the OPNDST macro contains the CID of the logical unit. The overall procedure is as follows:

1. Code an ACB indicating that logons are to be queued.
2. Code an RPL with OPTCD=(ACCEPT,Q) for connection.
3. Construct a NIB that does not identify a logical unit. ACF/VTAM fills in the symbolic name and CID of whatever logical unit logs on.
4. Open the ACB and issue a SETLOGON macro with OPTCD=START to initiate queuing of logons.
5. Issue an OPNDST to connect a logical unit. This may be synchronous or asynchronous.

Here is a simple example. A program is to process data from logical units that log on to the program, and the programmer does not know which logical units will log on. First, an ACB is opened that specifies that logons are to be queued for application program DAVE.

```
ACB1   ACB    AM=VTAM,APPLID=APID,MACRF=LOGON
APID   DC     X'08'
       DC     CL8'DAVE'
       .
       .
       .
       OPEN   ACB1
```

After the program opens ACB1, a SETLOGON macro is issued:

```
SETLOGON      RPL=RPL1,OPTCD=START
```

An RPL is defined that indicates that a logon will be accepted from any logical unit, and that the request for connection is to remain pending (queued) until the logical unit becomes available:

```
RPL1   RPL    AM=VTAM,ACB=ACB1,NIB=NIB1,
              OPTCD=(ACCEPT,ANY,Q)
```

A NIB is coded, specifying no specific logical unit:

```
NIB1          NIB MODE=RECORD
```

If any processing options are to be set in the NIB, they must be set before connecting the logical unit.

Now, the OPNDST is issued:

```
OPNDST        RPL=RPL1
```

After the logical unit is connected to the program, NIB1 will contain the CID and symbolic name of the logical unit so that it can be identified. The ARG field of the RPL also will contain the CID so that this same RPL can be used for data-transfer requests to and from the logical unit.

If the asynchronous option (OPTCD=(ANY,ASY)) had been selected, a CHECK or WAIT macro would be required to await completion of the OPNDST, and the program would be able to do other processing while the request was being processed. A series of OPNDST macros with OPTCD=(ACCEPT,ANY,ASY) could be coded at the beginning of the program. Then, as each logical unit logged on, an OPNDST would be completed, and the result would have to be tested with the CHECK macro instruction.

Accepting Logons in the LOGON Exit Routine

To accept a logon in a LOGON exit routine, the ACB is opened in the main program, and the logical unit is connected in the exit routine. After the ACB is opened and a SETLOGON macro with OPTCD=START is issued, the LOGON exit routine is scheduled for each logon that is received.

Here is a simple example to show how this procedure works. In the main program, ACBO is opened and SETLOGON is issued to allow logons to be queued. When any logon is queued, routine LOGON1 is scheduled to connect the logical unit.

```
PGM1
              .
              .
              .
              OPEN          ACBO
              SETLOGON      RPL=RPL1,OPTCD=START
              .
              .
              .
(Data transfer and message processing)
              .
              .
              .
LOGON1        BALR          3,0
              USING         *,3
              ST            14,SAVE1
              LA            13,SAVE2
              OPNDST        RPL=RPLO
              L             14,SAVE1
              BR            14
              .
              .
              .
```

```
SAVE1    DS      F
SAVE2    DS      18F
ACB0     ACB     AM=VTAM,APPLID=APID,
                 EXLST=EXLST0,MACRF=LOGON
APID     DC      X'08'
         DC      CL8'PGM1'
EXLST0   EXLST   AM=VTAM,LOGON=LOGON1
NIB0     NIB     MODE=RECORD
RPL0     RPL     ACB=ACB0,AM=VTAM,OPTCD=(ACCEPT,ANY),
                 NIB=NIB0
RPL1     RPL     ACB=ACB0,AM=VTAM
```

Notes on the sample coding:

1. The main program opens ACB0 to initiate ACF/VTAM processing and issues SETLOGON to initiate queuing of logons.
2. ACB0 defines the program to ACF/VTAM. ACB0 also specifies that an exit list is used and that logons directed to PGM1 are to be queued (MACRF=LOGON) for ACB0.
3. EXLST0 specifies a LOGON exit routine. Whenever a logon is queued for ACB0, the LOGON1 routine will be scheduled.
4. LOGON1 issues an OPNDST to accept any requesting logical unit. After the logical unit has been connected, LOGON1 returns control to the main program.
5. RPL0 and NIB0 are used for all logon processing. RPL0 specifies that any logon will be accepted and that NIB0 will be used to define the logical unit. (This can be varied by using a storage pool to provide an RPL and a NIB at connection. The connection RPL can then be used for subsequent data transfer.)

As each logical unit is connected, its CID is placed both in the NIB and in the ARG field of the RPL. If a pool of RPLs and NIBs is used, the terminal's CID is now in the RPL for future data transfer. If the same RPL is used for all connection requests, the CID may have to be moved into another RPL that will be used for data transfer.

Using INQUIRE in a LOGON Exit Routine

In the previous example, the LOGON exit routine connects any requesting logical unit without regard for its identity or authorization. It may be desirable to know more about the logical unit before accepting connection to it. The INQUIRE macro can be used to determine suggested session parameters associated with the logon or to determine the contents of the user logon message (which is part of the logon).

When the LOGON exit routine is invoked, register 1 contains the address of a parameter list. The second word of the parameter list contains the address of the symbolic name of the logical unit.

First, the symbolic name of the logical unit must be put into the NIB. Then, INQUIRE is issued to get the user logon message or the session parameters. Here is an example showing how to get the user logon message.

When the LOGON exit routine is entered, the address of an 18-word save area must be loaded into register 13, and register 14 must be saved for returning control. The MODCB macro is used to put the symbolic name of the logical unit (in the second word of the input parameter list) into the NIB.

```
L        2,4(1)
MODCB    AM=VTAM,NIB=NIB1,NAME=(*,0(2))
```

Note that MODCB uses an indirect form of addressing to put the symbolic name into the NIB. The parameter list only contains the *address* of the name, and the NIB needs the *actual name*. This indirect addressing indicates that an address is supplied from which a value is to be taken. Next, INQUIRE is issued:

```
INQUIRE     RPL=RPL1,OPTCD=LOGONMSG
```

The RPL specifies that the user logon message is to be placed in AREA1. Here is the LOGON exit routine:

```
LOGON1      BALR         3,0
            USING        *,3
            ST           14,SAVE1
            LA           13,SAVE2
            L            2,4(1)
            MODCB        NIB=NIB1,NAME=(*,0(2))
            INQUIRE      RPL=RPL1,OPTCD=LOGONMSG,NIB=NIB1
*           VERIFY THE LOGON MESSAGE
            OPNDST       RPL=RPL1,OPTCD=(ACCEPT,SPEC),NIB=NIB1
            L            14,SAVE1
            BR           14
            .
            .
            .
SAVE1       DS           F
SAVE2       DS           18F
NIB1        NIB          MODE=RECORD
RPL1        RPL          ACB=ACB1,AM=VTAM,AREA=AREA1,
                         AREALEN=30
AREA1       DS           CL30
```

Simulating Logons in a Primary Application Program

The simulated logon facility can be used to simulate the process by which a logical unit logs on to a program. But the program itself initiates the connection as it would in acquiring a logical unit. By using the SIMLOGON macro instruction, the application program requests ACF/VTAM to *generate* a logon for a logical unit and to queue it for the SIMLOGON-issuing program as though it had come from the logical unit. If the logical unit is available, ACF/VTAM creates the logon and queues it. When an OPNDST is issued in the LOGON exit routine, the logical unit is connected.

This method can be used as an alternative to acquiring a logical unit. An advantage to using SIMLOGON rather than acquiring the logical unit (OPNDST with OPTCD= ACQUIRE) is that each logical unit (those that send their own logons as well as those for which SIMLOGONs are issued) can be processed by the same LOGON exit routine. Another advantage is that, if the logical unit is not immediately available, the logon will, if requested, be queued until the logical unit becomes available. An OPNDST with OPTCD=ACQUIRE cannot request queuing if the logical unit is not available. Note that a program must be authorized to issue the SIMLOGON macro (that is, AUTH=ACQ must have been specified in the APPL definition statement).

A SIMLOGON macro instruction can be used to create a logon for more than one logical unit. To do this, the NIB field of the RPL specified in the macro instruction points to a NIB that is the first in a NIB list.

Here is the procedure for simulating a logon for a single logical unit:

1. Build an RPL for the request. The RPL points to the NIB for the logical unit to be

connected, and the NIB contains a logical unit name. Assume that the ACB is defined with MACRF=LOGON.

```
NIB1   NIB   NAME=KINGSTON
RPL1   RPL   AM=VTAM,ACB1,NIB=NIB1
```

2. To request the simulated logon, issue SIMLOGON.

   ```
   SIMLOGON     RPL=RPL1,OPTCD=(SYN,Q)
   ```

   When KINGSTON becomes available, ACF/VTAM generates a logon as though it had come from the logical unit. The logon is queued for ACB1.
3. Issue an OPNDST (with OPTCD=ACCEPT) to connect the logical unit. Usually this is issued in a LOGON exit routine, but it can be in the main program.

A user logon message to be included in the simulated logon can be built in a work area. The address of that work area is put in the AREA field of the RPL, and the length of the user logon message is coded in the RECLEN operand.

How a Secondary Application Program Requests Connection

A primary application program can take the initiative in establishing connection with a secondary application program. The initiative can be in the form of a SIMLOGON macro instruction or can be in the form of an OPNDST macro instruction with OPTCD=ACQUIRE. These macro instructions will acquire a secondary application program in the same way they would acquire any other secondary logical unit.

In many cases, however, the initiative for the session comes from the secondary application program. The secondary application program takes the initiative by issuing a REQSESS macro instruction, which asks the primary program for the session. The primary program can either accept or reject the request.

The roles of primary and secondary application program are established by the manner in which the connection is made. The application program that issues the REQSESS macro instruction indicates, by the very act of issuing that macro instruction, that it is to be the secondary half of the session. The program that issues the OPNDST macro instruction assumes the role of the primary end of the session. Having assumed one role or the other, the primary and secondary application programs must do certain things and cannot do others. The capabilities of and limitations on primary and secondary programs are summarized in Figure 5-2.

What a Secondary Application Program Needs to Request Connection

Before a secondary application program can issue a REQSESS macro instruction to ask a primary application program for a connection, the secondary application program must have the following language elements and routines available:

- A request parameter list (RPL) to define the request for connection
- A node initialization block (NIB) to identify the primary application program with which the connection is desired and, optionally, to indicate the logon mode that the secondary application program wants to suggest for the session
- A SCIP exit routine, which is scheduled when a Bind command is received and, later, when other session control commands (for example, a Start Data Traffic command) are received from the primary end of the session.

Primary Application Program	Secondary Application Program
Must issue the OPNDST macro instruction to request ACF/VTAM to establish the session (that is, to connect the two programs).	Cannot issue the OPNDST macro instruction.
Cannot issue the REQSESS macro instruction.	Can issue the REQSESS macro instruction to ask the primary application program to establish a session.
Cannot issue the OPNSEC macro instruction.	Can issue the OPNSEC macro instruction to accept a Bind command and to complete the secondary's end of the session. Can also issue a SESSIONC macro instruction to send a negative response to the Bind command and thereby reject the Bind command and prevent the session from being established.
Can issue the Start Data Traffic command at the beginning of the session (or have ACF/VTAM do it as part of the OPNDST processing) and can issue the command during the session to restart the flow of messages and responses. (The Start Data Traffic command is sent with the SESSIONC macro instruction.)	Cannot issue the Start Data Traffic command. (Optionally, can respond to Start Data Traffic command.)
Can issue the Clear command to stop the flow of messages and responses. (The Clear command is sent with the SESSIONC macro instruction.)	Cannot issue the Clear command.
Cannot issue the Request Recovery command.	Can issue the Request Recovery command to ask the primary application program to take recovery action. (The Request Recovery command is sent with the SESSIONC macro instruction.)
Must send the Set and Test Sequence Numbers command to start resynchronization of message sequence numbers. (The Set and Test Sequence Numbers command is sent with the SESSIONC macro instruction.)	Can only respond to the Set and Test Sequence Numbers command. (A response to the Set and Test Sequence Numbers command is sent with the SESSIONC macro instruction.)
Cannot issue the TERMSESS macro instruction.	Can issue the TERMSESS macro instruction to ask the primary application program to end the session (conditional termination) or to tell the ACF/VTAM servicing the primary application program to end the session (unconditional termination).
Cannot issue the Request Shutdown command.	Can issue the Request Shutdown command to ask the primary application program to end the session. (The Request Shutdown command is sent with the SEND macro instruction.
Can issue the Shutdown command to warn the secondary application program that the session is going to be ended and to tell the secondary application program to prepare for the shutdown. (The Shutdown command is sent with the SESSIONC macro instruction.)	Cannot issue the Shutdown command.
Cannot issue the Shutdown Complete command.	Can issue the Shutdown Complete command to inform the primary application program that preparation for shutdown is completed and the primary application program can now end the session. (The Shutdown Complete command is sent with the SEND macro instruction.)
Can issue the CLSDST macro instruction to end the session (that is, to disconnect the two programs).	Cannot issue the CLSDST macro instruction.

Figure 5-2. Protocols for Sessions between Primary and Secondary Application Programs

- An NSEXIT exit routine to handle network services request units (for example, a network services procedure error request unit if such an error occurs during the attempt to establish the session)

The RPL for a REQSESS Macro Instruction

When used with a REQSESS macro instruction, an RPL defines the manner in which the REQSESS operation is to be performed and identifies the NIB to be used in the operation. Here is a sample RPL for use with a REQSESS macro instruction:

```
REQRPL1    RPL    AM=VTAM,ACB=SECACB1,OPTCD=(ASY,NQ),
                  NIB=PRNIB1,EXIT=REQEXIT1,AAREA=0,
                  AREA=MSGTOPRI,RECLEN=4
```

where:

REQRPL1 is the label for the macro instruction and serves as the name of the RPL.

AM=VTAM specifies the access method that is to be used for the operation.

ACB=SECACB1 identifies the ACB that was opened by the secondary application program. This is the ACB to which the primary application program will be connected when the session is established.

OPTCD=(ASY,NQ) specifies that the operation is to be performed asynchronously (ASY). The NQ operand must be specified and indicates that the request is to be rejected immediately (that is, not queued) and reported as unsuccessful if the primary application program is not available. For example, the primary application program is not available if it has not opened its ACB, has not issued a SETLOGON macro instruction with OPTCD=START to start processing of logons, or has issued the SETLOGON macro instruction with OPTCD=QUIESCE or STOP.

NIB=PRINIB1 specifies the address of the NIB that contains the name of the primary application program with which connection is desired. For a REQSESS macro, the RPL must point to a single NIB; it cannot point to a NIB list.

EXIT=REQEXIT1 specifies that when the REQSESS operation is completed, the RPL exit routine named REQEXIT1 is to be scheduled.

AAREA=0 must be specified or allowed to take effect by default. This operand has no effect in the current level of ACF/VTAM.

AREA=MSGTOPRI specifies the address of a storage area that contains a user logon message to be sent to the primary application program as part of the logon that is generated as the result of the REQSESS macro.

RECLEN=4 specifies that the user logon message in MSGTOPRI is 4 bytes long.

The NIB for a REQSESS Macro Instruction

Two fields in the NIB used with the REQSESS macro instruction are significant for the REQSESS operation: the NAME field and the LOGMODE field.

The NAME field must contain the symbolic name of the primary application program with which connection is desired. This name is the name that was used in the name field of the APPL definition statement when the primary application program was defined to ACF/VTAM.

The LOGMODE field can optionally contain a logon mode name to identify the session parameters that the secondary application program wants to use for the session. If a logon mode name is specified, the name must be one that appears in the logon mode table that is associated with the secondary application program in the host computer in which the secondary application program is being executed. If LOGMODE does not contain a logon mode name (that is, the field contains zeros or blanks), the default session parameters from the logon mode table associated with the secondary application program are used. As the result of the REQSESS macro, ACF/VTAM creates a logon and sends it to the primary application program. As part of the logon process, ACF/VTAM also sends the session parameters so they will be available to the primary application program. For more information on session parameters, see "Establishing Session Parameters during Connection" later in this chapter.

The BNDAREA field of the NIB cannot be used by the secondary application program to specify a set of session parameters to be sent to the primary application program. ACF/VTAM ignores this field during a REQSESS or OPNSEC operation.

The Role of a SCIP Exit Routine in Session Establishment

During the exchange of commands and responses that establish a session, the secondary application program receives one or two commands that must be processed by a SCIP exit routine in the secondary program. One command is the Bind command, which is generated when the primary application program issues an OPNDST macro instruction. The other command is the Start Data Traffic command, which the primary end of the session may send to the secondary end after the session has been established.

To handle these commands, the secondary application program must have a SCIP exit routine. The scheduling of that exit routine is the only way that the secondary program can learn that the command has been received. In fact, the scheduling of the SCIP exit routine is the way that ACF/VTAM informs any application program of the receipt of a Bind, Unbind, Clear, Start Data Traffic, Request Recovery, or Set and Test Sequence Numbers command.

The Role of an NSEXIT Exit Routine in a REQSESS Operation

A secondary application program must also have an NSEXIT exit routine to handle a network services procedure error request unit if such a request unit is received during the attempt to establish a session.

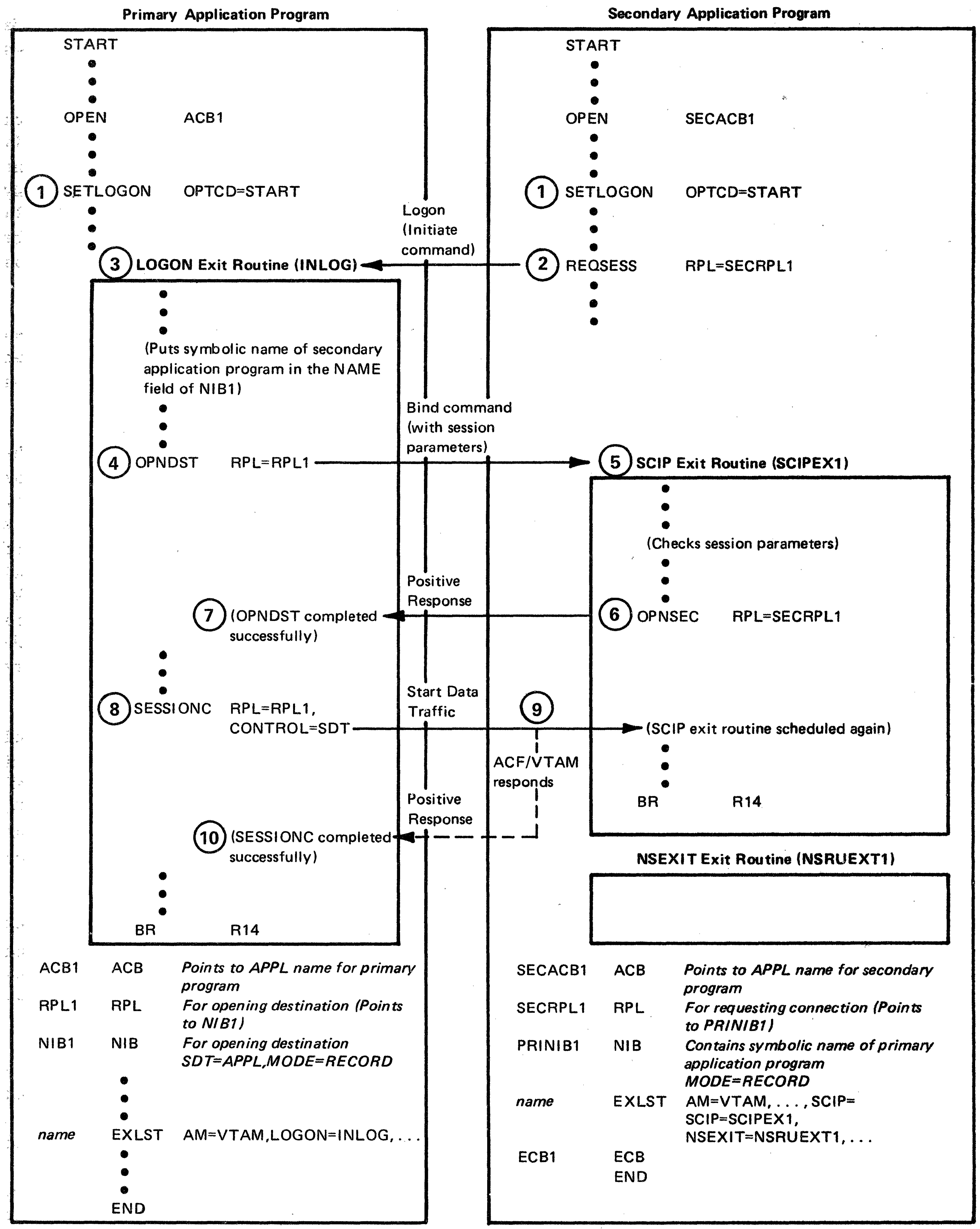
A network services procedure error is an indication that a connection procedure that has been started successfully (and which the application program thinks is proceeding normally) has been interrupted and will not be completed. For example, after a secondary application program has issued a REQSESS macro instruction and the macro instruction has been completed, the next thing the secondary application program expects to receive is a Bind command. Instead, it may receive a network services procedure error request unit. Receipt of that request unit indicates that either (1) the primary application program rejected the request for a session by issuing the CLSDST macro instruction or (2) after the ACF/VTAM that services the primary application program sent a positive response to the REQSESS macro, something happened that prevented that ACF/VTAM from completing its processing of the logon.

Receipt of a network services procedure error request unit is signaled to the secondary application program by scheduling its NSEXIT exit routine.

The General Pattern of a Secondary Program's Request for Connection

When a secondary application program requests a connection to a primary application program and the connection is made without difficulties, the exchange of commands and responses follows the pattern shown in Figure 5-3. The exchange is described in the paragraphs below, and the circled numbers that appear beside the paragraphs refer to related portions of the figure.

A session can be established between application programs in the same domain or different domains. When the programs are in the same domain, they are serviced by the same ACF/VTAM, and the communication between them actually occurs through


Primary Application Program
Secondary Application Program
Logon (Initiate command)
Bind command (with session parameters)
Positive Response
Start Data Traffic
9
ACF/VTAM responds
Positive Response
NSEXIT Exit Routine (NSRUEXT1)

```
Primary Application Program

START
•
•
•
OPEN        ACB1
•
•
•
(1) SETLOGON    OPTCD=START
•
•
•
(3) LOGON Exit Routine (INLOG)
    •
    •
    •
    (Puts symbolic name of secondary
    application program in the NAME
    field of NIB1)
    •
    •
    •
(4) OPNDST      RPL=RPL1
    (7) (OPNDST completed
        successfully)
    •
    •
    •
(8) SESSIONC    RPL=RPL1,
                CONTROL=SDT
    (10) (SESSIONC completed
         successfully)
    •
    •
    •
    BR          R14

ACB1    ACB     Points to APPL name for primary
                program
RPL1    RPL     For opening destination (Points
                to NIB1)
NIB1    NIB     For opening destination
                SDT=APPL,MODE=RECORD
        •
        •
        •
name    EXLST   AM=VTAM,LOGON=INLOG,. . .
        •
        •
        •
        END
```

```
Secondary Application Program

START
•
•
•
OPEN        SECACB1
•
•
•
(1) SETLOGON    OPTCD=START
•
•
•
(2) REQSESS     RPL=SECRPL1
•
•
•
(5) SCIP Exit Routine (SCIPEX1)
    •
    •
    •
    (Checks session parameters)
    •
    •
    •
(6) OPNSEC      RPL=SECRPL1
    (SCIP exit routine scheduled again)
    •
    •
    •
    BR          R14

SECACB1   ACB     Points to APPL name for secondary
                  program
SECRPL1   RPL     For requesting connection (Points
                  to PRINIB1)
PRINIB1   NIB     Contains symbolic name of primary
                  application program
                  MODE=RECORD
name      EXLST   AM=VTAM,. . . ,SCIP=
                  SCIP=SCIPEX1,
                  NSEXIT=NSRUEXT1,. . .
ECB1      ECB
          END
```

Figure 5-3. Exchange When a Secondary Application Program Requests Connection

ACF/VTAM. When the application programs are in different domains, each program is serviced by the ACF/VTAM in its own host computer, and commands and responses flow between the separate ACF/VTAMs.

1 After both programs have been started and have opened their ACBs, each program must issue a SETLOGON macro instruction with OPTCD=START. In the primary application program, this macro instruction tells ACF/VTAM to start scheduling the LOGON exit routine to process any logons that were previously received (and queued for ACB1) and for each future logon that is received. In the secondary application program, the macro instruction makes it possible for ACF/VTAM to schedule the SCIP exit routine when one of several session control commands is received.

2 To initiate the connection, the secondary application program takes the first step by issuing the REQSESS macro instruction. This macro instruction causes ACF/VTAM to create a logon and to send the logon (along with session parameters) to the primary application program.

3 When the logon reaches the primary end of the session, ACF/VTAM notifies the primary application program of the logon in either of two ways: (1) by scheduling the primary application program's LOGON exit routine, or (2) by completing an outstanding OPNDST with OPTCD=ACCEPT. When notification is done by scheduling the LOGON exit routine, a pointer to the symbolic name of the secondary application program attempting to log on is available in the second word of the 4-word parameter list passed to the exit routine by ACF/VTAM. When notification is done by completing an outstanding OPNDST macro instruction, the symbolic name of the secondary application program is available in the NIB associated with the OPNDST. Figure 5-3 assumes that the primary application program has a LOGON exit routine.

The LOGON exit routine performs any checks that the user wants and determines whether to accept the logon. As part of this processing, the exit routine moves the symbolic name of the secondary logical unit to the NAME field of a NIB. To reject the logon, it issues a CLSDST macro instruction. To accept the logon, it issues an OPNDST macro instruction, specifying an RPL that points to the NIB that contains the symbolic name of the secondary application program.

4 The OPNDST macro instruction causes ACF/VTAM to create a Bind command and to transmit that command to the secondary application program. The Bind command contains the session parameters that the primary application program wants to use for the session. Those parameters can be the same as those suggested by the secondary application program or they can be different.

5 When the Bind command is received at the secondary end of the session, the secondary program's SCIP exit routine is scheduled. The fact that the Bind command has been received can be determined by examining the CONTROL field of the read-only RPL provided to the exit routine by ACF/VTAM (the fifth word of the parameter list passed to the exit routine points to the read-only RPL). The SCIP exit routine checks the session parameters passed in the Bind command and determines whether it wants to proceed with establishing the session.

6 If the exit routine decides to go ahead with the session, it prepares a NIB for its next operation (the issuance of an OPNSEC macro instruction). This NIB can be the same as or different from the one that was used with the REQSESS macro

instruction. The exit routine then issues the OPNSEC macro instruction, specifying an RPL that points to the NIB. This macro instruction causes a positive response to the Bind command to be sent to the primary end of the session.

7 Receipt of the positive response causes ACF/VTAM to set up control information for the primary end of the session. This action completes the connection. If the session parameters specified use of the Start Data Traffic (SDT) command, that command must be sent from the primary end to the secondary end of the session before the flow of messages and responses can begin. Upon receipt of the positive response to the Bind command, ACF/VTAM checks the SDT field of the NIB associated with the OPNDST macro. If the field indicates SYSTEM, the primary's ACF/VTAM automatically sends a Start Data Traffic command to the secondary application program. If the SDT field indicates APPL, ACF/VTAM completes the OPNDST macro instruction, and the Start Data Traffic command must be sent by the primary application program. Figure 5-3 assumes that the Start Data Traffic command is to be sent by the primary application program.

8 The primary application program issues the SESSIONC macro instruction with CONTROL=SDT to transmit the Start Data Traffic command. This command informs the secondary application program that the exchange of messages and responses can begin.

9 Upon receipt of the Start Data Traffic command, ACF/VTAM schedules the secondary program's SCIP exit routine to inform the program that the command has been received. After a response to the SDT command has been sent, the secondary application program can send messages and commands according to the conventions established by the session parameters.

10 At the primary end of the session, receipt of the response to the Start Data Traffic command causes completion of the SESSIONC macro that was used to send the command.

This pattern of commands and responses is shown in more detail in Figure C-15 in Appendix C.

The flow is similar when the secondary application program is being acquired (with OPNDST OPTCD=ACQUIRE) by the primary application program. In this case, however, the flow begins at steps *4* and *5*. At step *4* an OPNDST with OPTCD= ACQUIRE produces the Bind command that is sent to the secondary application program. The secondary's SCIP exit routine, at step *5*, is scheduled to process the Bind command. For details on this flow, see Figure C-16 in Appendix C.

Example of a Secondary Application Program Requesting Connection

To associate itself with ACF/VTAM, a secondary application program builds an ACB and opens it. The program also issues the SETLOGON macro instruction with OPTCD=START before attempting to request connection to a primary application program.

To prepare to request connection, the secondary application program builds an RPL to define the request and a NIB to identify the primary application program with which it is to be connected. The secondary application program then issues the REQSESS macro instruction. The following is an example of what the coding might be to this point:

```
SECPGM
            .
            .
            .
            OPEN       SECACB
            (Test for successful completion of the OPEN operation)
            .
            .
            .
REQMACRO    REQSESS    RPL=PROC1RPL
            .
            .
            .
SECACB      ACB        AM=VTAM,APPLID=SECAPLID,EXLST=EXRTNLST,
                       MACRF=LOGON
SECAPLID    DC         AL1(L'MYNAME)
MYNAME      DC         CL7'SECPGM1'
EXRTNLST    EXLST      AM=VTAM,SCIP-CMDINRTN,NSEXIT=NSPERTN,
                       . . . (other exit routines) . . .
SLGNRPL     RPL        AM=VTAM,ACB=SECACB
PROC1RPL    RPL        AM=VTAM,ACB-SECACB,NIB=RQSTNIB,
                       OPTCD=(ASY,NQ),ECB=ECB1,AAREA=0
RQSTNIB     NIB        NAME=PROCESS1,LISTEND=YES,MODE=RECORD,
                       LOGMODE=TALKMOD1
```

As the result of the REQSESS macro instruction, a logon is sent to the primary end of the session. In addition, the session parameters associated with TALKMOD1 are sent to the primary end. The session parameters will be returned to the secondary program in the Bind command, either with or without change by the primary program.

The RPL specified for the REQSESS operation (PROC1RPL) indicates that the operation is to be performed asynchronously (ASY), that the request for connection is not to be queued if the primary program is not immediately available (NQ), and that completion of the operation is to be signaled by posting ECB1. Because the operation is performed asynchronously, the secondary program can continue processing until the operation is completed.

When the REQSESS operation is completed, ECB1 is posted, and the secondary application program issues a CHECK macro instruction to test whether the operation was successful and to mark the RPL as available for reuse:

```
CHECK     RPL=PROC1RPL
```

(CHECK must be used because the operation was asynchronous. If the operation were synchronous [OPTCD=SYN in the RPL], the secondary program would determine the results of the operation by testing register 15 and possibly register 0.) If the operation being checked was unsuccessful, the LERAD or SYNAD exit routine is invoked, if available. Otherwise, control is returned to the application program. Control is also returned to the application program if the operation was successful.

After completion of the REQSESS operation, the secondary application program can proceed with other processing (perhaps with communications if it is also functioning as the primary application program in session with other logical units), or it can enter a wait state. The next thing the secondary application program will see in relation to the connection request is either (1) an indication of a network services procedure error, or (2) an indication that a Bind command has been received from the primary end of the session.

Receipt of a network services procedure error is indicated by scheduling of the secondary program's NSEXIT exit routine, which must be provided in any application program that functions as the secondary end of a session. Receipt of this error indicates either that the primary end of the session rejected the session request by issuing a CLSDST macro instruction or that an error at the primary end of the session has nullified the REQSESS operation, even though successful completion of the macro instruction was reported in return codes. The reason for the error can be determined by examining the request unit and the read-only RPL that ACF/VTAM provides when it schedules the exit routine.

Receipt of a Bind command is indicated by the scheduling of the secondary program's SCIP exit routine, which also must be provided in any application program that functions as the secondary end of a session. The parameter list passed to the exit routine indicates that a Bind command was received and provides the starting address of the session parameters. The exit routine can examine the session parameters to determine whether they are acceptable.

If the session parameters are unacceptable or if, for some other reason, the secondary application program does not want to proceed with establishing the session, the SCIP exit routine issues a SESSIONC macro instruction to send a negative response to the Bind command:

```
SESSIONC RPL=PROC1RPL,STYPE=RESP,CONTROL=BIND,
         RESPOND=(EX,FME)
```

Values that indicate the exact reason for rejection of the Bind command must be provided in the SSENSEO, SSENSMO, and USENSEO fields of the RPL used for the SESSIONC operation.

If the session parameters are acceptable and the secondary application program wants to proceed with establishing the session, the secondary program prepares a NIB for an OPNSEC operation. Assume that the NIB was coded as follows:

```
OPNSCNIB    NIB    MODE=RECORD,USERFLD=data (up to 4 bytes),
                   LISTEND=YES
```

To prepare this NIB for the OPNSEC operation, the SCIP exit routine must move into the NAME field of the NIB the symbolic name of the primary application program that sent the Bind command. Remember that the parameter list passed by ACF/VTAM to the SCIP exit routine contains the address of the session parameters. Within those session parameters is the symbolic name of the application program that sent the Bind command. By using the ISTDBIND DSECT, the programmer can move the symbolic name into the NAME field of the NIB. The SCIP exit routine then issues an OPNSEC macro instruction that cites an RPL that points to OPNSCNIB:

```
OPNSEC      RPL=PROC1RPL,NIB=OPNSCNIB,OPTCD=SYN
```

This macro instruction causes ACF/VTAM to send a positive response to the Bind command, which in turn causes successful completion of the OPNDST at the primary end of the session. The macro instruction specifies that the operation is to be performed synchronously (SYN), meaning that processing in the SCIP exit routine stops until the OPNSEC operation is completed. When the operation is completed, the secondary application program tests register 15 to determine whether the operation was successful. Following successful completion of the operation, the CID for the session is available in the ARG field of the RPL and the CID field of the NIB.

With successful completion of the OPNSEC operation, the connection between the secondary and primary application programs is completed. However, if required by the session parameters, a Start Data Traffic (SDT) command must be sent from the primary end of the session (by ACF/VTAM or the primary application program, depending on the setting of the SDT field in the NIB used with the OPNDST macro instruction). Once the SDT command has been sent and responded to by ACF/VTAM or the secondary application program, the flow of messages and responses can begin.

Establishing Session Parameters during Connection

As part of the connection process, the primary and secondary ends of the session must agree on the communication rules to be followed during the session. These communication rules, called *session parameters*, enable each end of the session to know what the other end of the session will do and will not do in different communication situations.

The session parameters are bit settings that indicate such things as "the primary end of the session will send chained data" or "the secondary end of the session will not ask for responses to messages" or "the secondary end of the session will not send end-of-bracket indications if brackets are used." The session parameters are described in detail in Appendix J of *ACF/VTAM Macro Language Reference*. When the session parameters are part of the Bind command, they also include the symbolic name of the application program that sent the command and the user logon message (if any). The process of agreement on session parameters follows a general pattern, as described in the following section.

The General Pattern of Agreement on Session Parameters

A request for connection to a primary application program reaches that program in the form of a logon. A set of session parameters is associated with each logon. These parameters are available for inspection by the primary application program when it processes the logon.

During processing of the logon, the primary application program can decide to use the session parameters suggested by the originator of the logon, or the primary application program can choose a different set of parameters. In either case, when the application program issues an OPNDST macro instruction to accept the connection, it must designate a set of session parameters to be sent to the logical unit being accepted. The set of session parameters is sent as part of the Bind command, which is created by ACF/VTAM as a result of the OPNDST macro instruction. (Whether the primary application program uses the same parameters as those suggested in the logon or a different set may be determined by user conventions. For example, the primary application program is to always use the session parameters that accompany a logon or is to always disregard the suggested parameters and select session parameters on the basis of some criteria chosen by the user.)

When the Bind command reaches the logical unit, the logical unit can examine the session parameters in the command. At this point, the logical unit must either accept or reject the whole set of parameters; it cannot accept some and reject others. The logical unit accepts the session parameters by sending a positive response to the Bind command; it rejects the parameters by sending a negative response. When the response is negative, the connection is not completed.

Defining Sets of Session Parameters

In many cases, the ends of the session work with predefined sets of session parameters. When a set is defined, a name is associated with the set. That name is known as the *logon mode name*. The logon mode name is used in some logons and in the LOGMODE operand of certain macro instructions and commands to identify the set of session parameters.

Several sets of session parameters, each with its own name, can be grouped into a table known as a *logon mode table*. The table itself is identified by a *logon mode table name*, which is the name specified in the linkage-editor NAME statement when the table is link-edited.

In lieu of using a predefined set of session parameters, a primary application program can build a set of parameters at the time it is needed. The set of parameters is built in an area of the application program known as a *bind area*, whose address is placed in the NIB used for connection.

Tables That Contain Session Parameters

In each domain, predefined sets of session parameters can exist in user-defined tables, in an IBM-supplied default table, or in a user-defined default table that has replaced the IBM-supplied default table. (In subsequent discussions, the term *default table* is used to mean the IBM-supplied default table or the user-defined default table that has replaced the IBM-supplied table.) The tables are stored in a system data set associated with the operating system that controls the host computer in a particular domain.

Logon Mode Tables Built by the User: The user can define one or more logon mode tables by using the MODETAB, MODEENT, and MODEEND macro instructions, which are described in detail in the ACF/VTAM system programmer's guide for the operating system being used. After a table has been coded, it is assembled and link-edited into the appropriate library (the core image library for DOS/VS, SYS1.VTAMLIB for OS/VS1 and OS/VS2 SVS, and SYS1.LPALIB for OS/VS2 MVS).

In coding a logon mode table, the programmer uses a MODETAB macro instruction to identify the beginning of the definition. The symbolic name of this macro instruction becomes the CSECT name for the logon mode table. The symbolic name can also be used in the linkage-editor NAME statement when the table is link-edited, and thus can become the name of the logon mode table.

The MODETAB macro instruction is followed by one or more MODEENT macro instructions. Each MODEENT macro instruction creates one entry in the table, and each entry consists of a logon mode name and a set of session parameters (the logon mode name is the name used to designate the set of parameters). The end of the table is identified by the MODEEND macro instruction. The functions of the macro instructions and the basic structure of a logon mode table are shown in Figure 5-4.

When the network for a particular domain is defined to the ACF/VTAM in that domain, a logon mode table in that domain can be associated with a particular logical unit (a device-type logical unit or a secondary application program). However, a logon

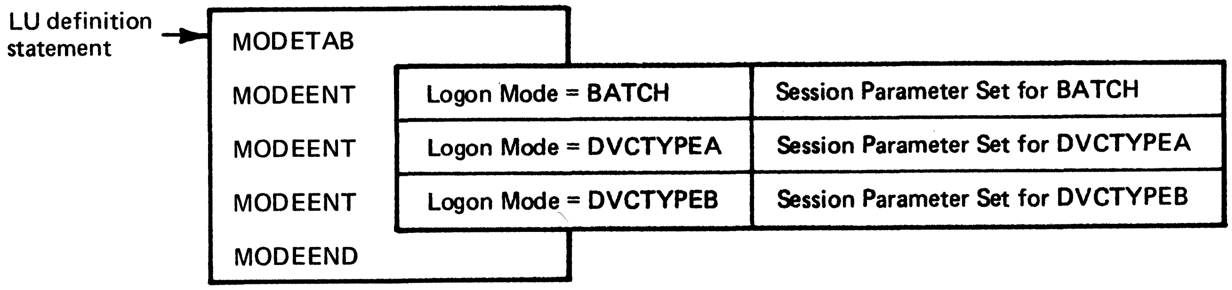


Figure 5-4. Logon Mode Table Macro Instructions

mode table in one domain cannot be associated with a logical unit in another domain. A logon mode table is associated with a logical unit by coding the logon mode table name in the MODETAB operand of the LU statement or APPL statement that defines the logical unit. (For a device-type logical unit, the association can also be made by coding the table name in the PU statement below which the LU statement appears, or in a GROUP or LINE macro instruction.) By making this association, the user identifies the first logon mode table that is to be searched when a logon mode name is supplied as part of the connection request. If the logon mode name is not found in this logon mode table, the default logon mode table is also searched. The various definition statements can identify the same logon mode table, or they can identify different logon mode tables (as indicated in Figure 5-5), but all of the logon mode tables referred to in the definition statements for a particular domain must be in the domain being defined.

Default Logon Mode Tables: If the user has not designated a logon mode table to be used for a logical unit (that is, did not code the MODETAB operand in a definition statement), ACF/VTAM associates a default logon mode table with the logical unit or program. The default table is one of the following: (1) for DOS/VS, the ISTINCLM table in the core image library, (2) for OS/VS1 and OS/VS2 SVS, the ISTINALM table in SYS1.VTAMLIB, or (3) for OS/VS2 MVS, the ISTINCLM table in SYS1.LPALIB.

In each operating system, the table under the name shown above is an IBM-supplied default table unless the IBM-supplied table has been replaced with a user-defined default table. For the contents of the IBM-supplied default table, see the ACF/VTAM system programmer's guide for the operating system you are using. The user can use the MODETAB, MODEENT, and MODEEND macro instructions to code a default table and then use that table to replace the IBM-supplied table in the appropriate library, storing it under the name ISTINCLM or ISTINALM as appropriate.

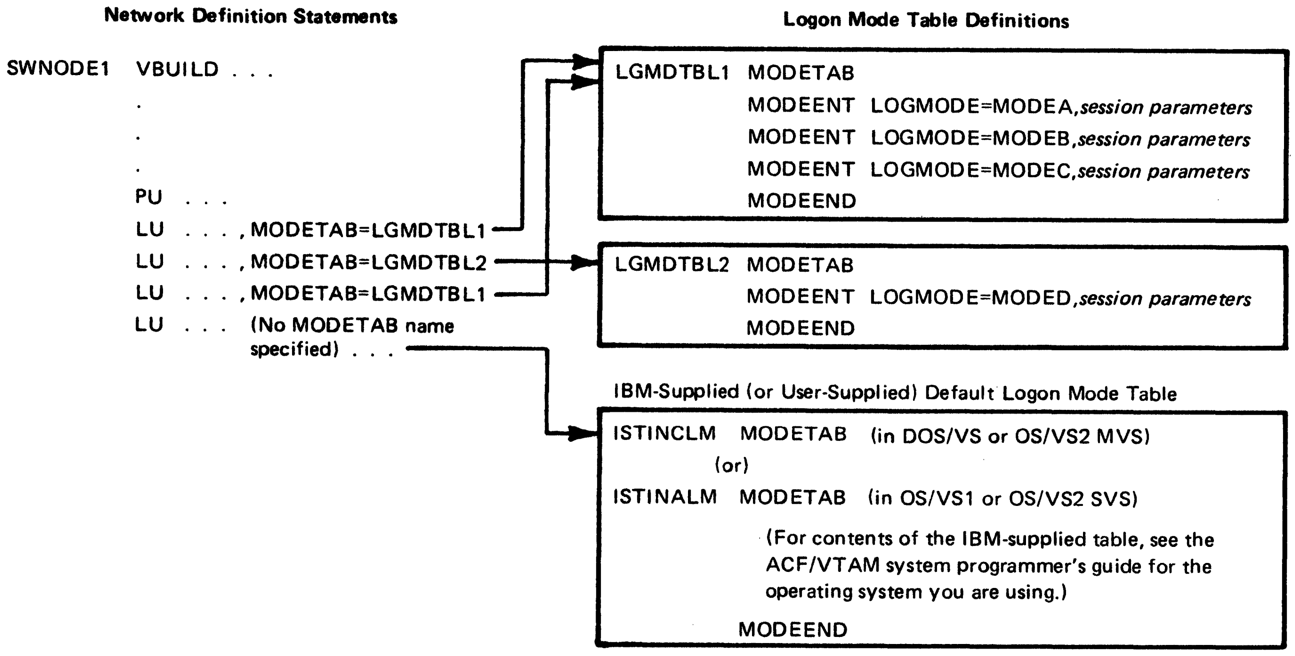

Network Definition Statements
SWNODE1 VBUILD . . .
PU . . .
LU . . . , MODETAB=LGMDTBL1
LU . . . , MODETAB=LGMDTBL2
LU . . . , MODETAB=LGMDTBL1
LU . . . (No MODETAB name specified) . . .
Logon Mode Table Definitions
LGMDTBL1 MODETAB
MODEENT LOGMODE=MODEA,session parameters
MODEENT LOGMODE=MODEB,session parameters
MODEENT LOGMODE=MODEC,session parameters
MODEEND
LGMDTBL2 MODETAB
MODEENT LOGMODE=MODED,session parameters
MODEEND
IBM-Supplied (or User-Supplied) Default Logon Mode Table
ISTINCLM MODETAB (in DOS/VS or OS/VS2 MVS)
(or)
ISTINALM MODETAB (in OS/VS1 or OS/VS2 SVS)
(For contents of the IBM-supplied table, see the ACF/VTAM system programmer's guide for the operating system you are using.)
MODEEND

Figure 5-5. Identification of Logon Mode Tables in LU Definition Statements

The Default Entry in a Logon Mode Table

As indicated above, one of two logon mode tables is associated by ACF/VTAM with each logical unit during ACF/VTAM definition: either the logon mode table identified in the MODETAB operand of the definition statement for the logical unit or, in the absence of such a specification, the default logon mode table. The table associated with the logical unit is the one searched for a logon mode name when such a name is specified for the logical unit during the connection process.

However, if no logon mode name is specified in a connection request (and, for OPNDST, no session parameters are supplied in a bind area), ACF/VTAM must still find a set of session parameters to include in the connection request. In this case, ACF/VTAM takes the default set of session parameters from the logon mode table associated with the logical unit. The default set may be either of two possible default entries:

1. If the user specified the DLOGMOD operand in the definition statement for the logical unit (or in a higher-level definition statement), the logon mode entry named in that operand is used to search the logon mode table associated with the logical unit and is used as the default entry for that particular logical unit.
2. If no DLOGMOD operand was specified for the logical unit, the *first entry* in the logon mode table associated with the logical unit is used as the default entry.

In the remainder of this section, the term *default entry* or *default session parameters* is used for either possibility.

How Logon Mode Names and Session Parameters are Used

A logon mode name can be used at different points in the connection process to designate a particular set of session parameters. A logon mode name can be used in these ways:

- A logical unit can include a logon mode name as part of its logon information to suggest a set of session parameters.
- A secondary application program can specify a logon mode name in the LOGMODE field of the NIB it uses with a REQSESS macro instruction to suggest a set of session parameters.
- A primary application program can specify a logon mode name in the LOGMODE field of the NIB it uses with an OPNDST macro instruction to indicate the session parameters that are to be sent to the secondary end of the session in the Bind command. **Note:** *When the logical unit or secondary application program that is logging on is in another domain, a logon mode name cannot be used with OPNDST OPTCD=ACCEPT.*
- A primary application program can specify a logon mode name in the LOGMODE field of the NIB it uses with a SIMLOGON macro instruction to indicate the session parameters to be associated with the simulated logon.
- A primary application program can specify a logon mode name in the LOGMODE field of the NIB it uses with a CLSDST macro instruction with OPTCD=PASS to indicate the session parameters to be associated with the logon generated as a result of that macro instruction.
- A network operator can specify a logon mode name in the LOGMODE operand of a VARY LOGON command to indicate the set of session parameters to be associated with the logon generated as a result of that command.

As noted previously, the logon mode table that is associated with a secondary logical unit by being named in a definition statement must be stored in a system data set in the domain that owns the logical unit. When a logon mode name is supplied as part of a connection request, the logon mode name is translated into session parameters in the domain that owns the logical unit. If necessary, the logon mode name is passed from the domain in which the connection request originates to the domain that owns the logical unit and is then translated in that domain. Similarly, if no logon mode name is supplied as part of the connection request, the default session parameters are taken from the appropriate logon mode table in the domain that owns the secondary logical unit. The named session parameters or default session parameters are then passed, along with the logon, to the domain that owns the primary application program to which the logon is directed. The suggested session parameters are then available to the primary application program when it begins to process the logon.

Logon Mode for a Logon from a Device-Type Logical Unit

When a logon from a logical unit originates in the primary application program's domain, a logon mode name may accompany the logon when it reaches ACF/VTAM. In this case, the logon mode name is translated into session parameters before the logon is presented to the primary application program for processing. If no logon mode name accompanies a the logon, the default session parameters are presented along with the logon.

A set of session parameters accompanies a logon received from a logical unit in another domain. In this case, the session parameters are found by the ACF/VTAM in the other domain before the logon is transmitted to the primary application program's domain.

Logon Mode for a Logon from a Secondary Application Program

A secondary application program suggests a set of session parameters by setting the LOGMODE field of the NIB associated with the REQSESS macro instruction. The effects of that field are:

If the field contains a logon mode name, the session parameters associated with that name are transmitted with the logon.

If the field contains zeros or blanks, the default entry from the appropriate logon mode table is transmitted with the logon.

Logon Mode for a Simulated Logon

A SIMLOGON macro instruction causes the ACF/VTAM in the domain in which the macro instruction is issued to generate a simulated logon and pass the logon back to the program in a way that makes it look as though the logon was received from the logical unit named in the NIB used with the macro instruction.

The LOGMODE field of the NIB used with the macro instruction controls the session parameters that are associated with the logon. If the LOGMODE field contains a logon mode name, the session parameters identified by that name are provided with the logon. Otherwise, the default session parameters are provided with the logon. If the SIMLOGON involves a logical unit in another domain, the named session parameters or the default parameters are found in logon mode tables in the other domain and returned to the domain in which the SIMLOGON was issued.

Logon Mode for a CLSDST Macro Instruction with OPTCD=PASS

A CLSDST macro instruction with OPTCD=PASS is used to pass a logical unit from one primary application program to another. The macro instruction disconnects the logical unit and causes a logon to be generated on behalf of that logical unit or secondary application program. The logon is presented to the application program whose symbolic name is pointed to by the AAREA field of the RPL used with the macro instruction.

According to the setting of the LOGMODE field of the NIB at the time the macro instruction is issued, a set of session parameters is found and provided with the logon. If the LOGMODE field contains a logon mode name, the named set of session parameters is found. If the LOGMODE field contains zeros or blanks, the default set of session parameters is found in the appropriate table. If the CLSDST with OPTCD=PASS involves a logical unit in another domain, the named session parameters or default session parameters are found in logon mode tables in that domain and are passed with the logon to the domain in which the receiving application program is located.

Logon Mode with Automatic Logon and VARY LOGON

When automatic logon has been specified for a device-type logical unit, the session parameters are determined by the method in which the automatic logon was generated: (1) by naming a controlling application program and (2) by naming a controlling application program by a VARY LOGON command.

Specification of the LOGAPPL operand in the LU statement causes the application program named in that operand to become the controlling application program for that logical unit. Whenever the logical unit is active and is not connected to another application program, ACF/VTAM automatically logs the logical unit on to the controlling application program. For the initial connection to that program and for each reconnection (after another program has disconnected the logical unit), the session parameters are the default parameters. (A controlling application program can relinquish control over a logical unit by issuing a CLSDST macro instruction with OPTCD=RELEASE.)

The network operator can change the controlling application program (or establish one) by issuing a VARY LOGON command for the logical unit. That command names another application program that is to become the controlling application program. The command must be issued in the domain that owns the logical unit, but the new controlling application program can be in another domain. (Note that a VARY LOGON command cannot be issued to specify automatic logon for an application program.) A LOGMODE operand can be specified in the VARY LOGON command, and inclusion or omission of that operand controls the session parameters that are associated with the logon. If the LOGMODE operand is included in the command, the logon mode name in that operand identifies the session parameters in the logical unit's logon mode table that are to be used for the first connection and subsequent reconnections with the new controlling application program. If the LOGMODE operand is omitted from the command, the logon mode name (if any) in the preceding VARY command is used to find the session parameters, or if no logon mode name has been supplied previously for the logical unit, the default parameters are used.

How the Primary Application Program Processes Session Parameters

As indicated in the preceding paragraphs, a set of session parameters is associated with each logon that reaches an application program. Those parameters (but not the logon mode name that might have been used to find the parameters) are available for inspection by the application program when it begins to process the logon (that is, when execution of the application program's LOGON exit routine begins).

The application program has complete control over whether the session parameters received with the logon or other session parameters are to be the ones sent to the logical unit in the next step of the connection process. That next step is to issue an OPNDST macro instruction with OPTCD=ACCEPT, which causes ACF/VTAM to generate a Bind command and send it to the logical unit. The Bind command contains the set of session parameters (see Figure 5-6) that the primary application program has designated to be included in the command.

Macro Instruction to Be Used	Desired Action	Setting of BNDAREA and/or LOGMODE Field When Macro Is Issued[1]	Restrictions or Qualications on Desired Action
REQSESS	Include named set of session parameters in the logon created by the macro.	LOGMODE=*logon mode name*	
	Include default set of session parameters in the logon created by the macro.	LOGMODE=0 *or* blanks[2]	
SIMLOGON or CLSDST with OPTCD=PASS	Include named set of session parameters in the logon created by the macro.	LOGMODE=*logon mode name*	
	Include default set of session parameters in the logon created by the macro.	LOGMODE=0 *or* blanks[2]	
INQUIRE with OPTCD=SESSPARMS	Get the session parameters associated with the logon for the logical unit being processed.	LOGMODE=0	The-e must be a pending logon for tne logical unit.
	Get a named set of session parameters from the logon mode table associated with the logical unit.	LOGMODE=*logon mode name*	Named logical unit must be in the same domain as the one in which the macro instruction is issued.
	Get the default set of session parameters from the logon mode table associated with the logical unit.	LOGMODE=blanks[2]	Named logical unit must be in the same domain as the one in which the macro instruction is issued.

OPNDST with OPTCD=ACCEPT	Use the session parameters associated with the logon to build the Bind command.	BNDAREA=0
		LOGMODE=0	

	Use a named set of session parameters from the logon mode table associated with the logical unit to build the Bind command.	BNDAREA=0
		LOGMODE=*logon mode name*
			Can only be used when the logical unit and its associated logon mode table are in the same domain as the one in which the macro instruction is issued.

	Use the default set of session parameters from the logon mode table associated with the logical unit to build the Bind command.	BNDAREA=0
		LOGMODE=blanks[2]
			Can only be used when the logical unit and its associated logon mode table are in the same domain as the one in which the macro instruction is issued.

	Use the contents of the bind area as the session parameters in the Bind command.	BNDAREA=*bind area address*
		LOGMODE=anything (Ignored)	

OPNDST with OPTCD=ACQUIRE	Use a named set of session parameters from the logon mode table associated with the logical unit to build the Bind command.	BNDAREA=0
		LOGMODE=*logon mode name*	

	Use the default set of session parameters from the logon mode table associated with the logical unit to build the Bind command.	BNDAREA=0
		LOGMODE=0 *or* blanks[2]	

	Use the contents of bind area as the session parameters in the Bind command.	BNDAREA=*bind area address*
		LOGMODE=anything (Ignored)	

[1] In all cases, the NAME field of the NIB must contain the symbolic name of the logical unit for which the desired action is to be taken and with which the logon mode table (if used) is associated. Where no BNDAREA specification is shown, the BNDAREA field is not involved in the operation. When a logon mode name is specified in LOGMODE, the name is resolved into an actual set of session parameters in the domain of tt.e secondary logical unit involved in the operation. Therefore, the logon mode name (if specified) must exist in a logon mode table associated with the secondary logical unit or in the system default logon mode table *in the domain* of the secondary logical unit.

[2] To get blanks, specify LOGMODE=C'

Figure 5-6. Setting NIB Fields to Acquire or Control Session Parameters

Two fields in the NIB associated with an OPNDST macro instruction play roles in determining which session parameters are sent in the Bind command. Those fields are the BNDAREA field and the LOGMODE field.

Effect of the BNDAREA Field on Session Parameters in a Bind Command

The BNDAREA field of the NIB associated with an OPNDST macro instruction can be used to provide the address of an area (called the *bind area*) within the application program in which the program can build a set of session parameters to be sent in the Bind command.

IBM provides a DSECT (named ISTDBIND) that can be used to set up session parameters in the bind area and can be used by an application program to examine session parameters. The DSECT is described in Appendix J of *ACF/VTAM Macro Language Reference*. The DSECT is provided as part of the system macro library (source statement library in DOS/VS and SYS1.MACLIB in OS/VS).

In determining which session parameters to include in a Bind command, ACF/VTAM always examines the BNDAREA field of the NIB first. If that field contains an address, the session parameters starting at that address are used in the command. If the BNDAREA field contains zeros, the setting of the LOGMODE field of the NIB controls which session parameters are sent in the Bind command.

Effect of the LOGMODE Field on Session Parameters in the Bind Command

When used with an OPNDST macro instruction, the LOGMODE field of the NIB provides a mechanism for designating which set of session parameters is to be included in the Bind command. The field can be used in different ways, depending on whether the connection request involves a logical unit in the same domain or in a different domain from the primary application program.

Using LOGMODE When the Logon Is from a Logical Unit in the Same Domain: When the primary application program knows that a logical unit is in the same domain, the primary program has the following options in using the LOGMODE field to specify the session parameters for the Bind command:

- Allow the session parameters associated with the logon to be incoporated into the Bind command. To do this, the application program sets the BNDAREA field of the NIB to 0 and sets the LOGMODE field to 0 before issuing the OPNDST macro instruction.
- Indicate that the default session parameters are to be sent in the Bind command. To do this, the application program sets the BNDAREA field of the NIB to 0 and sets the LOGMODE field to blanks before issuing the OPNDST macro instruction.
- Designate that a particular named set of session parameters from the logon mode table associated with the logical unit is to be sent in the Bind command. To do this, the application program sets the BNDAREA field of the NIB to 0 and puts a logon mode name in the LOGMODE field before issuing the OPNDST macro instruction.

In processing a logon from a logical unit in the same domain, the primary application program can also build a set of session parameters in the bind area and can specify that those parameters are to be sent. To do this, the application program puts the address of the bind area into the BNDAREA field of the NIB before issuing the OPNDST macro instruction. The presence of an address in the BNDAREA field causes ACF/VTAM to use the session parameters in the bind area and to ignore the setting of the LOGMODE field.

Using LOGMODE When the Logon Is from a Logical Unit in Another Domain: When the primary application program knows that a logon involves a logical unit in another domain, the primary application program has only one option in using the LOGMODE field to designate the session parameters to be used in the Bind command. That option is to designate that the session parameters that accompanied the logon are to be incorporated without change into the Bind command. To designate that, the primary application program sets the BNDAREA field to 0 and the LOGMODE field to 0 before it issues the OPNDST with OPTCD=ACCEPT.

In processing a logon from a logical unit in another domain, the alternative to setting the LOGMODE field to 0 is to build a set of session parameters in a bind area and to put the address of that area in the BNDAREA field of the NIB before issuing the OPNDST OPTCD=ACCEPT macro instruction.

Handling Session Parameters When the Logon Could Be from the Same Domain or Another Domain

In many cases, the primary application program does not know whether the logon involves a logical unit in its own domain or in a different domain. In this case, there are three standard actions the primary application program can take to designate the session parameters to be included in the Bind command:

1. Always accept and use the session parameters that accompany the logon. (Set the BNDAREA field to 0 and the LOGMODE field to 0 before issuing the OPNDST macro instruction.)
2. Always build a set of session parameters in a bind area and designate that those parameters are to be sent in the Bind command. (Put the address of the bind area into the BNDAREA field before issuing the OPNDST macro instruction.)
3. Examine (using an INQUIRE macro instruction) the session parameters that accompany the logon and possibly modify them in a bind area before issuing the OPNDST macro instruction.

The user may want to adopt one of the above options as a convention for any application program that can receive a mixture of same-domain and cross-domain logons.

Using the INQUIRE Macro Instruction to Get Session Parameters

The INQUIRE macro instruction can be used in several ways to get a set of session parameters. At any point in a program, the macro instruction can be issued to get a named set of session parameters or the default parameters from the logon mode table associated with a logical unit owned by the domain in which the macro instruction is issued. The session parameters are acquired from the logon mode table associated with the logical unit whose symbolic name is in the NAME field of the NIB. The setting of the LOGMODE field of the NIB identifies the particular set of parameters to be taken from that logon mode table. See Figure 5-6.

When a program is processing a logon, the INQUIRE macro instruction can be used to get the set of session parameters that accompanied the logon. To do this, the program puts the name of the logical unit in the NAME field of the NIB, sets the LOGMODE field of the NIB to 0, and issues the INQUIRE macro instruction. The session parameters that accompanied the logon can be acquired in this way regardless of whether the logon applies to a logical unit in the same domain or in a different domain. Acquisition of the session parameters enables the primary application program to inspect them and to decide whether those parameters or different ones should be sent to the logical unit when the OPNDST macro instruction is issued.

When the INQUIRE macro instruction is executed, ACF/VTAM finds the indicated session parameters and places them in the area of storage pointed to by the AREA field of the RPL. The AREALEN field of the RPL must specify the length of the storage area in which the session parameters (and any user logon message) are to be placed. To do this, the INQUIRE macro instruction can be issued twice. For the first INQUIRE, the AREALEN field is set to 0. This INQUIRE is completed with RTNCD=0 and FDBK2=5 (insufficient length), and RECLEN indicates the required length. Then, the INQUIRE is issued again, with the AREALEN field set to the correct length.

Specifying Session Parameters When Acquiring Connection

When a primary application program issues an OPNDST macro instruction with OPTCD=ACQUIRE to acquire a connection with a logical unit, the settings of the BNDAREA and LOGMODE fields of the NIB control the session parameters sent in the Bind command. See Figure 5-6. Examples of using the LOGMODE field and the BNDAREA field to control session parameters are provided next.

Example 1: Using Session Parameters Associated with a Logon

Assume that a logical unit named LU1 has sent a logon. Receipt of the logon causes scheduling of the LOGON exit routine. Coding near the beginning of the exit routine might be as follows:

```
                    .
                    .
                    .
INQ1      INQUIRE   RPL=RPL1,OPTCD=SESSPARM
                    (Test for RTNCD=0 and FDBK2=5 in the RPL)
                    (Load value in RECLEN field of the RPL into register 7)
                    .
                    .
                    .
          MODCB     AM=VTAM,RPL=RPL1,AREALEN=(7)
                    (Test return codes from execution of the MODCB macro)
                    .
                    .
                    .
INQ2      INQUIRE   RPL=RPL1,OPTCD=SESSPARM
                    .
                    .
                    .
                    (Checks session parameters placed in AREA1 and determines
                    that they are appropriate)
                    .
                    .
                    .
          OPNDST    RPL=RPL1
                    .
                    .
                    .
RPL1      RPL       AM=VTAM,NIB=NIB1,AREA=AREA1,AREALEN=0
NIB1      NIB       NAME=LU1,LOGMODE=0,BNDAREA=0
AREA1     DS        14F
```

The INQUIRE macro instruction at INQ1 attempts to get the session parameters with the RPL's AREALEN field set to 0. This causes failure of the macro instruction with the FDBK2 field set to 5 (insufficient length). Upon return from the macro instruction, however, the RPL's RECLEN field contains the number of bytes needed for the session parameters (and any user logon message). The required length is loaded into register 7, and the MODCB macro instruction is issued to put that value into the AREALEN field of the RPL. Then, at INQ2, the INQUIRE macro instruction is issued again, causing ACF/VTAM to put the session parameters into AREA1. (The session parameters are those that were received with the logon.) The coding checks the session parameters and determines that they are appropriate for the logical unit and for the type of session the application program will have with that unit. Therefore, the application program issues the OPNDST macro instruction, using the NIB whose BNDAREA and LOGMODE fields are set to 0. The zero in the LOGMODE field tells ACF/VTAM to use the session parameters associated with the logon to build the Bind command to be sent to the logical unit. (A large value of 14F is shown in the example for the amount of storage reserved for AREA1, and that may not be the correct value for your application program. The value should be equal to the maximum size of the session parameters and any user logon message.)

Example 2: Building and Using Session Parameters in a Bind Area

Assume that an application program wants to initiate a session with a logical unit named LU2 in the same domain. A logon mode table was defined and was identified in the MODETAB operand of the LU definition statement for LU2. The application program wants to get the default session parameters from the logon mode table, modify them, and then send the modified parameters to the logical unit in the Bind command when it acquires the logical unit. The coding could look like this:

```
          .
          .
          .
          INQUIRE   RPL=RPL2,OPTCD=SESSPARM
          .
          .
          .
          (Instructions test and modify the session parameters in SPAREA2)
          .
          .
          .
          MODCB     AM=VTAM,NIB=NIB2,BNDAREA=SPAREA2
          OPNDST    RPL=RPL2,OPTCD=ACQUIRE
          .
          .
          .
RPL2      RPL       AM=VTAM,NIB=NIB2,AREA=SPAREA2,
                    AREALEN=SPLEN
NIB2      NIB       NAME=LU2,LOGMODE=C' ',BNDAREA=0
SPAREA2   DS        XL(SPLEN)
SPLEN     EQU       BINUSE-ISTDBIND
```

Because the NIB's LOGMODE field contains blanks, the INQUIRE macro instruction causes the the default entry in the logon mode table to be moved into SPAREA2. The application program then modifies the session parameters to fit the way it wants to communicate with LU2. The MODCB macro instruction puts the address of SPAREA2 in the NIB's BNDAREA field. When the OPNDST macro instruction is executed, the modified session parameters are transmitted to LU2 in the Bind command.

How the Secondary Application Program Processes Session Parameters Received in a Bind Command

When a Bind command is received by a secondary application program, the program's SCIP exit routine is scheduled. When execution of the exit routine starts, the fourth word of the parameter list passed to the exit routine contains the address of the session parameters received in the command. Using that address, the exit routine can find the session parameters in the Bind request unit and examine those parameters (using the ISTDBIND DSECT).

If the session parameters are acceptable, the secondary application program issues an OPNSEC macro instruction. If the session parameters are not acceptable, the secondary application program rejects the Bind command by issuing a SESSIONC macro instruction with operands specifying a negative response. For an illustration of this process, see Figure C-15 in Appendix C.

Disconnection

When a primary application program has finished communicating with a logical unit, the program can disconnect it. The logical unit is then available for use by other programs. The program can reconnect it at some later time. There are several conditions that call for disconnecting a logical unit:

- The logical unit has logged off by issuing a character-coded logoff or Terminate command. This causes the LOSTERM exit routine to be entered. The LOSTERM exit routine disconnects the logical unit.
- A terminal that has logged on to the application program might use a prearranged logoff message that the terminal operator issues when finished using the program. Each input message must be checked to see if it is a logoff message. When a logoff message arrives, the terminal is disconnected and any storage pools are updated.
- Communication with the terminal is finished; there is no more data to send or receive. Here, the program has probably acquired the logical unit and is not expecting a logoff message. The program determines when communication is finished, and the logical unit should be disconnected.
- Another program has requested connection to one of your logical units. Your program can surrender the logical unit to the requesting program.
- An error or special condition occurs in relation to a logical unit. The program can disconnect the logical unit while continuing to service other logical units connected to the program. For more information, see Chapter 9, "Handling Errors and Special Conditions."

How a Primary Application Program Disconnects Logical Units

The CLSDST macro instruction is used by a primary application program to disconnect a specific logical unit. The logical unit to be disconnected can be identified either by its symbolic name or by its communication identifier (CID). When the identification is to be by the symbolic name, the RPL specified in the CLSDST macro instruction points to a NIB that contains the symbolic name. When the identification is to be by CID, the logical unit's CID must be in the ARG field of the RPL specified in the CLSDST macro instruction.

CLSDST Using a Symbolic Name

A logical unit can be disconnected by using the logical unit's symbolic name contained in a NIB. This method is used when the CID is not available or when it is more convenient to use the symbolic name than to use the CID. The symbolic name is normally used in these circumstances:

- In a LOGON exit routine when the exit routine has determined that it does not want to issue an OPNDST macro instruction to establish a session with the logical unit. To reject the logical unit's request for a session, the exit routine must issue a CLSDST macro instruction. Because no OPNDST macro instruction has been issued, no CID for the logical unit is available, and the CLSDST must use the symbolic name in the NIB.

- In a LOGON exit routine or elsewhere in a program when an OPNDST macro instruction has failed. If the logon is still pending (found by checking the logon-still-queued flag in the failing OPNDST's NIB) and the program does not want to issue another OPNDST macro to make another attempt to establish the session, the program should issue the CLSDST macro instruction to clean up unnecessary control information in ACF/VTAM. Here, as above, the failure of the OPNDST macro instruction means that no CID for the logical unit is available, and the CLSDST must use the symbolic name in the NIB.
- In a RELREQ exit routine where the symbolic name of the logical unit, rather than the CID, is provided in the parameter list that is available upon entry to the exit routine.
- In a main program to cancel a SIMLOGON macro instruction before the simulated logon has been processed. Here again, no CID is available and the CLSDST must use the symbolic name in the NIB.

The procedure for disconnecting by means of the symbolic name is: (1) be sure the NAME field of a NIB contains the symbolic name of the logical unit to be disconnected, (2) set the NIB field of the RPL to the address of the NIB that contains the symbolic name, and (3) issue the CLSDST macro instruction, specifying the RPL that points to the NIB. For example:

```
        CLSDST   RPL=RPL1
        .
        .
        .
RPL1    RPL      ACB=ACB1,AM=VTAM,NIB=NIB1
NIB1    NIB      NAME=LU1
```

CLSDST Using a CID

If the application program has just connected the logical unit or has just completed an input or output operation with the logical unit, the CID of the logical unit is available in the ARG field of the RPL that was used for the operation. For example, if the logical unit has sent a logoff message to the application program, the RPL used to read the message contains the CID of the logical unit. To disconnect the logical unit, issue a CLSDST macro instruction that specifies the same RPL:

```
        CLSDST   RPL=(1)
```

There are other sources of the CID:

- When the LERAD or SYNAD exit routine is scheduled, register 1 contains the address of the RPL that was used for the request that failed. That register and RPL can be used to disconnect the logical unit.
- When a DFASY, RESP, or SCIP (for other than a Bind command) exit routine is scheduled, the CID of the session from which the request or response was received is in the second word of the parameter list that ACF/VTAM makes available to the exit routine when the exit routine is scheduled. In this case, the MODCB macro instruction can be used to move the CID to the RPL to be used to disconnect the logical unit.

Storage Management at Disconnection

If you have been using the storage management techniques presented in Chapter 3, remember to replace the storage elements in their pools when you disconnect a logical unit.

Disconnecting All Logical Units at One Time

When an application program is finished processing and is to disconnect all logical units, it can either (1) issue a separate CLSDST macro instruction for each logical unit or (2) issue a CLOSE macro instruction and allow ACF/VTAM to disconnect the logical units.

Use of the CLOSE macro instruction, which eventually closes the ACB, causes ACF/VTAM to issue a synchronous CLSDST macro instruction for each logical unit to which the program is connected. The synchronous CLSDST macro instructions are executed one after another, which is slower than asynchronous execution in which processing of the CLSDST macro instructions would be overlapped. If the application program issues the CLSDST macro instructions itself, it can designate that the macro instructions are to be executed asynchronously. Thus, while issuing separate CLSDST macro instructions in the application program requires more coding, it results in faster execution than issuing the CLOSE macro instruction and having ACF/VTAM disconnect the logical units.

How a Secondary Application Program Requests Disconnection

In a session between two application programs, the secondary application program, instead of the primary application program, may take the initiative to end the session because the secondary program is the first to recognize that communication is finished or because the user may have established a convention by which the secondary program is required to inform the primary program when the session is to be ended. The secondary application program has two ways of initiating action to end the session. It can either (1) send a Request Shutdown (RSHUTD) command to the primary application program, or (2) issue a TERMSESS macro instruction, which causes the primary application program's LOSTERM exit routine to be scheduled.

Requesting Disconnection with a Request Shutdown Command

The sending of a Request Shutdown (RSHUTD) command (if allowed by the session's FM profile) is the way a secondary application program can send a disconnection request to a primary application program without involving ACF/VTAM in the notification process. The command goes from one application program to the other without being recognized or acted upon by ACF/VTAM. To the primary application program, the command represents a request to disconnect the secondary program as soon as possible. This allows the primary application program to continue communications with the secondary program, including the exchange of normal-flow messages, in order to do any cleanup operations that are necessary.

The Request Shutdown command is transmitted as an expedited-flow message. Because of this, the command cannot be used unless the primary application program has either a DFASY exit routine or an outstanding RECEIVE macro instruction with RTYPE= DFASY. When the DFASY exit routine is scheduled or when the RECEIVE is completed, the primary application program must determine that it has received a Request Shutdown command (rather than some other expedited-flow command) by checking the CONTROL field of an RPL. In the case of the DFASY exit routine, the exit routine checks the CONTROL field of the read-only RPL that is available to the exit routine.

After receipt of the command, the primary application program should issue a CLSDST macro instruction for the session as soon as possible.

Requesting Disconnection with the TERMSESS Macro Instruction

An alternative way for a secondary application program to request disconnection is to issue a TERMSESS macro instruction. The CID in the ARG field of the RPL or the symbolic name in the NAME field of the NIB can be used to identify the primary application program from which the secondary program wants to be disconnected.

The TERMSESS macro instruction specifies (in its OPTCD operand) whether the disconnection is to be conditional or unconditional. ACF/VTAM converts the macro instruction into a Terminate command (either conditional or unconditional, according to the OPTCD operand in the macro instruction). When the Terminate command reaches the primary end of the session, the primary application program's LOSTERM exit routine is scheduled. The fourth word of the parameter list passed to the exit routine contains decimal reason code 20 for an unconditional Terminate and decimal reason code 32 for a conditional Terminate.

A conditional TERMSESS macro instruction leaves disconnection of the secondary application program entirely at the discretion of the primary application program. The primary application program can issue the CLSDST macro instruction immediately or it can perform cleanup operations (including exchange of normal-flow messages with the secondary program) before it issues the macro instruction.

An unconditional TERMSESS macro instruction causes ACF/VTAM to immediately terminate the session. The primary application program should still issue a CLSDST macro instruction for the secondary program, but most of the disconnection processing will have been completed by the time it issues that macro instruction. No communication with the secondary application program is possible after the primary application program has been notified.

Chapter 6. Communicating with Logical Units

This chapter contains these major sections:

An introduction to communicating with logical units

Using ACF/VTAM to communicate

Using SNA protocols

How to communicate with non-SNA 3270 terminals as logical units

Appendix A discusses use of ACF/VTAM to communicate with BSC terminals, start-stop terminals, and (optionally) local non-SNA 3270 terminals.

An Introduction to Communicating with Logical Units

ACF/VTAM uses Systems Network Architecture (SNA) concepts to establish communications between an ACF/VTAM application program and a logical unit (including another ACF/VTAM application program). (Some of the basic SNA concepts are described in Chapter 1 of this book.) This chapter provides a general description of communication facilities.

Before learning how to communicate with logical units, it is necessary to understand some fundamental concepts about communicating with logical units.

Who Is Communicating: The ACF/VTAM Application Program and Logical Units

Both an ACF/VTAM application program and the logical units with which it communicates can contain program logic. This fact implies these general characteristics of communication between ACF/VTAM application programs and logical units:

- The design and coding of those parts of a logical unit and those parts of an ACF/VTAM application program that communicate with each other must be coordinated. In some cases, for example, both the application program and the logical unit may be designed by the same person; perhaps one is designed first and the other designed to complement it. This is a probable approach for application programs designed to serve a particular kind of logical unit (for example, a 3600 logical work station). Or the application program can be designed as a standard program with which all logical units must conform, and logical units can be required to meet the application program's interface. In either case, both ends of the communication must be coordinated.
- The existence of program logic in a terminal or cluster controller makes it possible to remove work from the host computer. The data that is exchanged between an application program and a logical unit may vary considerably, depending on what data processing (including the addition and deletion of device-control and format characters and data editing) can be performed by the logical unit rather than by the application program in the host computer.

What Is Communicated: Messages and Responses

An ACF/VTAM application program and a logical unit exchange messages and responses to messages. A *message* normally contains data; in addition to or instead of data, a message can contain control information (described in ACF/VTAM publications as *commands* or *indicators*). A *response* normally contains information about whether a particular message arrived and was processed successfully or unsuccessfully; in addition, it contains certain control information (commands or indicators). As explained later in this chapter, a response does not have to be returned for every message; it is possible for an application program and a logical unit to communicate without either side ever sending a response. (A *message* corresponds to a SNA request unit [RU] and associated request header [RH] indicators. A *response* corresponds to a SNA response unit and associated response header.)

Figure 6-1 illustrates this exchange of messages and responses between an application program and a logical unit.

What a Message Contains

A message contains:

Data

A command or indicator

Combinations of the above (for example, data and an indicator)

Data consists of information that is sent from or received in an ACF/VTAM application program's input/output area. Since both an application program and a logical unit contain program logic, each has the ability to insert, interpret, and strip off information before forwarding it to a terminal operator, to a recording medium, or to some other destination.

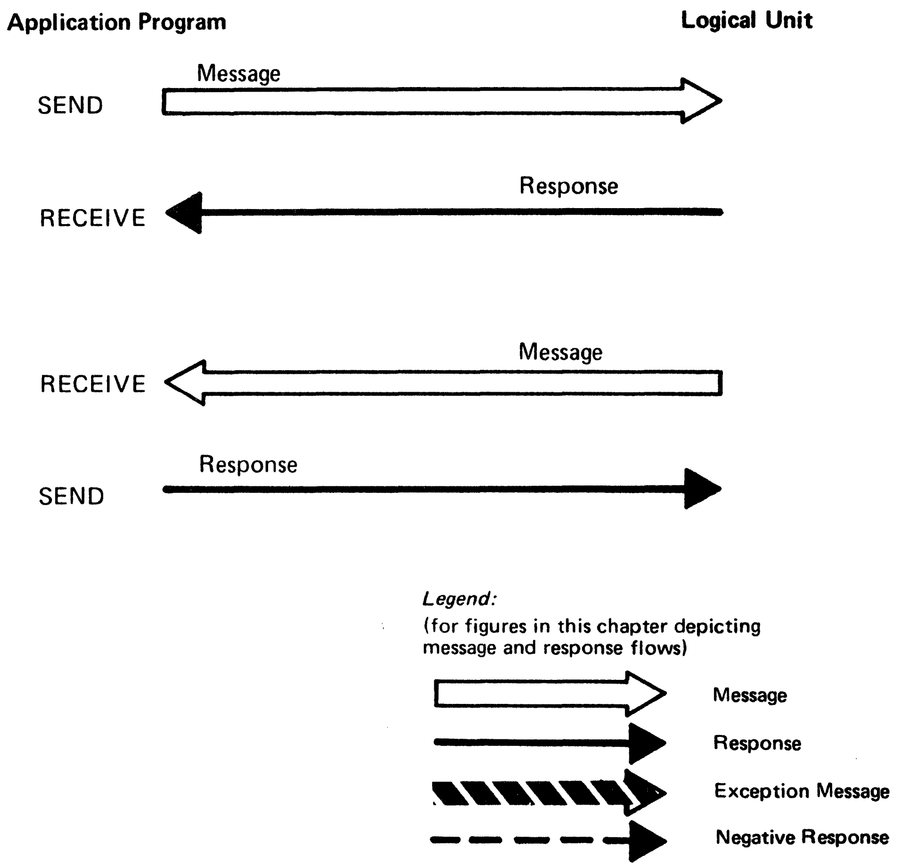


Figure 6-1. Exchange Messages and Responses

Here is an example of data that might be exchanged between an application program and a logical unit. An application program receives input from a connected logical unit as the result of issuing a RECEIVE macro instruction. When the RECEIVE is completed, the input area specified in the AREA operand of the RECEIVE contains data. For example, after completion of a RECEIVE, the input area might contain data in this format:

Code	Account Number	Amount Deposited

The code was either typed in by an operator at a terminal associated with the logical unit or it was inserted by the logical unit based on its analysis of the operator's input. The code is interpreted by the application program as a request for passbook update processing, and control is passed to the routine that handles that processing. The application program might prepare a data reply in this format:

Code	Account Number	Amount Deposited	New Balance

It sends the reply to the logical unit with a SEND macro instruction, specifying the output area in the AREA operand. Any device-control or format information required to print the message at a printer or keyboard-display unit is furnished by the logical unit when the message arrives.

In addition to the transaction data, the logical unit can also send certain control indicators. For example, the application program and the logical unit may be using change-direction indicators to ensure that only one of them at a time is sending (this method of communication is described in more detail later in this chapter). On receiving the message that contains data, the application program also checks the change-direction field of the RPL associated with the completed RECEIVE request:

```
    TESTCB      RPL=(2),CHNGDIR=CMD
```

The TESTCB macro tests whether a Change Direction Command indicator is part of the message. If not, the program prepares to receive a further message. If the indicator is in the message, the program can send the reply. When a data reply, such as the passbook update reply shown above, is prepared, the program can indicate in the reply that the next message is to come from the logical unit. To do this, the program includes a Change Direction Command indicator by specifying CHNGDIR=CMD in the SEND used to send the reply.

Note that only data is sent from or received in an I/O area of the application program; all indicators, commands, and response information are sent by being specified symbolically and received by being detected in appropriate fields of the RPL.

Certain commands and indicators can be sent only in messages that do not contain data. Examples of these indicators and commands are explained later in this chapter.

What a Response Contains

A response to a message contains information about the success or failure of transmission and processing of a particular message. In sending a message, the ACF/VTAM application program or logical unit specifies the circumstances under which it expects a response to the message. When sending a response from an ACF/VTAM application program, commands or indicators are specified symbolically in a SEND macro instruction. When receiving a response, response information and control commands are available in appropriate fields of the RPL associated with the completed RECEIVE or in a read-only RPL provided by ACF/VTAM on scheduling the ACF/VTAM application program's RESP exit routine.

Definite, Exception, or No Response Indication: In the example above under "What a Message Contains," the message sent by the logical unit (requesting a passbook update) might have contained an indicator requesting that:

- No response be returned, whether the message arrived and was processed successfully or not (no response requested)
- A response be returned only if the message contained a transmission error or could not be processed successfully (negative response requested)
- A response be returned, whether the message arrived with or without error and was processed successfully or not (definite response requested)

A request for no response is feasible if the logical unit has its own means of determining failure of the message's transmission, such as using a timer or assuming that the terminal operator will resend the message if there is no reply to it from the host computer after a certain length of time. In these cases, neither ACF/VTAM nor the host application program sends a response, because the logical unit is not prepared to receive it.

Frequently, a logical unit will request that a response (called an *exception response* or *negative response*) be returned only if the message is not received and processed successfully. If the message is received and processed successfully, no response is returned by the application program. However, if the message is not received successfully, ACF/VTAM indicates this in a return code and in additional information provided in RPL fields upon completion of the RECEIVE; the application program sends a negative response. Even when the message arrives successfully, the ACF/VTAM application program for its own reasons (for example, because it discovers the format of the message is improper) can send back a negative response, using the SEND macro instruction with STYPE=RESP. The negative response is indicated by specifying RESPOND=EX; additional information can be provided by using the SSENSEO, SSENSMO, and USENSEO fields of the RPL.

If the passbook update message above is received and processed successfully and a definite response was requested, the ACF/VTAM application program sends a positive response, using a SEND macro instruction and specifying STYPE=RESP (a response) and RESPOND=NEX (positive). If some messages require a definite response and others do not, the application program determines whether to send a response by testing for a NEX indication in the RESPOND field of the RPL associated with a completed RECEIVE.

Figure 6-2 illustrates the logical unit requesting (A) that a response be returned in either case and (B) that a response be returned only if the message does not arrive or is not processed successfully.

Again, referring to the passbook update example, on sending the message that it prepares after performing the passbook update, the ACF/VTAM application program can specify whether it wants no response, a response only if the message is unsuccessful, or a response regardless of what happens to the message. This is done by specifying an appropriate indication in the RESPOND operand of the SEND macro instruction. If a response is requested, it is received by the application program either by a RECEIVE that specifies RTYPE=RESP or by ACF/VTAM's scheduling the program's RESP exit routine. When the RESPOND field of the SEND RPL is set to NEX and the SEND macro instruction includes the POST=RESP option, the SEND is not completed until the response is

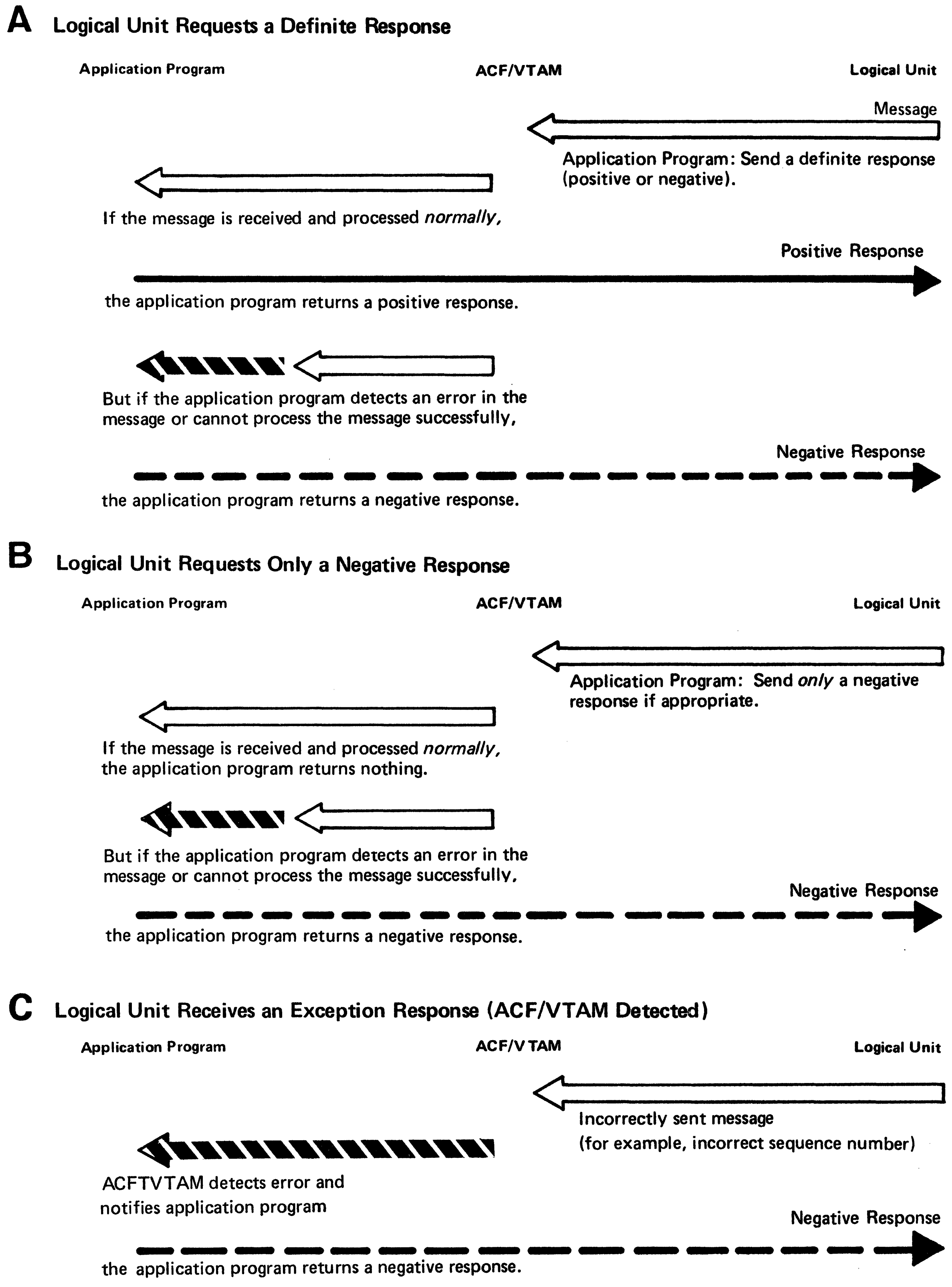

A Logical Unit Requests a Definite Response
Application Program
ACF/VTAM
Logical Unit
Message
Application Program: Send a definite response (positive or negative).
If the message is received and processed *normally,*
Positive Response
the application program returns a positive response.
But if the application program detects an error in the message or cannot process the message successfully,
Negative Response
the application program returns a negative response.
B Logical Unit Requests Only a Negative Response
Application Program
ACF/VTAM
Logical Unit
Application Program: Send *only* a negative response if appropriate.
If the message is received and processed *normally,* the application program returns nothing.
But if the application program detects an error in the message or cannot process the message successfully,
Negative Response
the application program returns a negative response.
C Logical Unit Receives an Exception Response (ACF/VTAM Detected)
Application Program
ACF/VTAM
Logical Unit
Incorrectly sent message
(for example, incorrect sequence number)
ACFTVTAM detects error and notifies application program
Negative Response
the application program returns a negative response.

Figure 6-2. A Logical Unit (A) Requests a Definite Response, (B) Requests Only a Negative Response, and (C) Receives an Exception Response

received. In this case, a RECEIVE is not used to obtain the response; the response information is available in the SEND RPL when the operation is complete.

Definite Response 1 and 2 Indication: In addition to the positive versus negative aspect of responses, there is another aspect to any response: whether it is response type 1 (formerly FME response) or response type 2 (formerly RRN response). Every response, independent of its positive or negative aspect, is designated by its sender as a response type 1, response type 2, or both. The meanings of the types of responses are agreed upon by the application program and the logical unit involved in the communication and may be determined by SNA protocols.

The application program indicates on each message whether it expects a definite response 1, a definite response 2, or both, to be returned. Combining these types of responses with the positive/negative response types described above yields seven possible combinations of response types that can be indicated for a given message:

Return a definite response 1 (either positive or negative)

Return a definite response 2 (either positive or negative)

Return definite responses 1 and 2 (either positive or negative)

Return only a negative response 1

Return only a negative response 2

Return only negative responses 1 and 2

Return no response of any kind

When definite responses 1 and 2 are requested, the responses are returned together in the same response; they are not returned separately. Similarly, when an error occurs and negative responses 1 and 2 are requested, the negative responses are returned together.

The logical unit, like the application program, also specifies for each message the types of responses it wants.

The user should be aware that SNA protocols dictate when responses should be requested and what responses are returned.

Specifying Special Handling of the Response to a Normal-Flow Message or Command: In special circumstances, a programmer may want the response to a normal-flow message or command to be handled as if it were an incoming normal-flow message from the logical unit. To accomplish this the programmer must have specified PROC=ORDRESP in the NIB at connection and must specify RESPOND=QRESP in the RPL when the message or command is sent. For information on how such responses are handled by ACF/VTAM, see "Controlling the Handling of Normal-Flow Responses" later in this chapter. In contrast to specifying RESPOND=QRESP, the programmer can specify RESPOND=NQRESP, thus telling ACF/VTAM to handle the response in the regular manner as a normal-flow response.

The Three Key Elements in a RESPOND Operand: When a program is sending a message or command and is specifying the type of response it wants to receive, it makes the specifications either by setting the RESPOND fields of the RPL before issuing the macro instruction or by specifying parameters in the RESPOND operand when the macro instruction is issued. There are three potential parameters that can be specified in the RESPOND field or operand:

1. Nature of response desired:

NEX	for positive or negative response
EX	for negative response only

2. Type of response:

FME	for response type 1
RRN	for response type 2
FME,RRN	for response types both 1 and 2

3. Handling of a normal-flow response when PROC=ORDRESP was set in the NIB:

NQRESP	for regular handling
QRESP	for handling as though the response was a normal-flow message coming from the logical unit

Thus, an example of a RESPOND operand in which all three parameters are specified is:

RESPOND=(NEX,FME,NQRESP)

These parameters indicate that a positive or negative response is to be returned; the response is to be type 1; and the response is not to receive any special handling by ACF/VTAM.

How Messages and Responses Are Exchanged

Messages and responses are exchanged by using SEND and RECEIVE macro instructions in an ACF/VTAM application program and by similar instructions in a logical unit. Using SEND/RECEIVE communication, messages can be sent simultaneously by the application program and by the logical unit. Messages that contain certain commands can be sent ahead of messages that contain data or other commands. Messages can be queued and responses correlated by using sequence numbers (a sequence number is automatically assigned to each message). This flow of data, commands, and responses between an ACF/VTAM application program and a logical unit can be synchronized, if necessary, by stopping the flow, resetting sequence numbers at one or both ends of the message exchange, and then restarting the flow. These concepts are described below.

The SEND and RECEIVE Macro Instructions

The ACF/VTAM application program sends and receives messages and responses by using the SEND and RECEIVE macro instructions. (A logical unit uses its own corresponding instructions to send and receive.) Some messages containing commands and some responses can be received by an ACF/VTAM application program by having ACF/VTAM schedule an exit routine designed to handle these commands (a DFASY exit routine) and responses (a RESP exit routine); alternatively, they can be handled by a RECEIVE with RTYPE=DFASY or RTYPE=RESP specified.

If required by the transmission services profile in the session parameters, the sending and receiving of messages and responses between the ACF/VTAM application program and a logical unit cannot begin until a Start Data Traffic (SDT) command has been sent from the primary end of the session to the secondary end of the session. When required, the SDT command must be sent at the beginning of a session, and it must be sent within a session if the message flow is to be restarted after it was stopped (with a Clear command). At the beginning of a session, the Start Data Traffic command is sent either by ACF/VTAM or the application program, depending on how the SDT field of the NIB was set when the OPNDST macro instruction was issued. If the SDT field indicated SYSTEM, the SDT command is sent by ACF/VTAM as part of the OPNDST processing. If the SDT field indicated APPL, the SDT command must be sent by the application program using the SESSIONC macro instruction. To resume message flow after it has been stopped, the application program sends the SDT command by using the SESSIONC macro. SESSIONC can also be used to halt the flow of messages and responses by specifying CONTROL=CLEAR.

If the secondary end of the session is an application program, a response to that SDT command can be sent by ACF/VTAM or the secondary application program, depending on how the SDT field of the NIB is set when the OPNSEC macro instruction is issued. If the SDT field indicates APPL, the response must be sent by the secondary application program (using a SESSIONC macro instruction).

If the secondary end of the session is a device-type logical unit, ACF/VTAM automatically responds to the SDT command.

Normal-Flow and Expedited-Flow Messages and Responses

Messages that contain data, messages that contain certain commands (called *normal-flow commands*), and the responses to such data messages and commands form the *normal-flow traffic* between an application program and a logical unit. Normal-flow traffic is handled separately from the expedited-flow traffic described below. Normal-flow messages and commands are sent sequentially, one after the other, through the network, and a normal-flow message or command that is sent before another message or command arrives sooner. Figure 6-3 illustrates this principle. Similarly, responses to normal-flow messages and commands keep their order as they travel through the network; a normal-flow response sent before another normal-flow response arrives before the second response. Note, however, that ACF/VTAM *does not maintain the exact sequence relationship between messages and responses in relation to each other*; that is, a response sent by a logical unit after a message may be presented to the application program before the message. The only way an application program can be sure of receiving normal-flow messages and responses in the exact order in relation to each other as they were sent by the logical unit is by specifying RESPOND=QRESP (and POST=SCHED) in the RPL used to send the message or command (see "Controlling the Handling of Normal-Flow Responses" below). (The reader should also note that use of authorized path in OS/VS2 MVS affects the order in which asynchronous operations are completed, and that because

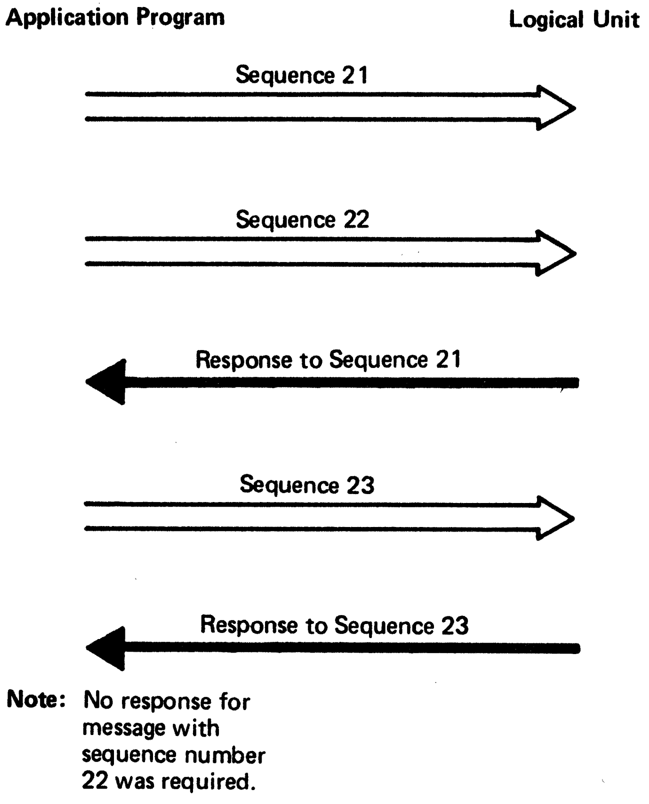


Figure 6-3. Normal-Flow Messages Are Sent Sequentially

of this, the sequences in which messages are received may be affected [see "Coding Considerations for OS/VS2 MVS Authorized Path" in Chapter 3].)

Messages that contain certain other commands (called *expedited-flow commands*) and responses to those commands are sent in a separate flow from the normal-flow messages and responses; these form the *expedited-flow traffic* in the network. Only one of these commands can be sent at a time by the application program; a response must be received to one expedited-flow command before another can be sent. The expedited-flow commands tell the receiver to do something that has higher priority than receiving normal-flow messages; for example, to stop sending normal-flow messages or to prepare to shut down communication with the other end of the session. Because of this, ACF/VTAM sends an expedited-flow command immediately—ahead of any normal-flow traffic that may be waiting to be sent. Figure 6-4 illustrates how ACF/VTAM gives priority to expedited-flow traffic.

The messages and responses that are sent on the normal flow and the expedited flow are listed in Figure 6-5. The responses to normal-flow data messages are also transmitted on the normal flow (and can be called *normal-flow responses*), and the responses to expedited-flow commands are transmitted on the expedited flow (and can be called *expedited-flow responses*).

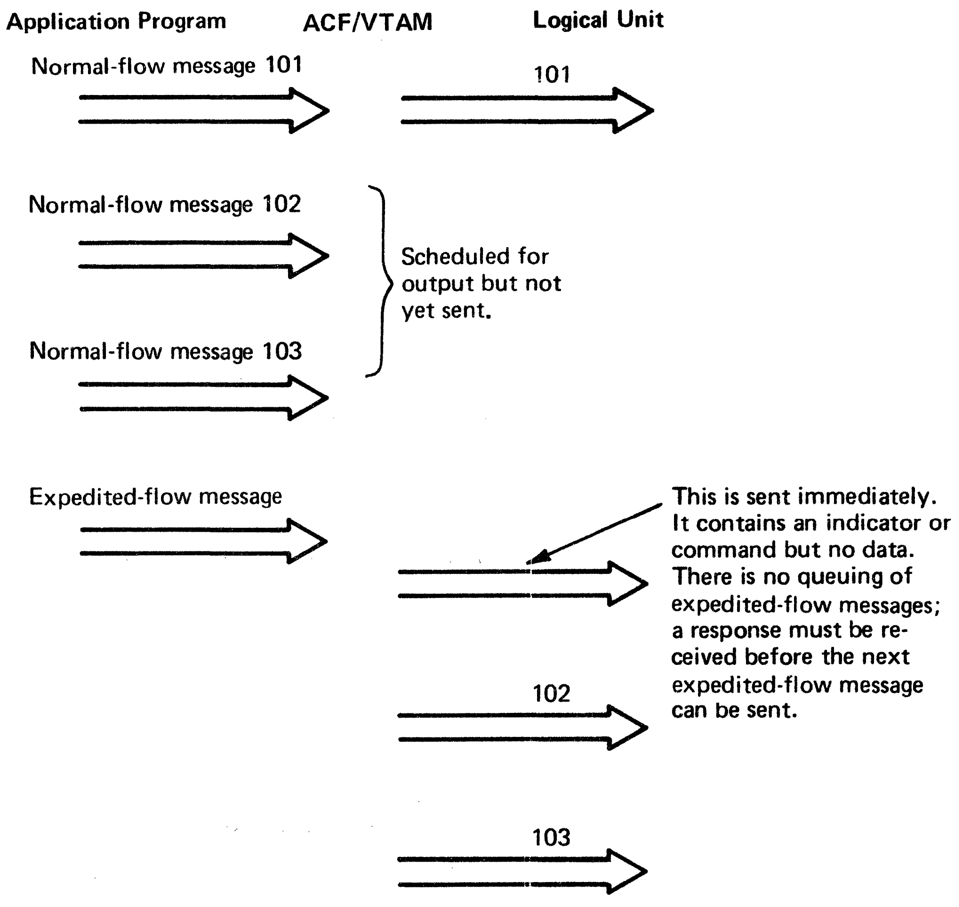

Application Program
ACF/VTAM
Logical Unit
Normal-flow message 101
101
Normal-flow message 102
Scheduled for output but not yet sent.
Normal-flow message 103
Expedited-flow message
This is sent immediately. It contains an indicator or command but no data. There is no queuing of expedited-flow messages; a response must be received before the next expedited-flow message can be sent.
102
103

Figure 6-4. The Difference between Normal-Flow and Expedited-Flow Messages

Normal Flow	Expedited Flow
Data messages	Quiesce at End of Chain (QEC) command
	Release Quiesce (RELQ) command
Bid command	Request Shutdown (RSHUTD) command
Cancel command	Shutdown (SHUTD) command
Chase command	Shutdown Complete (SHUTC) command
Logical Unit Status (LUS) command	Signal command
Quiesce Complete (QC) command	Stop Bracket Initiation (SBI) command
Ready to Receive (RTR) command	
Bracket Initiation Stopped (BSI) command	

Figure 6-5. Messages and Responses Transmitted on the Normal Flow and on the Expedited Flow

Controlling the Handling of Normal-Flow Responses: The macro instruction that sends a normal-flow message can be used to control how ACF/VTAM will handle the response to that message. The ability to exercise that control depends on whether PROC= NORDRESP or PROC=ORDRESP was specified in the NIB when connection was established.

If PROC=NORDRESP was in effect at the time of connection, the programmer has no control over how ACF/VTAM handles the responses. In this case, all normal-flow responses (regardless of the QRESP setting) are handled as responses, exactly as they are handled in VTAM Level 2. Thus, PROC=NORDRESP is specified in the NIB when a user wants the application program to be executed in ACF/VTAM as it is executed in VTAM Level 2.

If PROC=ORDESP was in effect in the NIB at connection, the programmer establishes, at the time the normal-flow message is sent, the way in which ACF/VTAM handles the response, as follows:

- When the message is sent with RESPOND=NQRESP in the RPL, the response is handled as an ordinary normal-flow response—meaning that it can cause completion of a POST=RESP operation, will cause scheduling of a RESP exit routine, and will cause completion of a RECEIVE RTYPE=RESP.
- When the message is sent with RESPOND=QRESP, the response is not handled as a response, but instead is handled as though it were an incoming normal-flow message from the logical unit—a DFSYN response. This means the "response" will not cause scheduling of a RESP exit routine and will not cause completion of a RECEIVE with RTYPE=RESP. It will, however, cause completion of the original SEND operation if the operation specified POST=RESP. If POST=RESP was not specified in the original operation, the application program can get the response by using a RECEIVE with RTYPE=DFSYN and checking the RTYPE field of the RPL upon completion. If the RTYPE field after completion contains RTYPE=(DFSYN,RESP), the program knows it has received a normal-flow response instead of a normal-flow message.

The key distinction between a NQRESP response and a QRESP response is this: the NQRESP response is handled as a regular normal-flow response and is presented to the application program in sequence with other normal-flow responses; the QRESP response is treated as an incoming normal-flow message and is presented to the application program in sequence with those messages. A response that satisfies a SEND which specifies POST=RESP and *either* QRESP or NQRESP, is always delivered immediately by ACF/VTAM and thus may get out of order with other normal-flow responses.

Note that when a program sends a normal-flow message on a session established with the ORDRESP NIB option, the POST operand in the macro instruction can be set to SCHED or RESP, and the completion of the macro instruction will be based on that setting. In VTAM Level 2, when a normal-flow command (as opposed to a data message) is sent, VTAM ignores the POST operand and automatically establishes POST=RESP (meaning that the operation is not completed until the response has been received). Similarly if ORDRESP is specified, the application program must specify the correct RESPOND setting for normal-flow commands (normally NEX,FME); whereas, in VTAM Level 2 and for NORDRESP, this value of RESPOND is assumed.

An application program will send most of its normal-flow messages with RESPOND= NQRESP. QRESP is used only for particular purposes, as described under "The Chase Command" and "Bracket Protocol" later in this chapter.

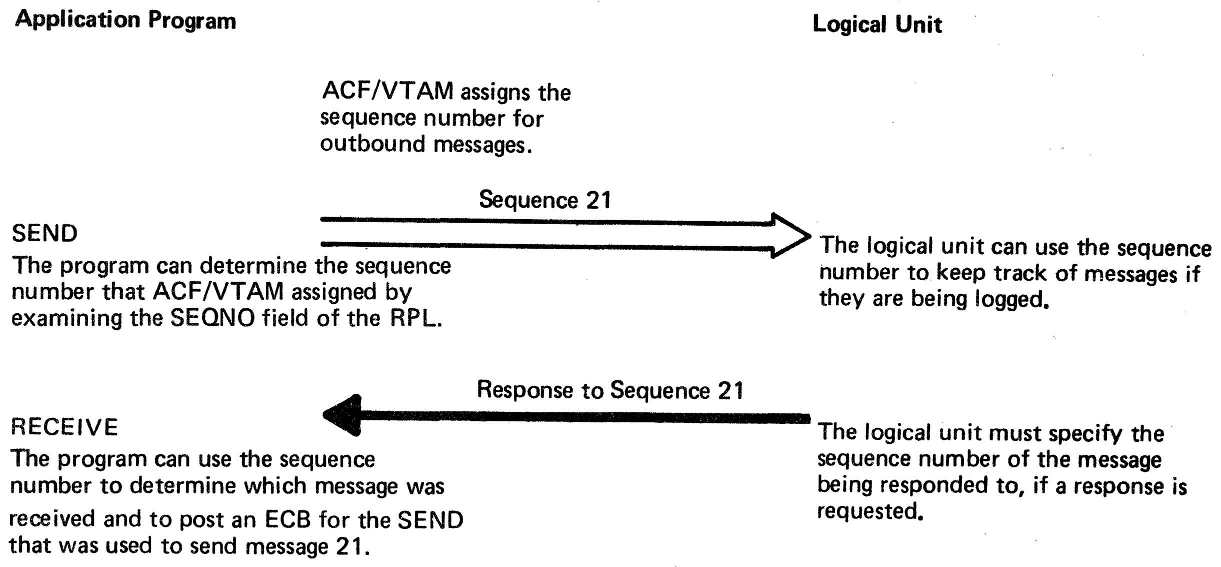
Sequence Numbers

In a session, each normal-flow message sent to a logical unit is assigned a sequence number by ACF/VTAM. To the primary application program, this number is known as the *outbound sequence number*, but to the logical unit, the number is known as the *inbound sequence number*. The numbering begins with 1 for the first normal-flow message sent after connection, and the number is increased by 1 for each subsequent message. This process continues until the logical unit is disconnected, unless sequence numbers are reset during the session (see "Controlling Flow" later in this chapter). (ACF/VTAM also assigns an identification number to each expedited-flow message it sends in a session, but those numbers are handled separately from the normal-flow sequence numbers.)

Similarly, the logical unit assigns a sequence number to each normal-flow message it sends to the application program. The numbering begins with 1, and the number is increased by 1 for each subsequent normal-flow message the logical unit sends. To the logical unit, this number is known as the *outbound sequence number*. To the primary application program and the ACF/VTAM that services that program, the number is known as the *inbound sequence number*. ACF/VTAM checks the inbound sequence numbers on the normal-flow messages it receives from a logical unit. Should a message arrive out of sequence (that is, its sequence number is not 1 greater than that of the last normal-flow message received), ACF/VTAM considers this to be a transmission error and indicates to the application program that an out-of-sequence message has been received.

When a normal-flow response is sent (either a positive or a negative response), the sender assigns to it the sequence number of the message being responded to. This provides the sender with a means of matching the response with its message. For example, an application program can send a group of messages, with each message indicating that only an exception response should be returned. Should an exception response be returned, the application program can use the sequence number to determine where in the group the error occurred. Sequence numbers are also useful for logical units that log each message that is received or sent. Figure 6-6 illustrates how sequence numbers are used. Later examples in this chapter show more specific examples of their use.

The SEQNO field of the RPL is used to convey sequence numbers between ACF/VTAM and the application program. The application program can determine the sequence number that ACF/VTAM assigned to an outbound message by checking the SEQNO field after completion of the SEND macro. For an inbound message or response, the application program determines the sequence number that was contained in the message or response by examining the SEQNO field after completion of the RECEIVE macro. To



ACF/VTAM assigns the sequence number for outbound messages.

The program can determine the sequence number that ACF/VTAM assigned by examining the SEQNO field of the RPL.

The logical unit can use the sequence number to keep track of messages if they are being logged.

The program can use the sequence number to determine which message was received and to post an ECB for the SEND that was used to send message 21.

The logical unit must specify the sequence number of the message being responded to, if a response is requested.

Figure 6-6. How Sequence Numbers Are Used

assign a sequence number to an outgoing response, the application program puts the sequence number into the SEQNO field before issuing the SEND macro.

Controlling Flow

The ACF/VTAM application program can start and stop the flow of all messages and responses between itself and a logical unit. In most cases, the flow begins when the application program sends a *Start Data Traffic* (SDT) command to the logical unit at the beginning of a session. Depending on how the SDT field of the NIB is set, the SDT command may be sent automatically by ACF/VTAM as part of the OPNDST processing, or it may have to be sent by the application program. The flow of messages and responses is stopped when the application program sends a *Clear* command to the logical unit. This not only prohibits any further transmission of messages and responses, but also causes the sequence numbers of the logical unit and ACF/VTAM to be reset to 0. The first data message or normal-flow command sent is assigned the sequence number 1. The Clear command also causes all incoming and outgoing data messages, responses, and commands in the network pertaining to the session but not yet received to be discarded. The Clear command is sent whenever it is needed to stop the flow of data and to clean up traffic flowing in the session. (Sometimes ACF/VTAM automatically sends a Clear.) When the Clear command is sent by the application program in the middle of a session, the flow can be restarted with the SDT command. The flow of messages and responses can be started and stopped any number of times, as illustrated in Figure 6-7.

The Set and Test Sequence Numbers (STSN) Command: Another command sent with the SESSIONC macro instruction is called the *Set and Test Sequence Numbers* (STSN) command. This command allows the application program to reset the normal-flow sequence numbers and to communicate with a logical unit to establish the proper sequence numbers. An attempt to resynchronize sequence numbers can begin when the application program or the logical unit recognizes that the sequence number of a message it has received is not 1 greater than the sequence number of the previous message it received. When the logical unit recognizes the sequence number error, it sends the *Request Recovery* (RQR) command to ask the application program to take recovery action. When this command is received by the application program, its SCIP exit routine is scheduled.

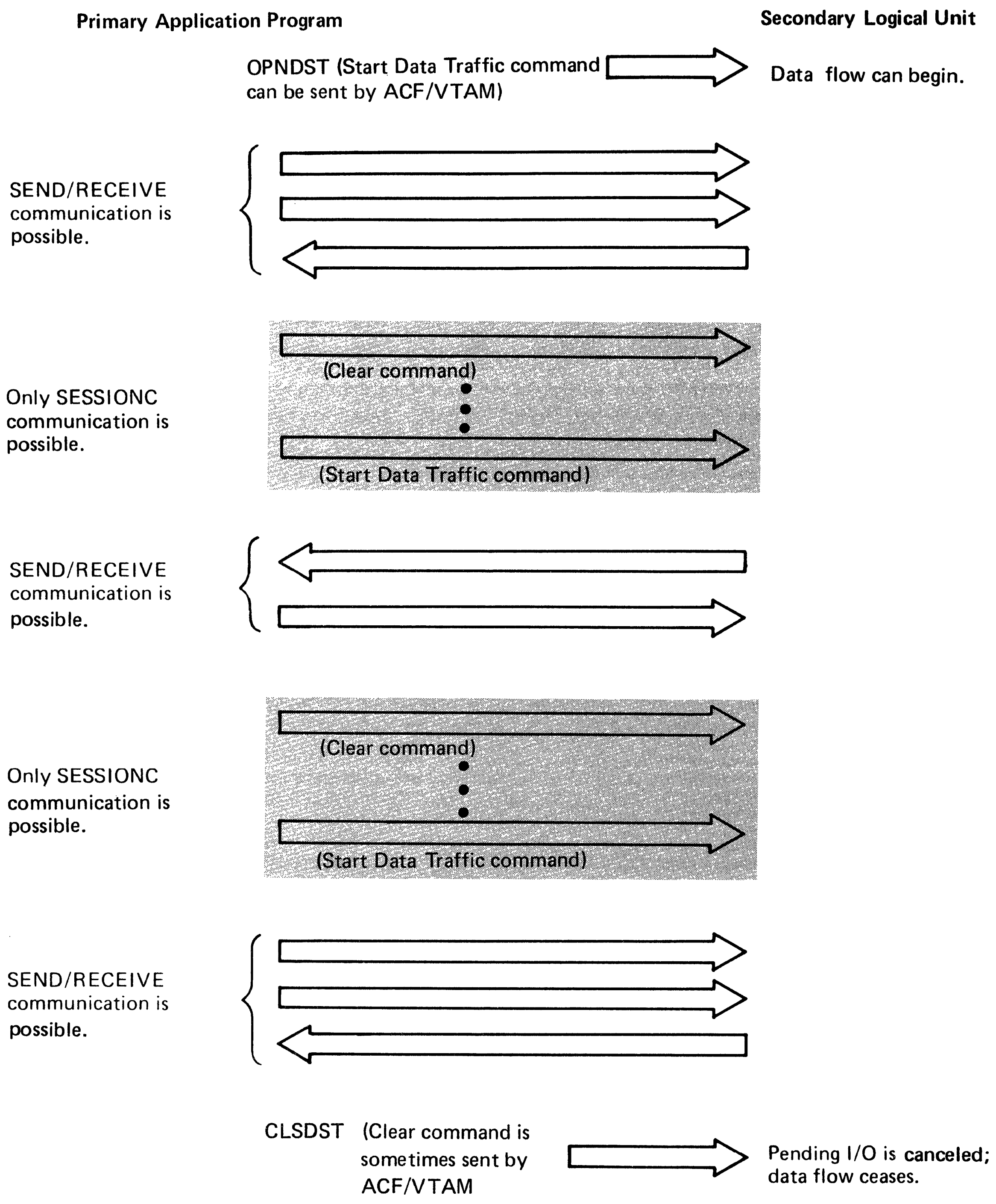

Primary Application Program
Secondary Logical Unit
OPNDST (Start Data Traffic command can be sent by ACF/VTAM)
Data flow can begin.
SEND/RECEIVE communication is possible.
Only SESSIONC communication is possible.
(Clear command)
(Start Data Traffic command)
SEND/RECEIVE communication is possible.
Only SESSIONC communication is possible.
(Clear command)
(Start Data Traffic command)
SEND/RECEIVE communication is possible.
CLSDST (Clear command is sometimes sent by ACF/VTAM
Pending I/O is canceled; data flow ceases.

Figure 6-7. Starting and Stopping the Flow of Messages and Responses

The primary application program normally uses the following procedure to resynchronize sequence numbers with the logical unit:

1. The application program issues the SESSIONC macro instruction with CONTROL= CLEAR to stop the message flow and to remove all undelivered messages and responses pertaining to its session that are still in the network.
2. The application program then issues the SESSIONC macro instruction with CONTROL=STSN to question the logical unit about normal-flow sequence numbers. With this macro instruction, the application program can send sequence number values to the logical unit and, from the response, determine whether the logical unit "agrees" with those numbers. Or, the application program can request the logical unit to return whatever values it considers to be the correct sequence numbers. Or, the application program can tell the logical unit to set its sequence numbers to particular values. To reach agreement with the logical unit, the application program may have to send several STSN commands, with the logical unit responding to each command. When agreement is finally reached, either the logical unit or the application program, or both, may have to return to a previous point in their operations and resend one or more messages.
3. After agreement on sequence numbers is reached, the application program issues the SESSIONC macro instruction with CONTROL=SDT to restart the flow of messages and responses.

For examples of the use of the SESSIONC macro instruction with CONTROL=STSN, see Figures C-3, C-9, and C-18 in Appendix C.

Another use of Set and Test Sequence Number commands is for restarting message flow, where the ACF/VTAM application program, having taken periodic checkpoints of messages that it was sending to a logical unit, wants to inform the logical unit of the sequence numbers at which it is restarting after a system, session, application program or logical unit failure.

The Chase Command: The Chase command can be used by an application program at any point in its processing to ensure that the program has received all responses from the other end of the session. When the other end of the session receives the Chase command, it must send any unsent response to previous messages or normal-flow commands before it sends the response to the Chase command. Thus, when the sender receives the response to the Chase command, the sender knows there are no outstanding responses for that session.

The Chase command is frequently used by an application program before a session termination command. For example, a secondary application program that has received a Shutdown command might issue a Chase command to get any outstanding responses from the primary application program before issuing the Shutdown Complete command. (See Figure C-12 in Appendix C.) Or, the Chase command can be issued by a primary application program before it issues a CLSDST macro instruction.

Using the Chase command may cause a problem. When a response cannot be passed immediately to the application program, it is placed on a queue to await presentation to the program. If the program sends a Chase command with POST=RESP, the operation is posted complete as soon as the response to the Chase command is received by ACF/VTAM. If any responses were previously queued for the program, it may not be prepared to process them, having interpreted the respons to the Chase command as an indication that all responses were received.

To avoid this problem, the session should be established with PROC=ORDRESP specified in the NIB and, the Chase command should be sent with a macro instruction that specifies POST=SCHED and RESPOND=QRESP:

```
SEND      RPL=RPL1,STYPE=REQ,CONTROL=CHASE,POST=SCHED,
          RESPOND=(FME,NEX)
```

The response to the Chase is thus handled in order with respect to other normal-flow responses. Note that if any outstanding responses might have the QRESP indicator on, then the Chase must also be sent with RESPOND=(NEX,FME,QRESP) to ensure that the Chase response will be received in order with other responses having QRESP on.

For more information on the QRESP and NQRESP parameters, see "Specifying Special Handling of the Response to a Normal-Flow Message or Command" under "What a Response Contains" earlier in this chapter.

Identifying Logical Units

When an application program receives a logon from a logical unit and before it connects the logical unit, the application program has available to it the logical unit's user-supplied name. This is an 8-byte symbolic name created for the logical unit during ACF/VTAM definition. (When the logon is processed by a LOGON exit routine, one word in the parameter list passed to that exit routine points to the symbolic name. When the logon satisfies an outstanding OPNDST macro with OPTCD=(ACCEPT,ANY), the NAME field of the NIB [pointed to by the RPL] contains the symbolic name.)

After connection is established with the logical unit (that is, when the OPNDST with OPTCD=ACCEPT is completed), the application program is also provided with a 4-byte ACF/VTAM-supplied identification (called a communication identifier, or CID) for the session with the particular logical unit that has been connected. After the OPNDST is completed, the CID is in the ARG field of the RPL used for the connection. The application program uses the CID for all I/O requests issued in the specific-mode (all SEND requests and all RECEIVE requests specifying OPTCD=SPEC).

When a RECEIVE macro instruction issued in the any-mode is completed, ACF/VTAM provides the identity of the logical unit that sent the data. Since the application program will probably communicate with the logical unit in the specific-mode, it is the CID, rather than the symbolic name, that ACF/VTAM supplies to the application program. (This CID is provided in the ARG field of the RPL used with the RECEIVE macro.) Should the identity be significant, the application program has three ways to relate the CID to the logical unit's symbolic (user-supplied) name:

- The application program can use an INQUIRE macro to translate the CID into a symbolic name.
- The application program can maintain a table of CIDs and their symbolic equivalents.
- When the application program establishes connection with the logical unit, the application program can initially assign a 4-byte value to the logical unit (by putting the value in the USERFLD field of the NIB), and ACF/VTAM returns the value each time that logical unit's data satisfies a RECEIVE. The 4-byte value can be anything the application program chooses to associate with the logical unit. It can be used to identify the logical unit, or it can contain the address of a subroutine that is to handle that logical unit's data.

Using ACF/VTAM to Communicate with Logical Units

Using ACF/VTAM to communicate with logical units requires an understanding of these major alternatives:

- Having ACF/VTAM perform an operation synchronously or asynchronously with respect to execution of the ACF/VTAM application program (OPTCD=SYN|ASY specified in the SEND or RECEIVE macro instruction)

- For asynchronous operations, having ACF/VTAM post an ECB or having it schedule an exit routine when the operation is completed (OPTCD=ASY and either ECB=*address* or EXIT=*address* specified in the SEND or RECEIVE macro instruction)
- Having ACF/VTAM schedule a message to be sent or to send it and confirm its arrival (POST=SCHED|RESP on a SEND)
- Having a message from *any* connected logical unit put in an ACF/VTAM application program area or having a message from a *specific* logical unit put in a program area (OPTCD=ANY|SPEC on a RECEIVE)
- Having a logical unit be in continue-any mode or in continue-specific mode (OPTCD=CA|CS on any RPL-based macro instruction)
- Having a RECEIVE with RTYPE=DFASY satisfied or having a DFASY exit routine scheduled when a logical unit sends an expedited-flow (DFASY) message
- Having a RECEIVE with RTYPE=RESP satisfied or having a RESP exit routine scheduled when a logical unit sends a response
- Having ACF/VTAM retain or discard portions of an incoming message that is too long to fit in the program's unit area (PROC=KEEP|TRUNC)

Major Alternatives

Some of these alternatives are also discussed in Chapter 3, "Organizing a Program." Here they are discussed specifically in relation to communicating.

Synchronous versus Asynchronous Operations

Synchronous Requests: An ACF/VTAM application program can request that a communication operation be performed synchronously with relation to the execution of the program. For example:

```
SEND     RPL=(2),STYPE=REQ,AREA=AREA1,RESPOND=(NEX,FME),
         OPTCD=SYN,POST=RESP
```

This macro instruction requests that a message (STYPE=REQ) be sent from AREA1 with a response to be returned whether or not the message arrives and is processed successfully (RESPOND=(NEX,FME)). Execution of the ACF/VTAM application program is suspended because it has made a synchronous request (OPTCD=SYN), and the next instruction is not executed until ACF/VTAM has determined that the requested operation has been performed. In this case, however, the requested operation is the *scheduling* of a SEND (POST=SCHED) rather than the *actual transmission* (with ACF/VTAM receiving a response). In most cases, when scheduling is specified, ACF/VTAM returns control to the requesting program in a relatively short period of time; however, certain circumstances may cause a long delay. For example, posting may not occur until a pacing response is returned from the logical unit. The ASY option is therefore usually preferable.

Here is another example of a synchronous SEND:

```
SEND     RPL=(2),STYPE=REQ,AREA=AREA1,RESPOND=(NEX,FME),
         OPTCD=SYN,POST=RESP
```

For this SEND, the ACF/VTAM application program has to wait until ACF/VTAM receives a response to the message (POST=RESP). A program that communicates with only a few logical units and can wait for each communication request to be completed before doing any further processing might use this kind of synchronous operations; for most programs, however, this is not efficient.

Note that POST=RESP cannot be specified unless a definite response is requested; that is, no response (NEX,NFME or NEX,NFME,NRRN) or exception response only (EX,FME or EX,NFME,RRN) cannot be specified with POST=RESP, because ACF/VTAM would never know that the message had arrived.

Here is an example of a RECEIVE for input from a specific logical unit with OPTCD=SYN:

```
RECEIVE     RPL=(2),RTYPE=DFSYN,AREA=AREA1,AREALEN=100,
            OPTCD=(SYN,SPEC)
```

Here, execution of the ACF/VTAM application program is suspended until input arrives from the logical unit (whose CID is located in the RPL's ARG field). This is undesirable except in simple programs, perhaps where batch input is being received. It is also efficient enough if a message is already in ACF/VTAM buffers. This is true, for example, if the message received in ACF/VTAM's buffer is larger than the amounts of data read each time a RECEIVE is issued (and the KEEP option, described later in this chapter, is used).

Here is an example of a RECEIVE for input from any logical unit with OPTCD=SYN:

```
RECEIVE     RPL=(2),RTYPE=DFSYN,AREA=AREA1,AREALEN=200,
            OPTCD=(SYN,ANY)
```

Here, execution of the ACF/VTAM application program is suspended until input arrives from any connected logical unit that is not in continue-specific mode. This type of request is most likely to be used in a program that communicates with only a few logical units. It can also be used with a large number of logical units if response time is not important.

Asynchronous Requests: An ACF/VTAM application program can also request that a communication operation be performed asynchronously with relation to the execution of the program. For example:

```
SEND        RPL=(2),AREA=AREA1,STYPE=REQ,RESPOND=(NEX,FME),
            OPTCD=ASY,POST=SCHED,ECB=ECB1
```

This SEND requests that ACF/VTAM schedule the sending of the data from AREA1 to the logical unit. As soon as scheduling of the output has been completed, ACF/VTAM notifies the program either by posting an ECB (shown here) or by scheduling an RPL exit routine. (The relative advantages of posting ECBs and scheduling RPL exit routines are discussed in Chapter 3 and below.)

Rather than the scheduling, the actual sending of a message can be requested to be performed asynchronously with relation to the execution of the program. For example:

```
SEND        RPL=(2),AREA=AREA1,STYPE=REQ,RESPOND=(NEX,FME),
            OPTCD=ASY,POST=RESP,EXIT=RPLEXIT
```

This SEND requests that ACF/VTAM initiate sending of the message at AREA1 and immediately return control to the program. When ACF/VTAM receives a response indicating the success or failure of the transmission and processing, ACF/VTAM schedules an RPL exit routine at RPLEXIT. The program continues processing; the RPLEXIT exit routine automatically gets control when this operation is completed. Or, if ECB-posting is specified instead of the exit routine, the program continues processing—minus the time ACF/VTAM takes to get control and post the ECB—until it discovers the ECB is posted or until the program issues a WAIT or a CHECK macro instruction.

In general, synchronous operations are easier to program but inefficient with regard to the amount of processing that the program can do. Asynchronous operations are more difficult to program, but are required to handle communication with a reasonably large number of logical units.

Note: *If two or more asynchronous requests for the same logical unit are issued, ACF/VTAM may not process these requests in the same order in which they were issued. For example, if two SEND macro instructions specifying POST=RESP, OPTCD=ASY are executed for the same logical unit without an intervening CHECK, the second message can arrive at the logical unit before the first.*

ECB versus RPL Exit Routine

If asynchronous operations are requested, each request can specify that ACF/VTAM do either of two things when the operation is completed: (1) post an ECB or (2) schedule an RPL exit routine.

Here is an example of a SEND macro instruction that specifies that an ECB be posted upon completion:

```
SEND      RPL=(2),AREA=AREA1,STYPE=REQ,RESPOND=(NEX,FME),
          OPTCD=ASY,POST=RESP,ECB=ECB1
```

Figure 6-8 shows the sequence of events that might occur following the issuance of this macro instruction.

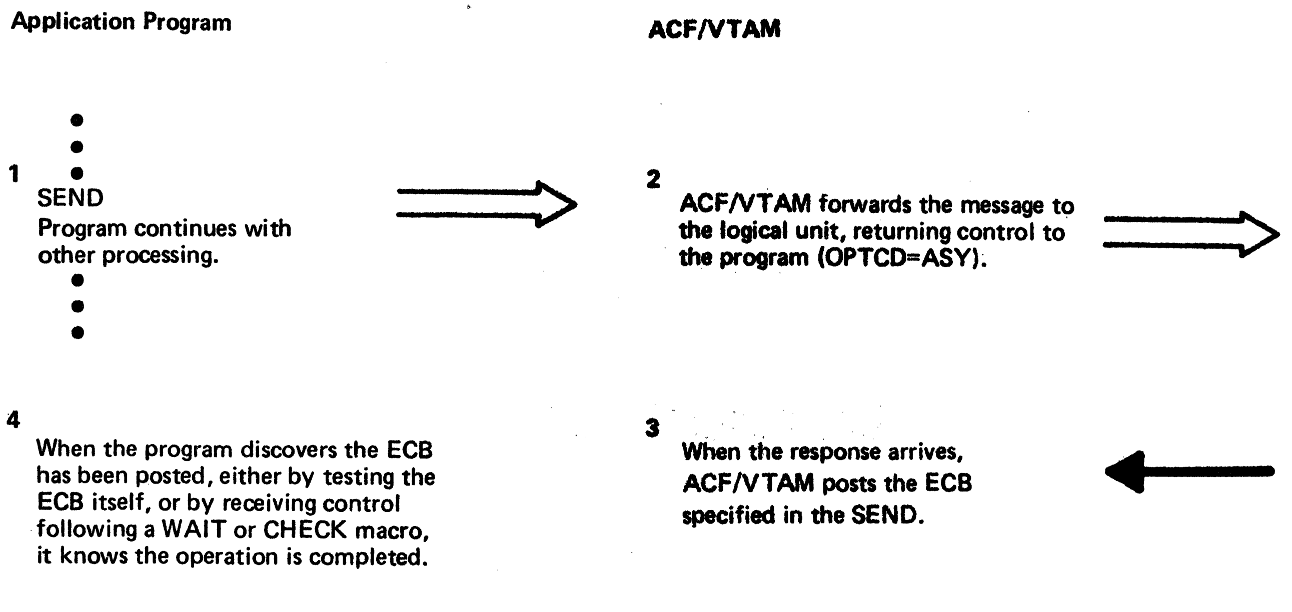
Here is an example of a SEND macro instruction that specifies that an RPL exit routine be scheduled upon completion:

```
SEND      RPL=(2),AREA=AREA1,STYPE=REQ,RESPOND=(NEX,FME),
          OPTCD=ASY,POST=RESP,EXIT=RPLEXIT
```

Figure 6-9 shows the sequence of events that might occur following the issuance of this macro instruction.

Scheduled versus Responded Output Operations

(This alternative is also discussed above as an example in "Synchronous versus Asynchronous Operations.")



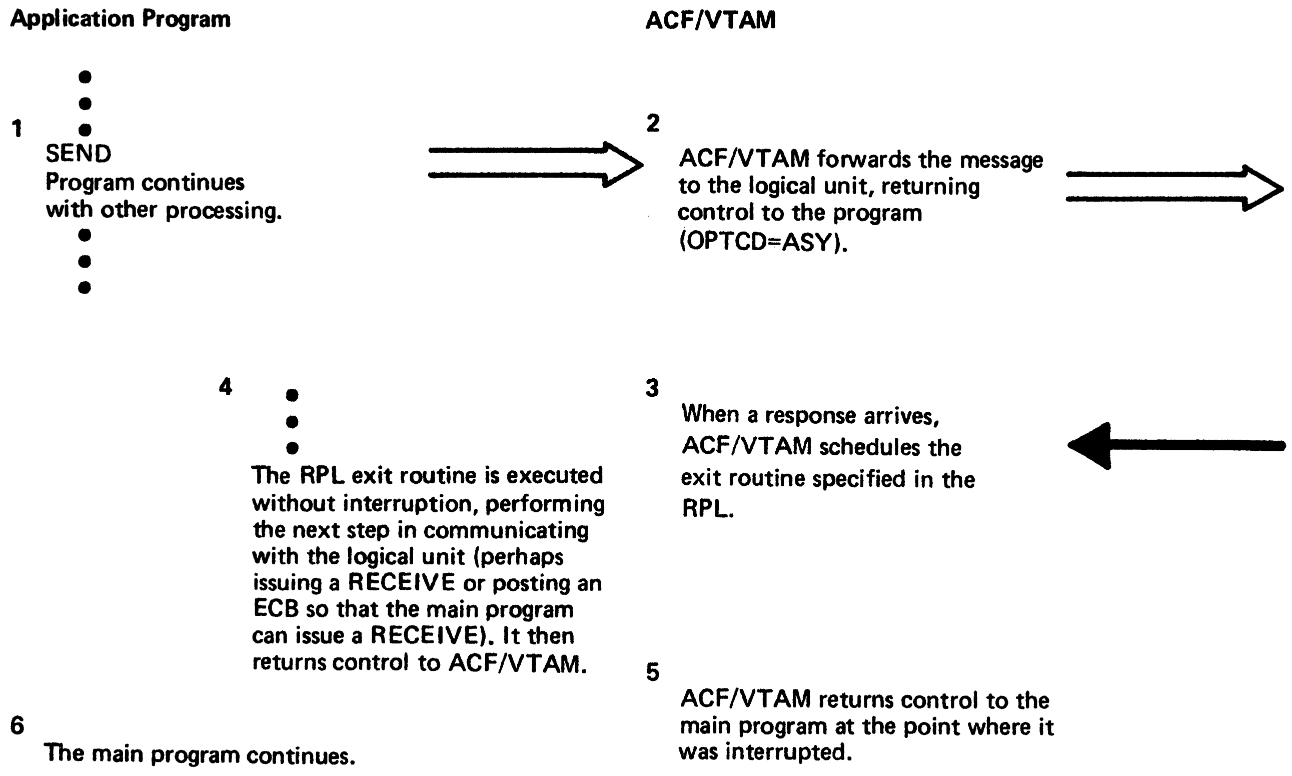
1 SEND
Program continues with other processing.

2 ACF/VTAM forwards the message to the logical unit, returning control to the program (OPTCD=ASY).

3 When the response arrives, ACF/VTAM posts the ECB specified in the SEND.

4 When the program discovers the ECB has been posted, either by testing the ECB itself, or by receiving control following a WAIT or CHECK macro, it knows the operation is completed.

Figure 6-8. The General Sequence of Events When ECB-Posting Is Specified


Application Program

ACF/VTAM

1 SEND
Program continues with other processing.

2 ACF/VTAM forwards the message to the logical unit, returning control to the program (OPTCD=ASY).

3 When a response arrives, ACF/VTAM schedules the exit routine specified in the RPL.

4 The RPL exit routine is executed without interruption, performing the next step in communicating with the logical unit (perhaps issuing a RECEIVE or posting an ECB so that the main program can issue a RECEIVE). It then returns control to ACF/VTAM.

5 ACF/VTAM returns control to the main program at the point where it was interrupted.

6 The main program continues.

Figure 6-9. The General Sequence of Events When an RPL Exit Routine Is Specified

The ACF/VTAM application program requests the sending of a message to a logical unit in one of two ways:

- The application program can indicate that as soon as the message has been scheduled for transmission and transferred to an ACF/VTAM buffer area, thus freeing the application program's output data area, ACF/VTAM is to consider the output operation completed (by returning control and either posting an ECB or scheduling an RPL exit routine, as specified in the output request). This is called *scheduled* output and is illustrated in Figure 6-10.
- The application program can indicate that ACF/VTAM is not to consider the operation completed until the message has been received by the logical unit and a response has been returned. This is called *responded* output and is illustrated in Figure 6-11.

Responded output is easier to use, but requires that the output data area not be reused until a response has been received by ACF/VTAM. If the response indicates that an error occurred, the data is still available for retransmission. Scheduled output allows the application program to send a series of messages that all use the same RPL and, possibly, the same output area. It also allows the program to decide whether or not a response to the message must be returned. If message chaining is used (discussed later in this chapter), a positive response is not required for every message that is sent.

With responded output, completion status information is returned as part of the operation. With scheduled output, the operation is completed when the message is scheduled, before any completion status information is available. To determine how the output operation was completed, the application program must issue an input request to obtain a response containing the completion status information. This is why the application program in Figure 6-10 issues three input requests in addition to the three output requests.

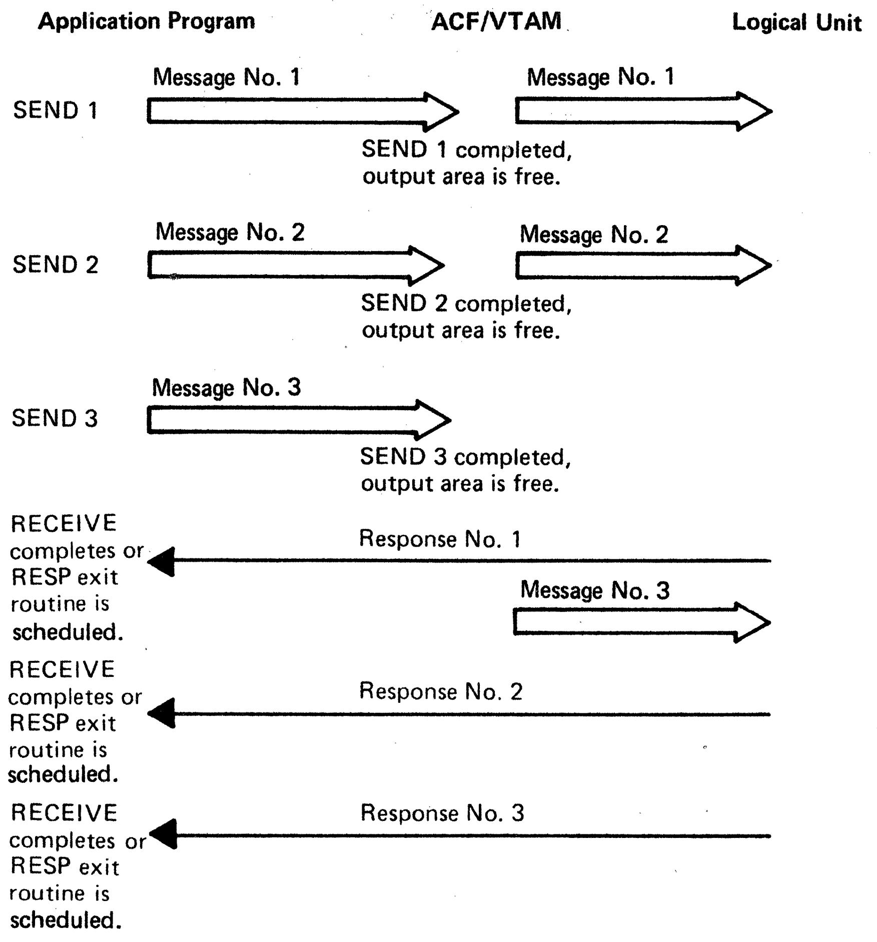

Application Program
ACF/VTAM
Logical Unit
SEND 1
Message No. 1
Message No. 1
SEND 1 completed, output area is free.
SEND 2
Message No. 2
Message No. 2
SEND 2 completed, output area is free.
SEND 3
Message No. 3
SEND 3 completed, output area is free.
RECEIVE completes or RESP exit routine is scheduled.
Response No. 1
Message No. 3
RECEIVE completes or RESP exit routine is scheduled.
Response No. 2
RECEIVE completes or RESP exit routine is scheduled.
Response No. 3

Figure 6-10. Scheduled Output

Receiving Input from any Logical Unit versus Receiving Input from a Specific Logical Unit

The ACF/VTAM application program can obtain data from a *specific* logical unit, or it can request data from *any* one of its connected logical units. The application program designates the desired mode—specific or any—with each RECEIVE macro instruction. These two modes are called, respectively, the *specific-mode,* and the *any-mode.*

In general, an application program initially requests input from a logical unit in the any-mode, and then communicates with the logical unit in the specific-mode until the transaction, inquiry, or conversation is completed. While communications proceed with one logical unit, the application program keeps a RECEIVE macro instruction (issued in the any-mode) pending so that a new transaction, inquiry, or conversation can be handled while the previous ones continue.

In the any-mode, the application program does not know the identity of the logical unit until the data has been moved into its input area and the RECEIVE has been completed. Since the logical unit is initially unknown, the amount of incoming data may also be unknown. This means that the application program must either reserve an input area large enough to hold the largest possible amount of incoming data or execute additional instructions to handle overlength data. On the other hand, the any-mode allows the application program to use just one input area for data from all of its logical units, rather than using a separate input area for each of its logical units.

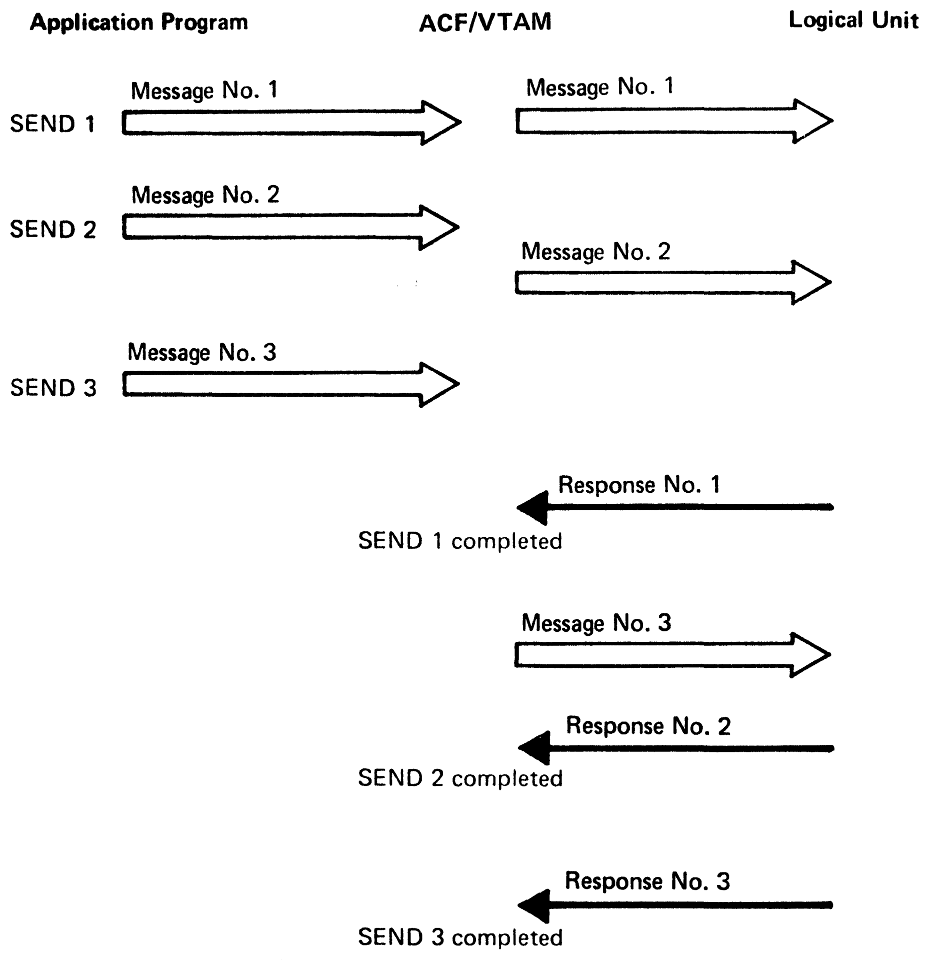


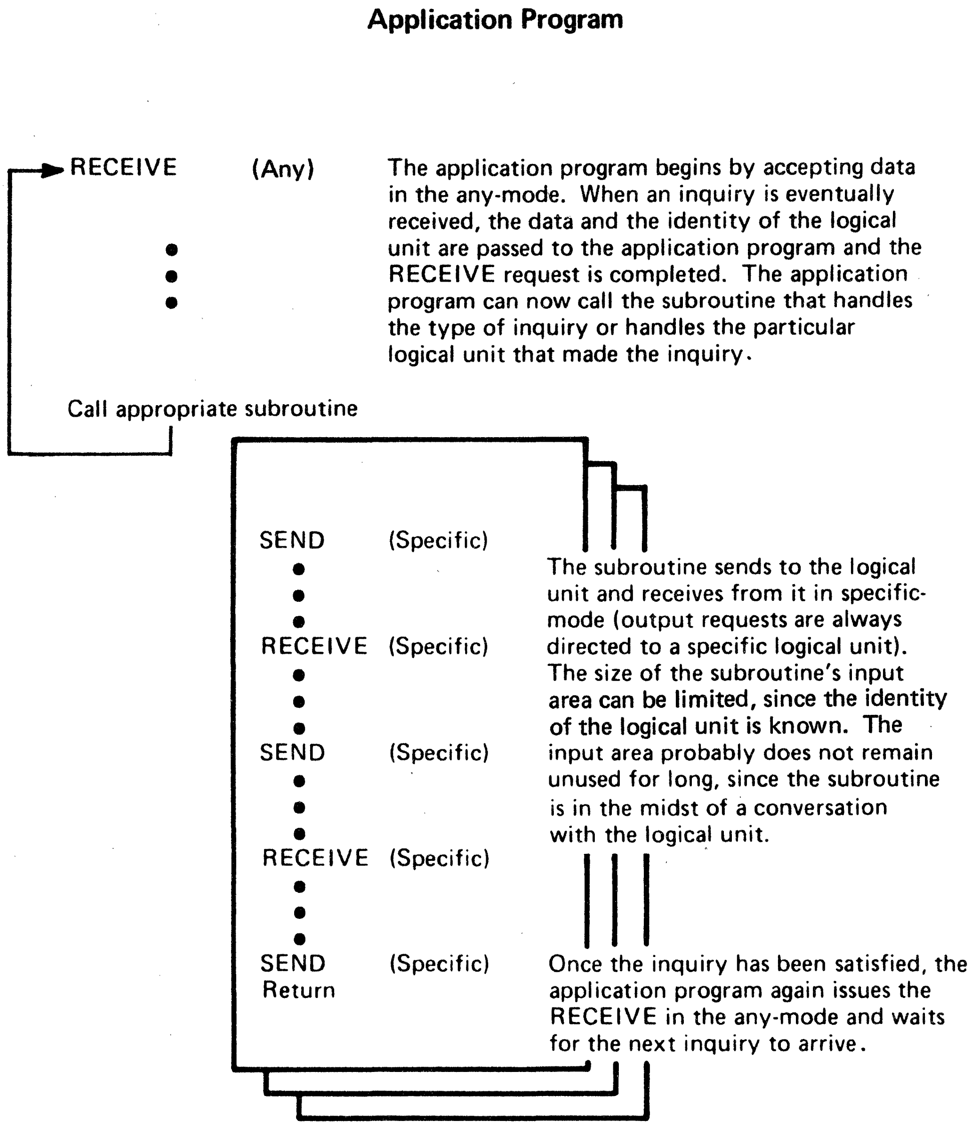
Figure 6-11. Responded Output

With the specific-mode, the application program must specify the identity of the logical unit supplying the data. Since the identity of the source is known, the size of the input data is more predictable than with the any-mode. A disadvantage is that, since any given logical unit may not supply data for some time, the application program may have to contend with unused data areas. The simplest way to avoid this problem is to not issue RECEIVE requests in the specific-mode unless data has already arrived in ACF/VTAM's buffers or is expected to arrive in a relatively short time.

Input data areas can be more efficiently managed by using a combination of specific-mode and any-mode. As an example, consider an application program that obtains an inquiry from any of its logical units, handles that inquiry with a series of SEND and RECEIVE macro instructions, and then obtains a new inquiry. Part of such a program is illustrated in Figure 6-12.

The Continue-Any versus the Continue-Specific Mode

The example in Figure 6-12 assumes that I/O requests are handled synchronously. The application program handles each inquiry serially, never accepting a new inquiry until it has completed the previous one. Although this procedure might be suitable for application programs that deal with short inquiries and a few logical units, most applications require handling inquiries in parallel.


Application Program

RECEIVE (Any)

The application program begins by accepting data in the any-mode. When an inquiry is eventually received, the data and the identity of the logical unit are passed to the application program and the RECEIVE request is completed. The application program can now call the subroutine that handles the type of inquiry or handles the particular logical unit that made the inquiry.

Call appropriate subroutine

```
SEND      (Specific)
  •
  •
  •
RECEIVE   (Specific)
  •
  •
  •
SEND      (Specific)
  •
  •
  •
RECEIVE   (Specific)
  •
  •
  •
SEND      (Specific)
Return
```

The subroutine sends to the logical unit and receives from it in specific-mode (output requests are always directed to a specific logical unit). The size of the subroutine's input area can be limited, since the identity of the logical unit is known. The input area probably does not remain unused for long, since the subroutine is in the midst of a conversation with the logical unit.

Once the inquiry has been satisfied, the application program again issues the RECEIVE in the any-mode and waits for the next inquiry to arrive.

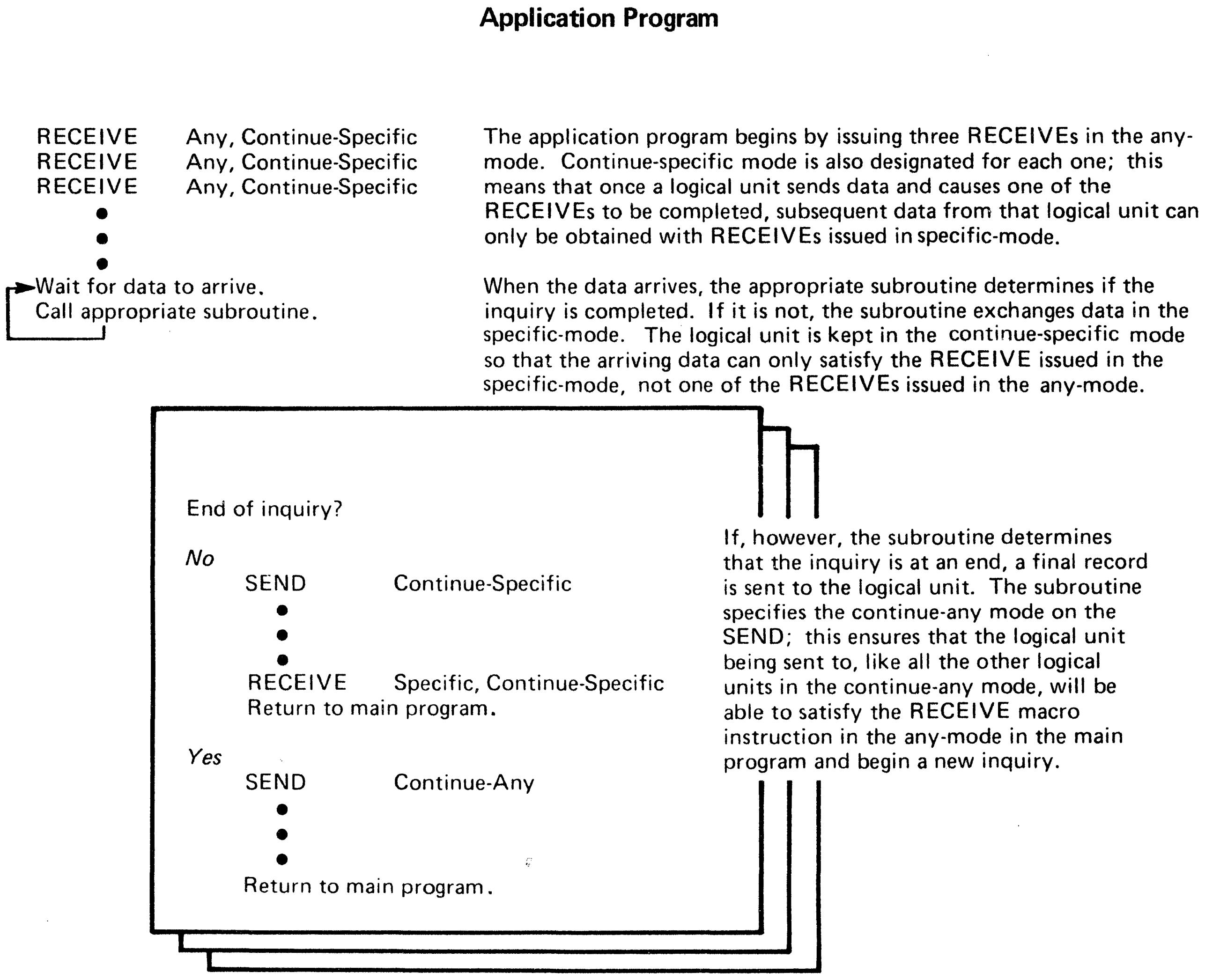
Figure 6-12. Example of Using Any-Mode and Specific-Mode to Handle an Inquiry from a Logical Unit

An application program that handles more than one inquiry concurrently (Sample Program 2 in Part 3 is an example) can use asynchronous request handling and issue new RECEIVEs in the any-mode before the previous inquiry is completed. This, however, raises the possibility that both a RECEIVE for a specific logical unit and a RECEIVE for any logical unit (which includes the specific logical unit as well) might be awaiting data at the same time. Consequently, data that is meant to satisfy the subroutine's RECEIVE might instead be intercepted by the RECEIVE in the main program, which is meant only to receive new inquiries.

To eliminate this sort of problem, ACF/VTAM allows the application program to indicate when a particular logical unit's data can be received by a RECEIVE macro instruction issued in the any-mode, and when the data must be received by a RECEIVE macro instruction issued in the specific-mode. The former is called *continue-any* mode, and the latter is called the *continue-specific* mode. These modes are designated when an I/O request is issued, but do not become effective until the I/O operation is completed. The RESETSR macro instruction can also be used to reset the mode for a logical unit. Figure 6-13 illustrates how the various modes described above relate to one another.

An Explicit RECEIVE versus DFASY and RESP Exit Routines for Responses and Expedited-Flow Commands

An ACF/VTAM application program that may recieve expedited-flow commands (for example, a Quiesce at End of Chain command) or that may receive responses has a choice of ways in which this kind of input can be received. A RECEIVE can be used in which RTYPE=DFASY (for commands) or RTYPE=RESP (for responses) is specified, or both can be specified in the same RECEIVE. In addition, normal input can complete the same RECEIVE (for example, RTYPE=(DFSYN,DFASY,RESP can be specified). The program then examines the RTYPE field of the RPL to determine which kind of input was received and branches to an appropriate routine. Alternatively, RECEIVEs can be used only for normal-flow messages, and the addresses of the special input routines can be designated (the DFASY and RESP exit routines) in an EXLST macro instruction to handle responses and expedited-flow commands using exit routines, however, requires execution of more system instructions to schedule the exit routines and is therefore less efficient than checking the RTYPE field.



Application Program

```
RECEIVE    Any, Continue-Specific
RECEIVE    Any, Continue-Specific
RECEIVE    Any, Continue-Specific
   •
   •
   •
Wait for data to arrive.
Call appropriate subroutine.
```

The application program begins by issuing three RECEIVEs in the any-mode. Continue-specific mode is also designated for each one; this means that once a logical unit sends data and causes one of the RECEIVEs to be completed, subsequent data from that logical unit can only be obtained with RECEIVEs issued in specific-mode.

When the data arrives, the appropriate subroutine determines if the inquiry is completed. If it is not, the subroutine exchanges data in the specific-mode. The logical unit is kept in the continue-specific mode so that the arriving data can only satisfy the RECEIVE issued in the specific-mode, not one of the RECEIVEs issued in the any-mode.

```
End of inquiry?
No
     SEND        Continue-Specific
      •
      •
      •
     RECEIVE     Specific, Continue-Specific
     Return to main program.
Yes
     SEND        Continue-Any
      •
      •
      •
     Return to main program.
```

If, however, the subroutine determines that the inquiry is at an end, a final record is sent to the logical unit. The subroutine specifies the continue-any mode on the SEND; this ensures that the logical unit being sent to, like all the other logical units in the continue-any mode, will be able to satisfy the RECEIVE macro instruction in the any-mode in the main program and begin a new inquiry.

Figure 6-13. An Example of Using Continue-Any and Continue-Specific Modes to Handle Concurrent Inquiries

Handling Overlength Input Data

When an application program issues a RECEIVE macro instruction, the length of the incoming data is often unpredictable. As noted earlier, this is particularly true of RECEIVE macro instructions issued in the any-mode. ACF/VTAM provides two ways of handling data that is too large for the input area:

- ACF/VTAM can discard the overlength data. The excess data is lost. This facility, called the TRUNC (truncate) option, is useful in applications that must impose rigid size limitations on input data. For example, an inventory-control application might require the logical unit to supply an account number no longer than 10 bytes.
- ACF/VTAM can keep the data. ACF/VTAM fills the input area, saves the remainder, and completes the input request. Additional input requests must be issued to obtain the excess data. This facility is called the KEEP option.

When the data message read by ACF/VTAM is larger than the number of bytes specified in the AREALEN operand of a RECEIVE macro instruction, the RECLEN field of the RPL indicates, after completion of the RECEIVE, the number of bytes that were available before the RECEIVE was executed. This characteristic of the RECLEN field is shown in Figure 6-14.

The application program can select the appropriate option when the logical unit is connected (PROC=TRUNC|KEEP specified in the NIB). Or it can select it when the RECEIVE is issued (OPTCD=TRUNC|KEEP specified in the RPL).

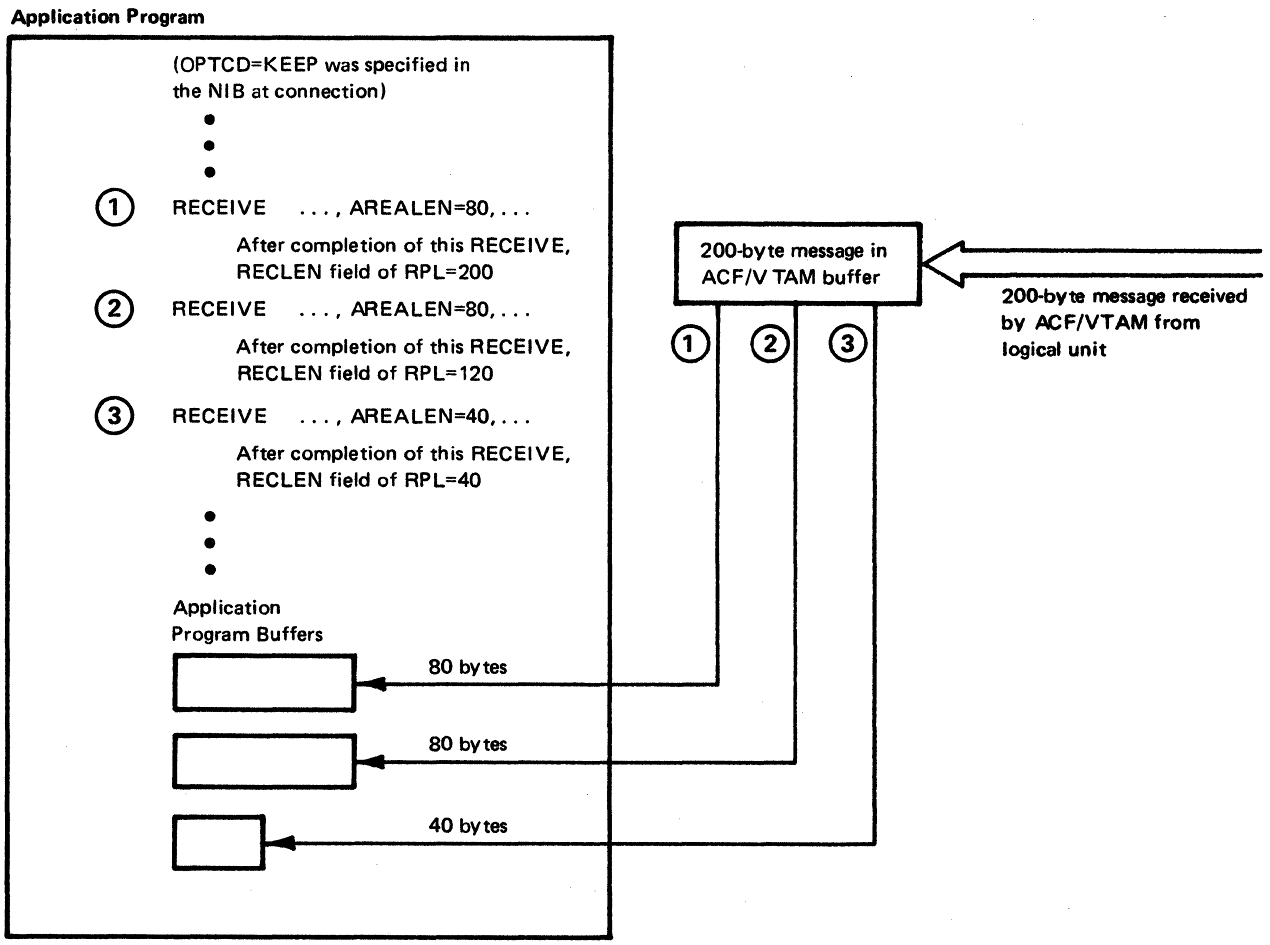

Application Program
(OPTCD=KEEP was specified in the NIB at connection)
① RECEIVE ..., AREALEN=80,...
After completion of this RECEIVE, RECLEN field of RPL=200
② RECEIVE ..., AREALEN=80,...
After completion of this RECEIVE, RECLEN field of RPL=120
③ RECEIVE ..., AREALEN=40,...
After completion of this RECEIVE, RECLEN field of RPL=40
Application Program Buffers
80 bytes
80 bytes
40 bytes
200-byte message in ACF/VTAM buffer
① ② ③
200-byte message received by ACF/VTAM from logical unit

Figure 6-14. An Example Showing Values in the RECLEN Field of an RPL

Using SNA Protocols

The major alternatives described above are of interest to all ACF/VTAM application program designers. Here are some additional facilities that not every user will require, but which should be considered:

- The chaining of messages so that the number of responses required is minimized (CHAIN=ONLY|FIRST|MIDDLE|LAST)
- The quiescing of messages so that a sender can be told to temporarily stop sending when, for example, an input buffer is about to overflow (CONTROL=QEC)
- A method of communication that ensures that only the ACF/VTAM application program or the logical unit can be sending at one time, using either:
  - Quiesce protocol
  - Change-direction protocol
- A method of communication that ensures that unexpected output from an ACF/VTAM application program will be postponed until completion of an existing transaction (bracket protocol)

Figures C-6, C-7, and C-8 in Appendix C show examples of quiesce, bracket, and change-direction protocols.

Chaining

Application programs (or logical units) can group any number of messages into a set called a *chain*. The sender can indicate which part of a chain is being transmitted—the first message of the chain, the last message of the chain, neither (the message is somewhere in the middle), or both (the message is the sole element of the chain).

Systems Network Architecture (SNA) allows only three types of chains to be used:

No-response chain, in which each element in the chain requests no response.

Exception-response chain, in which each element in the chain requests a negative response only.

Definite-response chain, in which the last element in the chain requests a definite response and all other elements request a negative response only.

With these types of chains, no more than one response per chain is sent from the receiving logical unit.

The sender of a chain can at any time send a *Cancel* command to the receiver (the sender might send this command because the receiver has returned a negative response). The Cancel command informs the receiver that the current chain is abnormally terminated, that the receiver will receive no further elements in the chain, and that the receiver may want to discard the chain elements it has already received.

The actual unit of work that the chain represents is determined entirely by the application program and the logical unit. When connection is established, the application program and logical unit determine what chaining protocols are to be used.

Figure 6-15 illustrates a possible use of chaining. In this example, a logical unit submits an inquiry to the application program. The application program can obtain various pieces of information from data files and send them to the logical unit as each becomes available. By chaining the output requests, the application program has a convenient way of telling the logical unit whether any given piece of data represents the beginning, middle, or end of a reply to an inquiry.

Figure 6-16 shows the use of chaining illustrated by Figure 6-15 in more detail. Chaining is also shown in Figure C-5 in Appendix C.

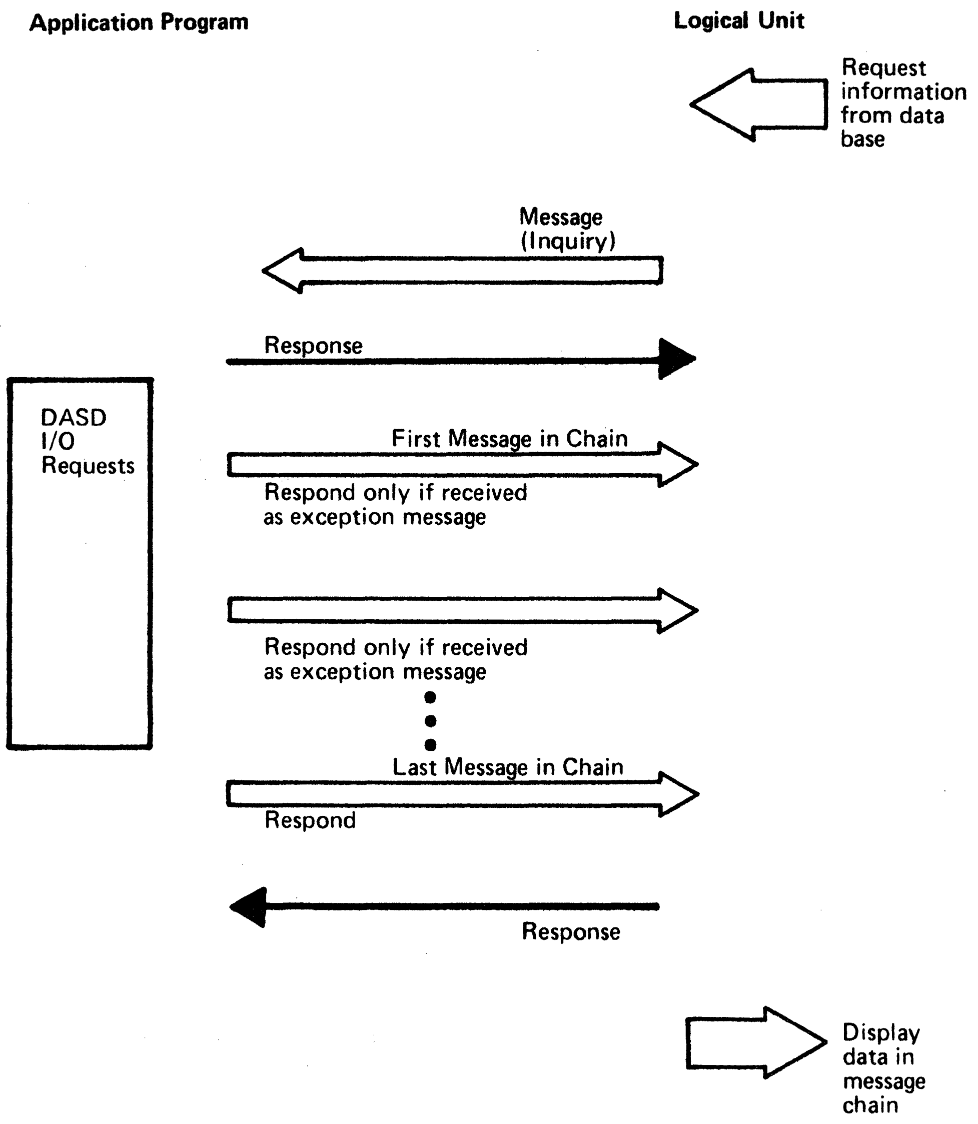

Application Program
Logical Unit
Request information from data base
Message (Inquiry)
Response
DASD I/O Requests
First Message in Chain
Respond only if received as exception message
Respond only if received as exception message
Last Message in Chain
Respond
Response
Display data in message chain

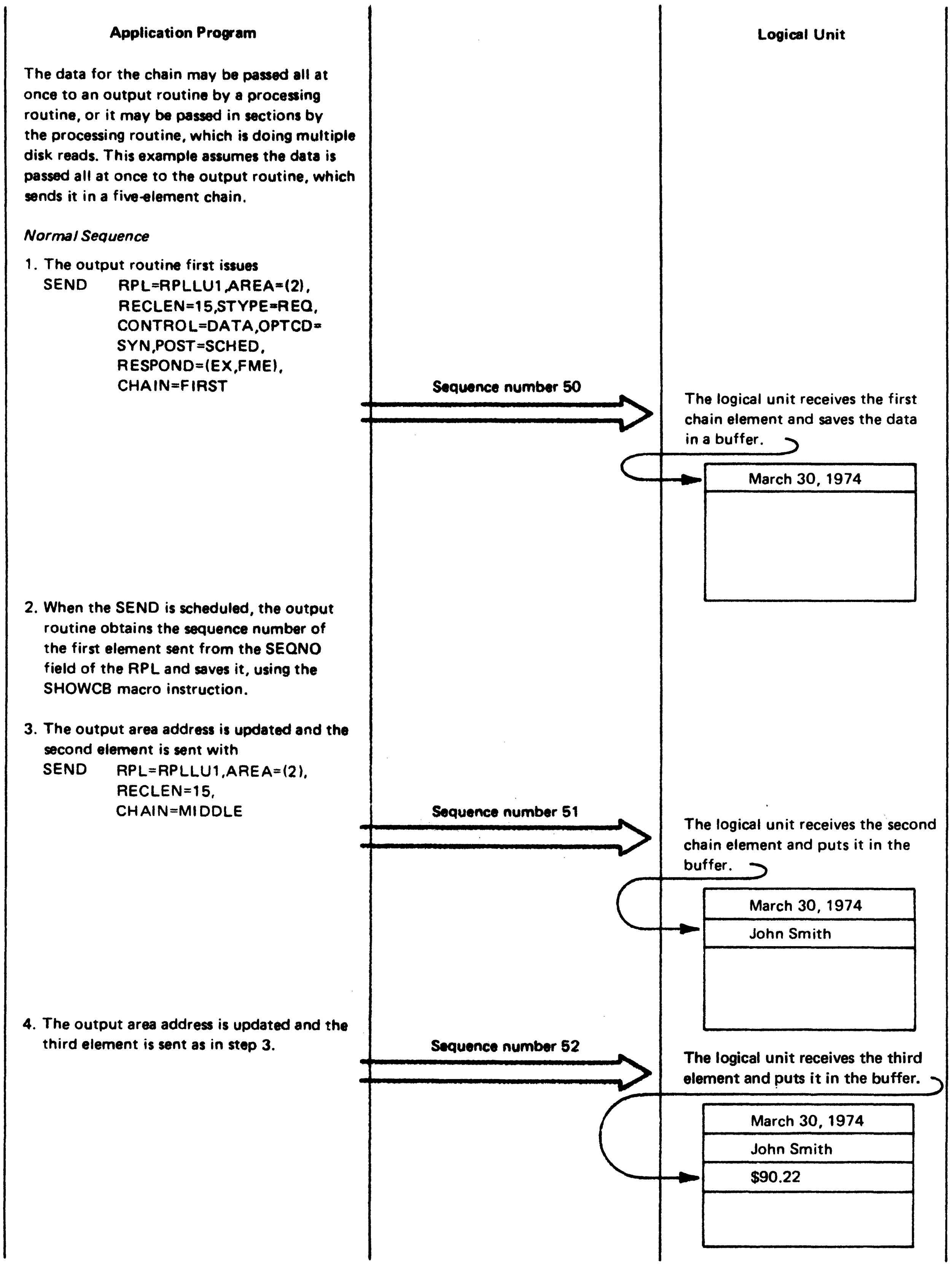
Figure 6-15. An Example of Message Chaining

Request and Response Modes

When session parameters are sent to a logical unit as part of the connection process, certain combinations of protocol bits establish certain request and response modes. Essentially, the bits indicate whether chaining will be permitted, how often a response will be requested, and in what order the responses must be returned.

There are two modes in which the senders of messages can operate: immediate control mode or delayêd control mode.

Immediate Control Mode: When operating in this mode, the sender sends only single-element messages (that is, cannot send a chain), and the sender requests a definite response to each single-element message. After sending each message, the sender must wait for a response before sending the next message.



Application Program

Logical Unit

The data for the chain may be passed all at once to an output routine by a processing routine, or it may be passed in sections by the processing routine, which is doing multiple disk reads. This example assumes the data is passed all at once to the output routine, which sends it in a five-element chain.

Normal Sequence

1. The output routine first issues

```
SEND      RPL=RPLLU1,AREA=(2),
          RECLEN=15,STYPE=REQ,
          CONTROL=DATA,OPTCD=
          SYN,POST=SCHED,
          RESPOND=(EX,FME),
          CHAIN=FIRST
```

The logical unit receives the first chain element and saves the data in a buffer.

2. When the SEND is scheduled, the output routine obtains the sequence number of the first element sent from the SEQNO field of the RPL and saves it, using the SHOWCB macro instruction.

3. The output area address is updated and the second element is sent with

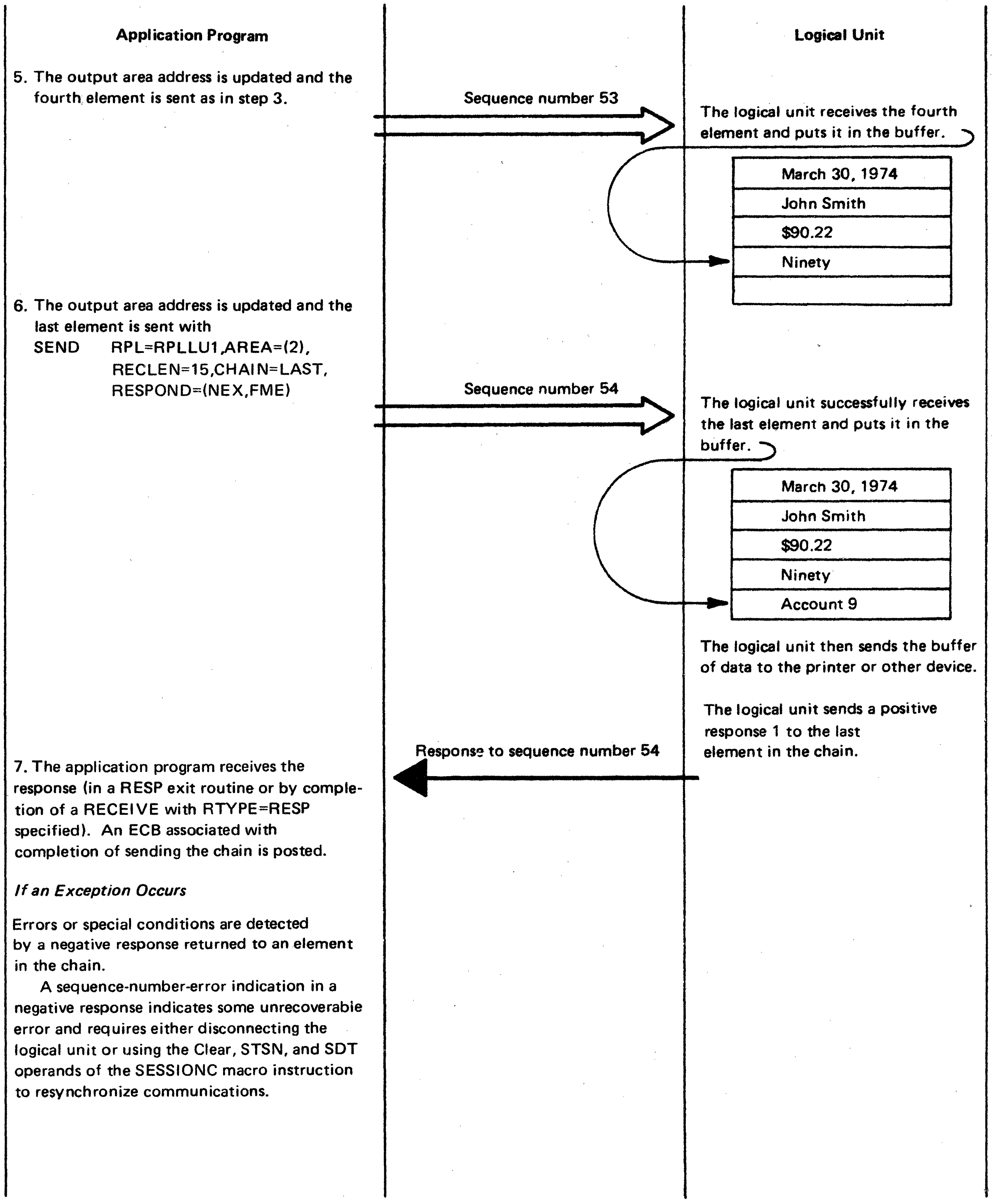
```
SEND      RPL=RPLLU1,AREA=(2),
          RECLEN=15,
          CHAIN=MIDDLE
```

The logical unit receives the second chain element and puts it in the buffer.

4. The output area address is updated and the third element is sent as in step 3.

The logical unit receives the third element and puts it in the buffer.

Figure 6-16 (Part 1 of 2). An Example of Sending a Chain of Messages to a Logical Unit That Is Buffering Data



5. The output area address is updated and the fourth element is sent as in step 3.

The logical unit receives the fourth element and puts it in the buffer.

6. The output area address is updated and the last element is sent with

```
SEND     RPL=RPLLU1,AREA=(2),
         RECLEN=15,CHAIN=LAST,
         RESPOND=(NEX,FME)
```

The logical unit successfully receives the last element and puts it in the buffer.

The logical unit then sends the buffer of data to the printer or other device.

The logical unit sends a positive response 1 to the last element in the chain.

7. The application program receives the response (in a RESP exit routine or by completion of a RECEIVE with RTYPE=RESP specified). An ECB associated with completion of sending the chain is posted.

If an Exception Occurs

Errors or special conditions are detected by a negative response returned to an element in the chain.

A sequence-number-error indication in a negative response indicates some unrecoverable error and requires either disconnecting the logical unit or using the Clear, STSN, and SDT operands of the SESSIONC macro instruction to resynchronize communications.

Figure 6-16 (Part 2 of 2). An Example of Sending a Chain of Messages to a Logical Unit That Is Buffering Data

Delayed Control Mode: When operating in delayed control mode, the sender may send single-element messages and may also send multiple-element messages (chains). There are two forms of delayed control mode:

Immediate request mode: The distinguishing characteristic of this mode is that the sender may send a series of elements constituting one or more complete chains and ask for a definite response only in the last element in the series. In addition, once the sender has requested a definite response, it will send no other element until it receives the definite response. Thus, if the sender is sending a series of single-element messages, only the last single-element message will request a definite response; the other single-element messages will request an exception response only. If the sender is sending a chain, only the last element in the chain will request a definite response. If the sender is sending multiple chains, only the last element in the last chain requests a definite response; all preceding elements ask for an exception response only.

Delayed request mode: The distinguishing characteristic of this mode is that, while the sender may insert requests for definite responses into the series of elements it is sending, it is not required to wait for any of those responses. This mode can be used to send multiple chains, with a definite response requested in the last element of each chain (all other elements in each chain would request an exception response only), and the sender can send any number of chains before stopping to wait for responses.

The receiver may be in either of two modes:

Immediate response mode: The receiver sends responses in the same order as the sender requested them. Thus, when the sender receives a response, it can infer that the receiver has received all preceding elements and that no negative responses will be forthcoming for those preceding elements.

Delayed response mode: The receiver need not return responses in the same order as they were requested. A response for one element may be delayed beyond the response for a subsequent element. There is one restriction, however, on the receiver. The receiver must send responses for elements preceding a Chase command before it sends the response to the Chase command.

Quiescing

ACF/VTAM provides a set of commands that the application program can use to request a logical unit to stop sending normal-flow messages (data messages and data flow control commands) to the program. A logical unit can also request that the ACF/VTAM application program stop sending.

One use of this facility is to ensure that, at a given time, only one side (the ACF/VTAM application program or the logical unit) can send normal-flow messages. (This use of the quiesce commands is described below as "quiesce protocol.")

Another use of quiescing is to stop the other end of the session from sending because of a temporary condition or problem. This action is usually needed when the sender is sending a long chain or a series of chains and the receiver wants the transmissions to be stopped temporarily. Often, the receiver needs to halt the transmissions because the receiver is running out of buffer space in which to store the incoming data. Another reason is to stop the incoming messages long enough to allow the receiver to send an informational message of its own.

To understand how quiescing works, consider the situation in which the receiver is running out of buffer space. Assume that this condition develops at the logical unit while the application program is in the middle of sending a chain to the logical unit. To tell the application program that it should stop sending data, the logical unit sends a *Quiesce at End of Chain* (QEC) command to the application program. The exact meaning of that command must have been worked out between the logical unit and the application program before the programs were coded. Receipt of the command might mean "stop sending immediately and do not complete the chain," or it might mean "stop sending after you complete the current chain." If it means "stop sending immediately," the application program can send a Cancel command or a special message to tell the logical unit to discard the beginning of the chain. If the QEC command means "complete the chain before stopping," the application program continues sending elements until the chain is completed. In either case, the application program signals its compliance with the QEC request by sending a *Quiesce Complete* (QC) indicator to the logical unit. The logical unit then continues disposing of previously received elements (perhaps by printing them or by writing them to disk storage).

When buffers are available to hold more incoming data, the logical unit sends a *Release Quiesce* (RELQ) command to the application program. Upon receipt of that command, the application program recommences sending (either at the beginning of the aborted chain or at the beginning of a new chain, depending upon the agreed-upon protocol). Figure 6-17 illustrates the use of quiescing to prevent buffer overflow.

After the application program stops sending elements and before it sends the Quiesce Complete indicator, the program can send certain normal-flow commands. For example, at that point, the application program can send a Chase command to ensure that it has received all responses before it sends the Quiesce Complete indicator.

Protocols for Ensuring Orderly Communications

Certain types of devices are limited in their communication with each other to specific directions of traffic flow. Some devices can only send messages; others can only receive (*master/slave* or *simplex*). Some devices can both send and receive, but can only do one of them at a time (*half duplex*). Others can send and receive simultaneously (*full duplex*). These characteristics are one factor that affect the selection of session parameters, which are sent by the application program to the logical unit when connection is established (see "Establishing Session Parameters during Connection" in Chapter 5). Other factors that affect the selection of session parameters are (1) the type of communication that will take place (interactive versus batch, for example) and (2) particular conventions that are agreed upon between programmers before the host application program and the logical unit program are written.

Systems Network Architecture (SNA) provides several protocols that enable the application program and the logical unit to coordinate and control the direction of flow and their exchanges of messages. None of these protocols is enforced by ACF/VTAM; ACF/VTAM sends the commands and indicators specified by the sender without checking them and without comparing them to the current status of communications. It is the responsibility of the application program and the logical unit to abide by the communication rules (the session parameters) they agreed upon when the connection was established.

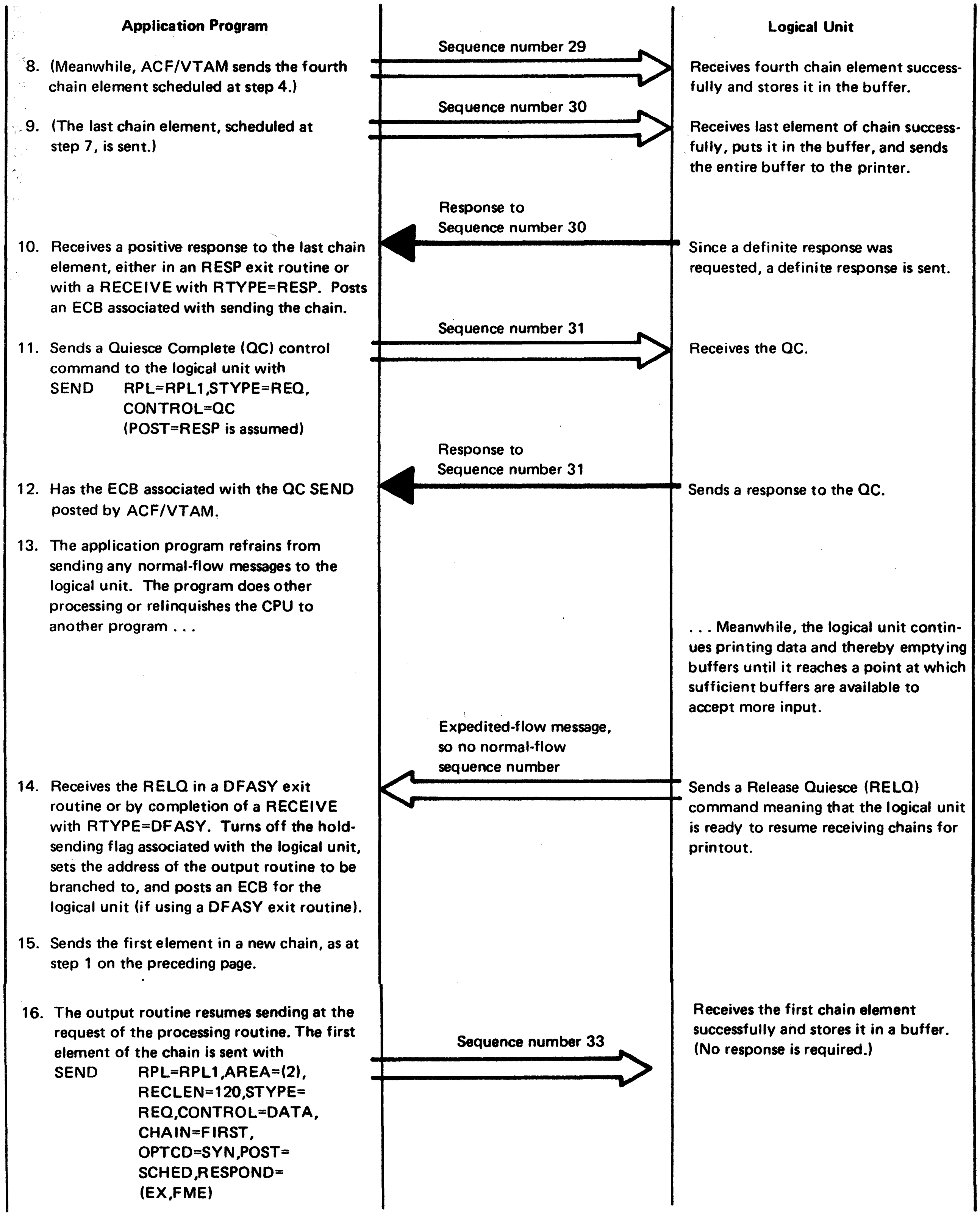
Quiesce Protocol: As described above, the quiesce commands can be used to temporarily stop the sender from sending when the receiver encounters a problem or special condition. Another use of the quiesce commands is to ensure that, at any one time, only one end of the session (the application program or the logical unit) can send normal-flow messages. This second use of the quiesce commands is called *quiesce protocol.*

In this protocol, one end of the session controls the direction of flow by using the quiesce commands to "turn off" normal-flow transmission by the other end of the session. For example, assume that the application program is to control the direction of flow. Whenever the application program has not quiesced the logical unit, the logical unit is free to send normal-flow messages. When the application program wants to start sending, it informs the logical unit by transmitting the Quiesce at End of Chain command on the

Application Program		**Logical Unit**
The ACF/VTAM application program is sending continuous chains [1] of data to the logical unit for a printout. Each chain contains five elements. Each element is sent with a SEND macro instruction. A processing routine passes the data for each chain to the output routine. This example begins with a new chain being sent by the output routine.		
1. Sends the first chain element with SEND RPL=RPL1,AREA=(2), RECLEN=120,STYPE= REQ,CONTROL=DATA, CHAIN=FIRST,OPTCD= SYN,POST=SCHED, RESPOND=(EX,FME)	Sequence number 26 ⇒	Receives first chain element successfully and stores it in a buffer. (No response is required.)
2. Updates the data area address in register 2 and sends the second chain element with SEND RPL=RPL1,AREA=(2), CHAIN=MIDDLE	Sequence number 27 ⇒	Receives second chain element successfully and stores it in the buffer.
3. Updates the data area address in register 2 and sends the third chain element with SEND RPL=RPL1,AREA=(2), CHAIN=MIDDLE	Sequence number 28 ⇒	Receives third chain element successfully and stores it in the buffer.
4. Updates the data area address in register 2 and sends the fourth chain element with SEND RPL=RPL1,AREA=(2), CHAIN=MIDDLE	ACF/VTAM schedules the output but has not yet sent it when . . .	. . . The logical unit recognizes that it is running low on buffer space because it is receiving data faster than it can print it. The logical unit must tell the application program to stop sending. This will give the logical unit time to clear out some of its buffers. It sends a Quiesce at End of Chain (QEC) command.
5. ACF/VTAM schedules the program's DFASY exit routine or completes a RECEIVE that specifies RTYPE=DFASY. The RPL contains QEC in the CONTROL field. (If requested by the application program, ACF/VTAM will have sent a response to the QEC command.)	Expedited-flow message; no normal-flow sequence number ⇐	
6. The program sets a program-defined flag indicating that, for this logical unit, the next output request after sending the present chain is to be held in abeyance until the quiesce is released.		

7. The program updates the data area address and schedules the sending of the fifth and last element in the chain with
SEND RPL=RPL1,AREA=(2),
CHAIN=LAST,
RESPOND=(NEX,FME)		

Figure 6-17 (Part 1 of 2). An Example of a Logical Unit Quiescing an Application Program in Order to Interrupt Continuous Sending



Application Program		Logical Unit
8. (Meanwhile, ACF/VTAM sends the fourth chain element scheduled at step 4.)	Sequence number 29 ⇒	Receives fourth chain element successfully and stores it in the buffer.
9. (The last chain element, scheduled at step 7, is sent.)	Sequence number 30 ⇒	Receives last element of chain successfully, puts it in the buffer, and sends the entire buffer to the printer.
10. Receives a positive response to the last chain element, either in an RESP exit routine or with a RECEIVE with RTYPE=RESP. Posts an ECB associated with sending the chain.	◀ Response to Sequence number 30	Since a definite response was requested, a definite response is sent.
11. Sends a Quiesce Complete (QC) control command to the logical unit with SEND RPL=RPL1,STYPE=REQ, CONTROL=QC (POST=RESP is assumed)	Sequence number 31 ⇒	Receives the QC.
12. Has the ECB associated with the QC SEND posted by ACF/VTAM.	◀ Response to Sequence number 31	Sends a response to the QC.
13. The application program refrains from sending any normal-flow messages to the logical unit. The program does other processing or relinquishes the CPU to another program . . .		. . . Meanwhile, the logical unit continues printing data and thereby emptying buffers until it reaches a point at which sufficient buffers are available to accept more input.
14. Receives the RELQ in a DFASY exit routine or by completion of a RECEIVE with RTYPE=DFASY. Turns off the hold-sending flag associated with the logical unit, sets the address of the output routine to be branched to, and posts an ECB for the logical unit (if using a DFASY exit routine).	⇐ Expedited-flow message, so no normal-flow sequence number	Sends a Release Quiesce (RELQ) command meaning that the logical unit is ready to resume receiving chains for printout.
15. Sends the first element in a new chain, as at step 1 on the preceding page.		
16. The output routine resumes sending at the request of the processing routine. The first element of the chain is sent with SEND RPL=RPL1,AREA=(2), RECLEN=120,STYPE= REQ,CONTROL=DATA, CHAIN=FIRST, OPTCD=SYN,POST= SCHED,RESPOND= (EX,FME)	Sequence number 33 ⇒	Receives the first chain element successfully and stores it in a buffer. (No response is required.)

[1] Chaining is shown in this example. However, quiescing can also be performed when continuous sending does not involve chaining (each SEND specifies CHAIN=ONLY).

Figure 6-17 (Part 2 of 2). An Example of a Logical Unit Quiescing an Application Program in Order to Interrupt Continuous Sending

expedited flow. On receipt of that command, the logical unit knows that it must stop sending normal-flow messages when it completes sending the current chain. The logical unit also knows that the next normal-flow transmission will come from the application program. The application program then starts sending normal-flow messages and continues until it sends the Release Quiesce command to the logical unit. On receipt of that command, the logical unit knows that it can again start sending normal-flow messages. In this way, the application program alternately grants the logical unit permission to send (by transmitting the Release Quiesce command) and stops the logical unit from sending (by transmitting the Quiesce at End of Chain command). The direction of flow can similarly be controlled by the logical unit.

Quiesce state applies only to normal-flow traffic. While a program or logical unit is in quiesce state, it can send responses and expedited-flow commands. Figure 6-18 shows an example of quiesce protocol.

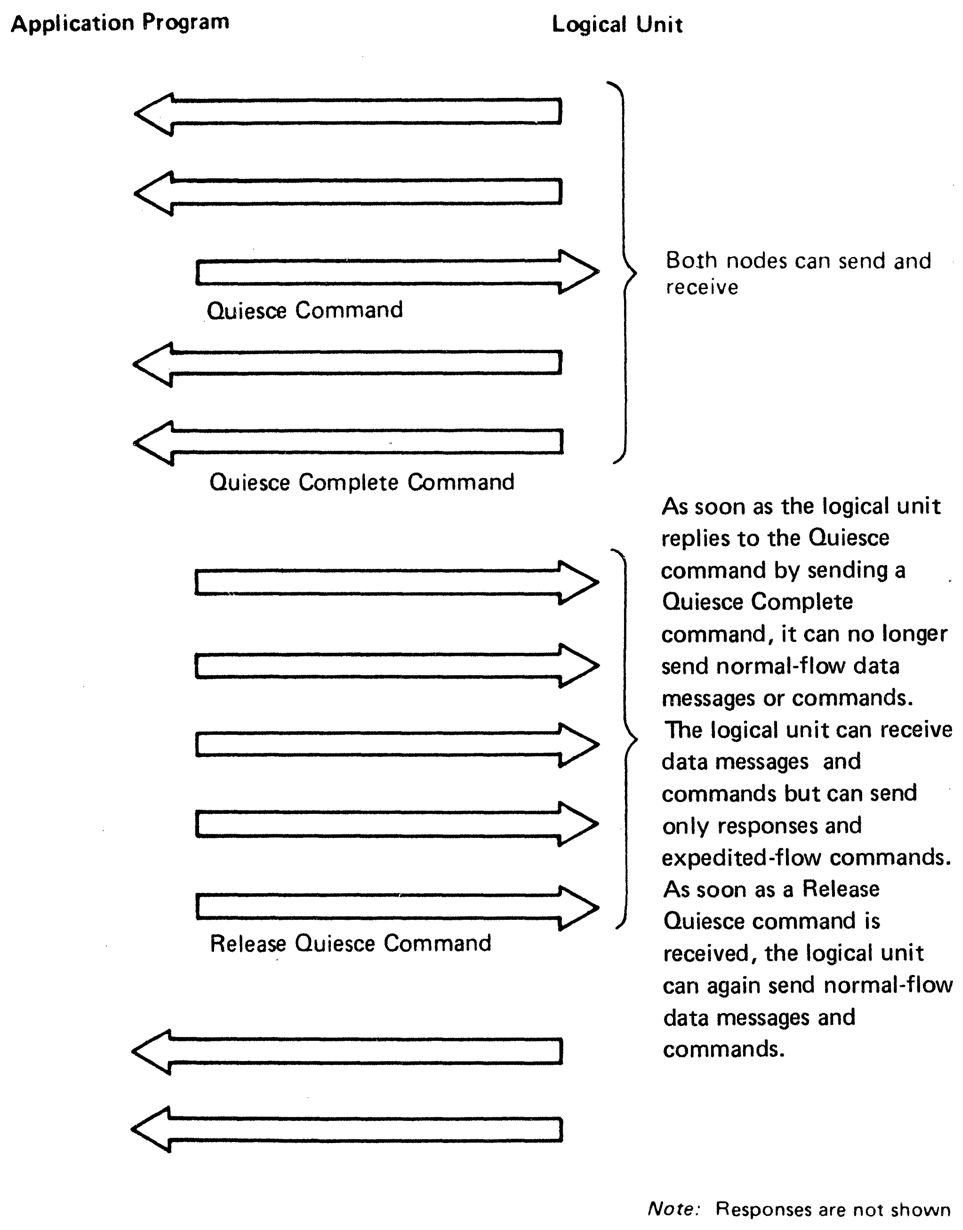

Application Program
Logical Unit
Both nodes can send and receive
Quiesce Command
Quiesce Complete Command
As soon as the logical unit replies to the Quiesce command by sending a Quiesce Complete command, it can no longer send normal-flow data messages or commands. The logical unit can receive data messages and commands but can send only responses and expedited-flow commands. As soon as a Release Quiesce command is received, the logical unit can again send normal-flow data messages and commands.
Release Quiesce Command
Note: Responses are not shown

Figure 6-18. Quiesce Protocol

Change-Direction Protocol: In this protocol, the application program and the logical unit alternately relinquish the ability to send normal-flow messages by transmitting the *Change Direction Command indicator* to the other end of the session. This protocol is used in the half-duplex mode of communication, which is the mode in which a unit can either send or receive but cannot do both at the same time.

There are two forms of half-duplex communication: *half-duplex flip-flop communication* and *half-duplex contention communication.* In half-duplex flip-flop communication, one end of the session (the application program or the logical unit) is designated in the session parameters as the first to send a message after a session is established; thereafter, the program and the logical unit notify each other, in turn, that the other side can begin sending normal-flow messages. In half-duplex contention communication, after connection has been established, the application program and the logical unit can both attempt to start sending a normal-flow message at the same time (called *contention*). The one that is allowed to proceed is the one that was designated in the session parameters as the one that would always win in a contention situation. Similarly, in contention communication, when either end of the session finishes sending a chain of normal-flow messages, both ends can attempt at the same time to start sending a new message; again, the winner of the contention is the one designated as such in the session parameters.

One bit in the common protocol portion of the session parameters controls priority for initial sending in half-duplex communication. One setting of the bit indicates that the logical unit has priority for sending; that is, (1) in flip-flop communication, the logical unit is to send the first normal-flow message in the session, or (2) in contention communication, the logical unit is to win the contention. The other setting of the bit indicates that the application program is to have priority for sending.

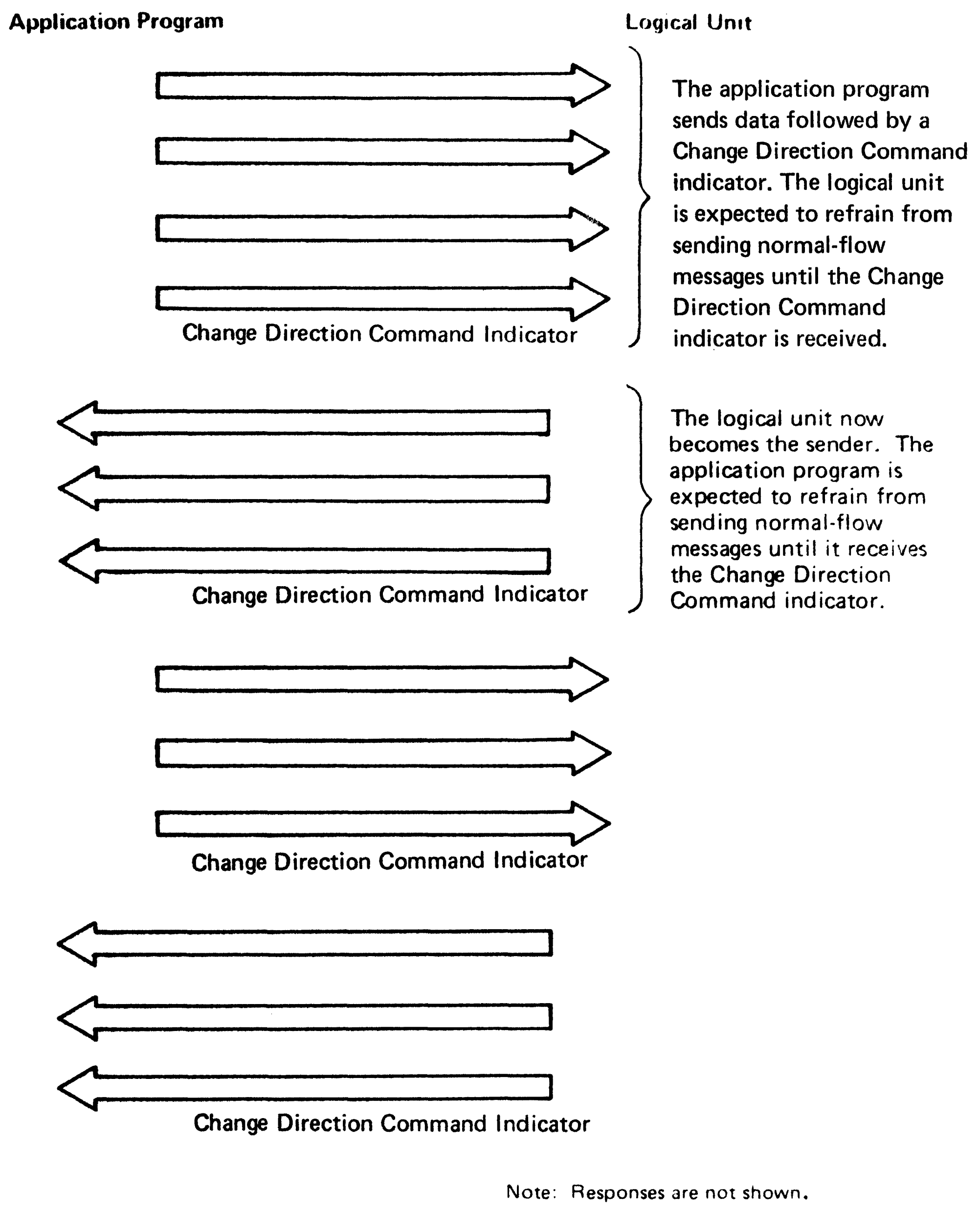
Change-direction protocol must be used in half-duplex flip-flop communication; the protocol may optionally be used in half-duplex contention communication.

Change-direction protocol works like this: The side that is the first to send continues sending normal-flow messages until it reaches the end of the data it wants to send. In the last element of the last chain, the sender includes a *Change Direction Command* indicator. The other side then sends normal-flow messages until it relinquishes its ability to send by including the Change Direction Command indicator in the last element of a chain. Communication continues to alternate in this fashion indefinitely, as shown in Figure 6-19.

While the receiver is awaiting the Change Direction Command indicator, it can transmit (as part of a response) a prompting indicator to the other side that in effect says, "I would like the Change Direction Command indicator sent to me now." This prompting indicator, called a *Change Direction Request* indicator, can be honored or it can be ignored. This indicator is supported by ACF/VTAM and by certain logical units (for example, logical units in the IBM 3600 Finance Communication System), but the indicator is not recognized by Systems Network Architecture (SNA). SNA defines the Signal command for requesting a Change Direction Command indicator.

The side that is awaiting a Change Direction Command indicator (like the side that has been quiesced in quiesce protocol) is prohibited only from sending normal-flow traffic. It is free to send responses and expedited-flow commands.

As mentioned previously, ACF/VTAM does not enforce the change-direction protocol. Should the side waiting for a Change Direction Command indicator begin sending data anyway, ACF/VTAM does not prevent the transmission. Compliance with the change-direction protocol is entirely the responsibility of the application program and the logical unit.



Application Program
Logical Unit
The application program sends data followed by a Change Direction Command indicator. The logical unit is expected to refrain from sending normal-flow messages until the Change Direction Command indicator is received.
Change Direction Command Indicator
The logical unit now becomes the sender. The application program is expected to refrain from sending normal-flow messages until it receives the Change Direction Command indicator.
Change Direction Command Indicator
Change Direction Command Indicator
Change Direction Command Indicator
Note: Responses are not shown.

Figure 6-19. Change-Direction Protocol

Bracket Protocol: A *bracket* is any unit of work that an application program and a logical unit have been programmed to accomplish. A bracket may consist of any combination of data messages and data replies, ranging from a single message in one direction to an elaborate exchange of messages and replies. But, no matter how simple or complex the series of messages and replies may be, the characteristic that makes them all part of the same bracket is that they all pertain to the same unit of work.

A data-base inquiry transaction is a typical example of a bracket. In such a transaction, the logical unit sends an inquiry to the host computer asking for some piece or body of information stored in the data base. For example, an insurance agent at a terminal asks the computer to provide information on all insurance policies issued to a particular client. In answer to the inquiry, the application program in the host computer sends a single message or a series of messages containing the requested information. At this point, the bracket might end. Or, as the result of one of the replies, the logical unit might ask for further details, and the bracket does not end until the application program has acquired the details from the data base and sent them to the logical unit.

Bracket protocol is used when one or both of the ends of the session cannot begin processing a new unit of work until the current one has been completed. For example, it can be used if the logical unit or application program cannot start handling a new inquiry until the replies to the current inquiry have been completed. Bracket protocol provides a way of ensuring that a new unit of work is not started until the preceding one has been finished.

The application program and logical unit that are using bracket protocol indicate on each first-in-chain or only-in-chain message whether that chain is the beginning, middle, or end of the bracket. These delimiters allow the receiving node to determine whether or not a new bracket can be started. A *Begin Bracket* indicator is included in the first element of the first chain in a bracket. The *End Bracket* indicator is included in the first element of the last chain in the bracket.

When a connection is established, bits in the session parameters sent by the application program to the logical unit determine who wins bracket contention when both sides want to begin a bracket simultaneously, who can end a bracket, and whether bracket termination is conditional (the side sending the End Bracket indicator does not consider the bracket ended until it receives a positive response to the element that includes the indicator) or unconditional (termination occurs when any response is returned). Figure 6-20 shows an example of bracket protocol.

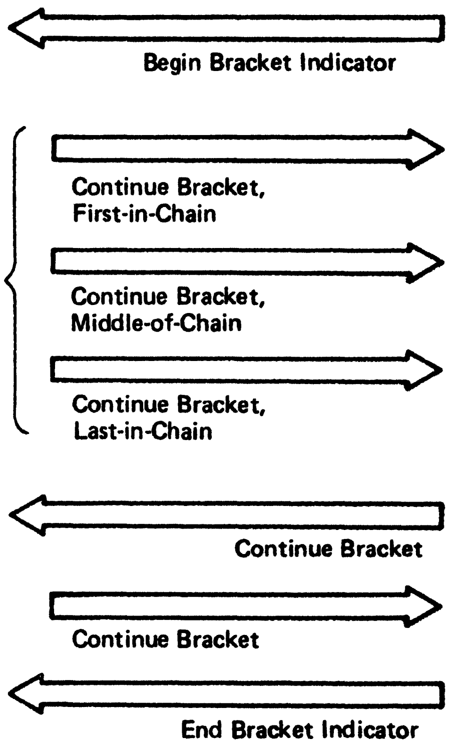
One bracket-related bit in the session parameters determines the winner of bracket contention by assigning the role of *first speaker* to one participant (application program or logical unit) and the role of bidder to the other participant. The *first speaker* is the participant that is given the ability to begin a bracket without asking permission from the other side. The *bidder* is the participant that must request and receive permission from the first speaker to begin a bracket. The bit in the session parameters designates whether the application program or the logical unit is to be the first speaker; the other participant is automatically the bidder.

When a bracket is ended, the first speaker can start a new bracket if it wants. The bidder, however, must request permission to begin a bracket. The bidder can do this in either of two ways:

- The bidder can request permission by sending a *Bid* command to the first speaker. A positive response to the Bid command indicates that the first speaker has granted permission. A negative response indicates that permission is denied. The negative response, however, may be accompanied by sense data that indicates whether the first speaker will or will not later grant the permission by sending a *Ready to Receive* command. On receipt of that command, the bidder can begin a bracket.
- The bidder can request permission by starting to send a message in which the first element contains a Begin Bracket indicator. The response indicates whether or not the bidder can continue with the bracket, with a positive response indicating that it can continue and a negative response indicating that the attempt was rejected. The negative response, however, may be accompanied by sense data that indicates whether the first speaker will or will not later send the Ready to Receive command. There are restrictions on attempting to begin a bracket by starting to send a message with a Begin Bracket indicator:
  1. If the bidder is sending only a single-element message or a single chain, the message or the first element in the chain must have the Begin Bracket and End Bracket indicators.

Application Program **Logical Unit**

The logical unit receives an inquiry from one of its input devices.


Begin Bracket Indicator
Continue Bracket, First-in-Chain
Continue Bracket, Middle-of-Chain
Continue Bracket, Last-in-Chain
Continue Bracket
Continue Bracket
End Bracket Indicator

The logical unit transmits a message to the application program with a Begin Bracket indicator.

The application program processes the inquiry. This results in transmission of a chain that ends with a query regarding the adequacy of the data.

The logical unit replies with a request for more data.

The application program transmits the additional data.

The logical unit determines that it has the data needed to satisfy the inquiry and notifies the application program that the bracket is ended.

Note: In this example, the logical unit determines the beginning and the end of the bracket. In other applications, the application program could determine the beginning and the end of the bracket, or one node could determine the beginning and the other node determine the end.

The logical unit displays the requested information.

Figure 6-20. Bracket Protocol

2. If the bidder is sending multiple chains, the first element in the first chain must contain the Begin Bracket indicator and the bidder must ask for a definite response to the first chain. If the bidder gets a negative response, it knows that its bid was rejected and that it must terminate the chain (either by sending the Cancel command or by sending an element marked last in chain).

Like quiesce and change-direction protocol, bracket protocol is not enforced by ACF/VTAM. It must be adhered to by the participants.

Special Use of RESPOND=QRESP with Bracket Protocol: Consider this situation: The application program (which is the bidder) and the logical unit (the first speaker) are in a session involving half-duplex contention and the use of brackets. They are within a bracket. They have agreed in the session parameters that a bracket is not terminated until the sender of an End Bracket indicator (EB) gets a response to the message containing EB.

Now, the application program sends a message containing EB. Simultaneously, the logical unit sends a data message that was meant to be within the current bracket. In VTAM Level 2 and if RESPOND=NQRESP is used for the EB message in ACF/VTAM, it is possible for the application program to get the response to the EB message before it gets the data message. If that happens, the application program fails to know that the logical unit meant the data message to be within the bracket.

To avoid this problem, the application program should send the EB message with a macro instruction that specifies QRESP as one of the RESPOND parameters. That parameter causes ACF/VTAM to treat the response to the message as if it were a normal-flow message. Since the logical unit must send the EB response after it sends the in-bracket message, and because the EB response will be treated as a normal-flow message, the application program will get the in-bracket message and the EB response in that order—the correct order.

Note that, because the EB response is treated as a normal-flow message, it will not cause scheduling of a RESP exit routine, nor can it cause completion of a RECEIVE specifying RTYPE=RESP. The macro instruction used to send the message containing the End Bracket indication and RESPOND=QRESP must specify POST=SCHED. The EB response itself (because it is treated as a normal-flow message) must be gotten with a RECEIVE RTYPE=DFSYN (not RTYPE=RESP).

For more information on the QRESP and NQRESP parameters in the RESPOND operand, see "Specifying Special Handling of the Response to a Normal-Flow Message or Command" under "What a Response Contains" earlier in this chapter.

Function Management Header Option

The function management (FM) header option is specified through the RPL or SEND macro. Specifying OPTCD=FMHDR, indicates that a user-defined or SNA-defined FM header is included in a data message to a logical unit. This option only applies for record mode (with STYPE=REQ,CONTROL=DATA) and indicates to ACF/VTAM how the format bit in the request header (RH) of a specific data message is to be set. If FMHDR is coded, the format bit is set in the RH and sent to the receiver of the message.

Similarly, if the format bit is on in the RH of a received message (indicating the presence of an FM header), it causes FMHDR to be set in the RPL used for the receive operation. FMHDR can be tested with the TESTCB macro instruction or by using the IFGRPL DSECT. OPTCD=FMHDR is set whenever a command or command response is received.

When connection is established, the application program and logical unit determine whether FM headers can be used.

Additional SNA Protocol Information

In addition to the protocols described earlier in this chapter, the following protocols can be specified in the session parameters when connection is made between the application program and the logical unit:

Whether or not a logical unit can remove extraneous blank characters before data is transmitted (compression)

Who has error recovery responsibility

Whether an alternate character code is acceptable (for example, ASCII instead of EBCDIC)

Communicating with the 3270 Information Display System

The application program can communicate with a BSC 3270 or local non-SNA 3270 using record-mode macro instructions. This is possible if MODE=RECORD is specified (in the

NIB) when the OPNDST macro instruction is issued to connect the 3270. ACF/VTAM provides the record-mode facility so that application programs communicating with other logical units can use the same macro instructions to communicate with 3270s.

The application program can also communicate with local non-SNA 3270s (or BSC 3270s defined with PU=NO) in the same manner used to communicate with BSC and start-stop terminals–using basic-mode and macro instructions.

For more information on communicating with 3270s, see Appendix A, "Communicating with BSC and Start-Stop Terminals," or refer to *Introduction to Programming the IBM 3270 Information Display System*, GC27-6999.

Chapter 7. Using Exit Routines

Other chapters discuss exit routines in relation to connecting or communicating with logical units or closing the program. This chapter discusses how exit routines work, summarizes them, discusses the advantages and disadvantages of using them, and describes procedures to follow in using them. These exit routines apply to application programs that use record-mode macro instructions to communicate with logical units. See Appendix A for descriptions of special exit routines that apply only to non-SNA terminals used in the basic mode.

How Exit Routines Work

ACF/VTAM provides for the use of two general kinds of exit routines: RPL-specified exit routines and EXLST exit routines. The two kinds of exit routines work somewhat differently as described below.

How RPL-Specified Exit Routines Work

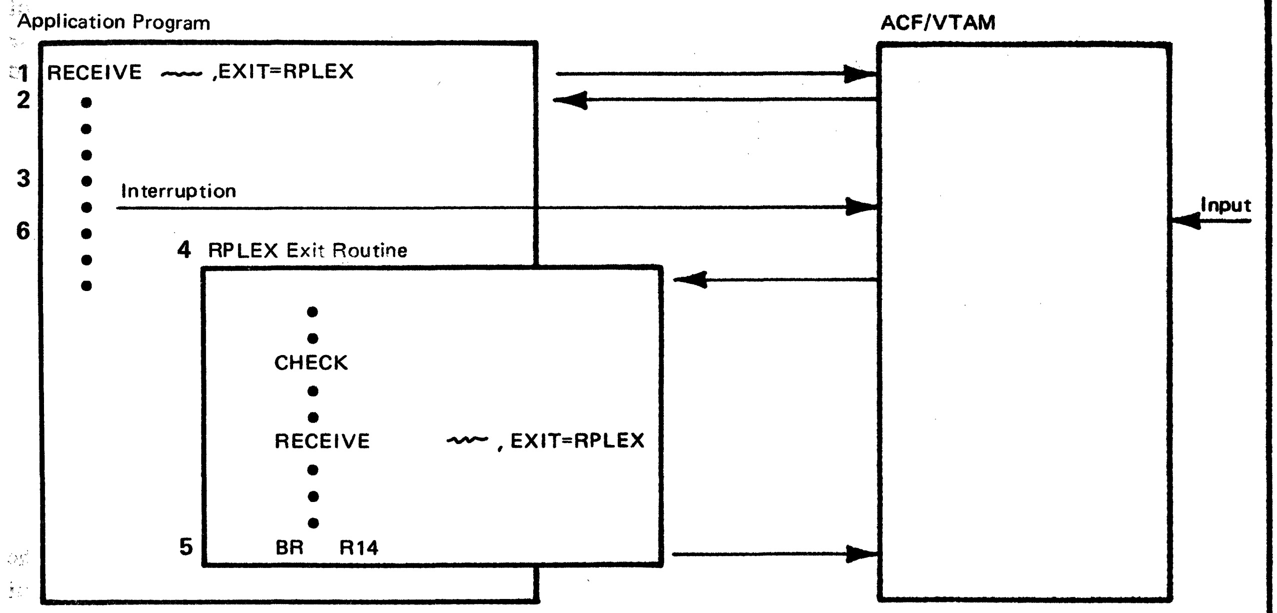
The instructions to be executed when an RPL-based operation is completed can be written as a separate routine. This routine, called an RPL exit routine, can be specified in the RPL-based macro instruction that requests the operation. The address of the exit routine is specified in the EXIT operand of the macro instruction or is placed in the EXIT field of the RPL. When the requested operation is completed, ACF/VTAM schedules and causes entry to the RPL exit routine. When ACF/VTAM gives control to the RPL exit routine, the routine cannot be interrupted even though other pending events are completed; the exit routine must return control to ACF/VTAM before ACF/VTAM can return control to other parts of the application program, including other exit routines that ACF/VTAM may have scheduled. (A LERAD or SYNAD exit routine, however, can be entered if an error or special condition occurs during an RPL-based request that is issued in the RPL exit routine.) Figure 7-1 illustrates the use of an RPL exit routine.

Designating a routine as an RPL exit routine is an alternative to having ACF/VTAM post an ECB when an asynchronous event is completed. A program can use one or the other technique exclusively, or it can use a mixture of ECB-posting and RPL exit routines. Sample Program 2 in Part 3 shows an example of an RPL exit routine. The same RPL exit routine can be designated by more than one macro instruction; in other words, an RPL exit routine can be established as a common exit routine.

If the application program also uses ECBs, the RPL exit routine may post an ECB related to the logical unit being communicated with, so that the main program will later discover that an event is completed. Since it may be necessary to reuse the RPL associated with the request whose completion caused entry to the exit routine (for example, for reissuance of a RECEIVE request within the exit routine) and because it is a means of causing entry to a LERAD or SYNAD exit routine if an error occurs, a CHECK macro instruction may be required in the exit routine. If the RPL does not have to be reused, the CHECK macro can be in the main program, perhaps following the discovery of the posted ECB associated with the logical unit.

How EXLST Exit Routines Work

This type of exit routine differs from the RPL exit routine in being a special-purpose exit routine. The special purpose is understood by both the ACF/VTAM application program and ACF/VTAM. Instead of being specified in a particular macro instruction request, the identity of an EXLST exit routine is established only when the exit list in which its name is specified is identified to ACF/VTAM, either when the program is opened or, for certain types of exit routines, when a logical unit is connected. In general, EXLST exit routines



1 The ACF/VTAM application program issues an asynchronous RECEIVE request, which passes control to ACF/VTAM. The request specifies the scheduling of the RPLEX when the operation is completed. ACF/VTAM accepts the request and returns control to the program at the next sequential instruction (**2**).

2 The program continues execution until input arrives, and ACF/VTAM interrupts the program when the input arrives.

3 ACF/VTAM schedules RPLEX as the next exit routine. Since an exit routine is not currently being executed, RPLEX is immediately given control (**4**).

4 RPLEX is executed without any other part of the program gaining control. A CHECK macro instruction is issued to mark the RPL action. A RECEIVE is issued to read input again. It is an asynchronous request specifying that RPLEX be scheduled when the operation is completed. (If more input arrives and the operation is completed, RPLEX is to be scheduled but not reentered until after it finishes and returns control to ACF/VTAM.

5 RPLEX, having completed its job, returns control to ACF/VTAM.

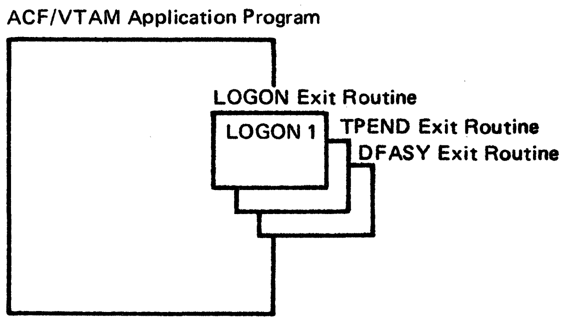
6 If the RECEIVE in RPLEX has not been completed, ACF/VTAM returns control to the main program that was interrupted at **3**. If the RECEIVE in RPLEX has been completed, RPLEX is again given control.

Figure 7-1. An Example of Using an RPL Exit Routine

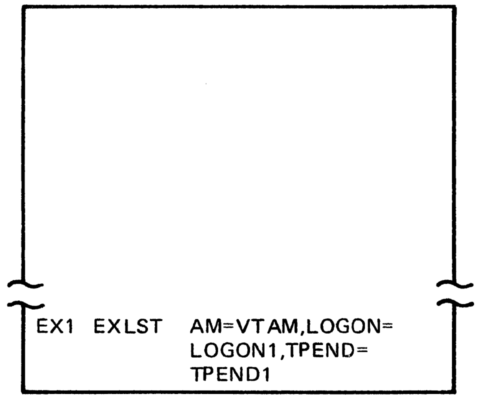
are special-purpose exit routines, entered only when a somewhat unusual event occurs, such as the network operator's issuance of a HALT command to shut down the network.

Here is how EXLST exit routines work:

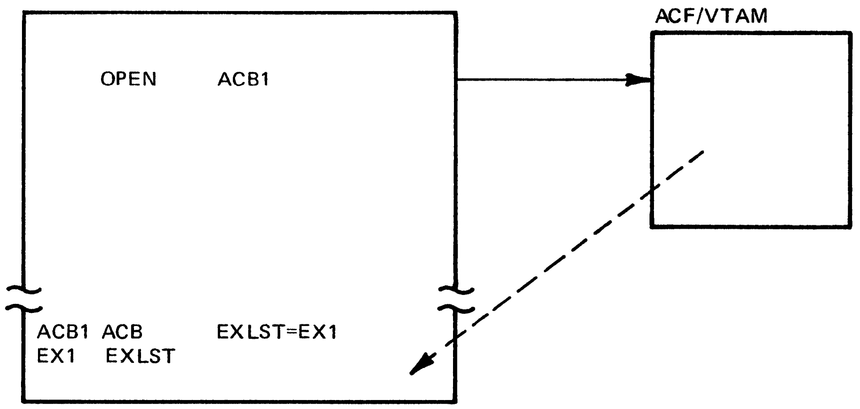
1. An ACF/VTAM application contains a number of exit routines written for different purposes (for example, a LOGON exit routine and a TPEND exit routine).



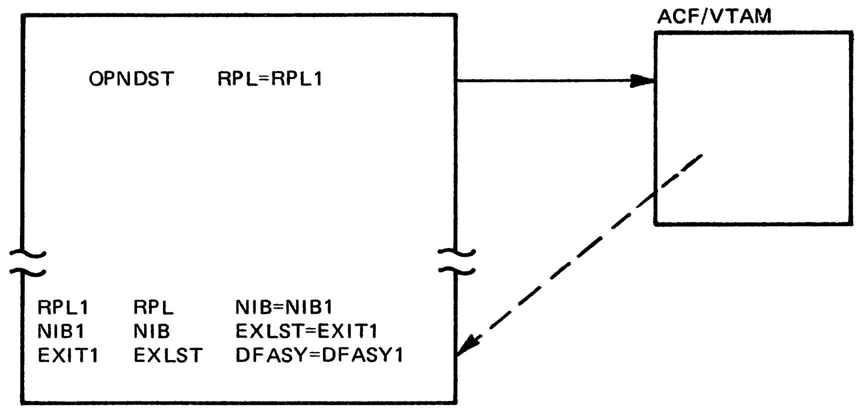
2. The program names the special-purpose exit routines and puts their names in an exit list. The exit list is created with the EXLST macro instruction. Each exit routine name is specified with an appropriate ACF/VTAM-provided operand, such as LOGON and TPEND.



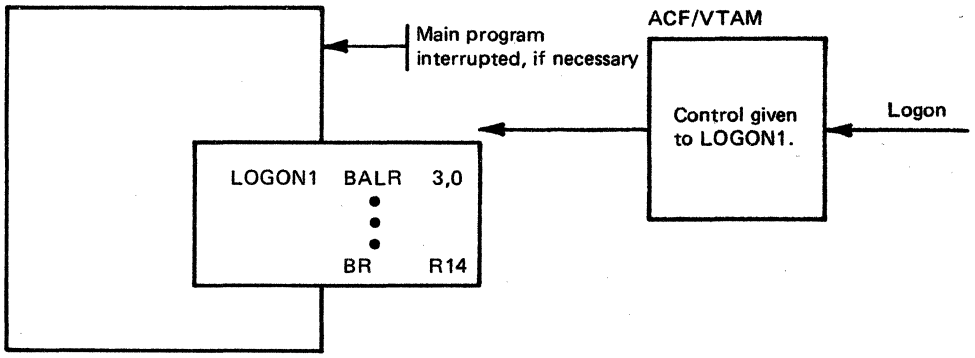
3a. This exit list, identified by the name of the EXLST macro, can be specified in the EXLST operand of the program's ACB. When the ACB is opened, the list of exit routines becomes available to ACF/VTAM.



3b. Alternatively, certain types of exit routines—DFASY, SCIP, and RESP—can be specified in the EXLST operand of the NIB that is used when a logical unit is connected. After the logical unit is connected (that is, after OPNDST is completed), ACF/VTAM will use an exit routine identified in the NIB exit list in preference to the corresponding exit routine specified in the ACB exit list. The preference applies only for the logical unit represented by the NIB at connection. If an appropriate exit routine is not in the exit list passed during connection, ACF/VTAM will look in the ACB-specified exit list that was passed for the entire program when the ACB was opened.



4. When an event occurs for which a related exit routine exists, ACF/VTAM schedules the appropriate exit routine, using the exit routine address (name) that it has been provided. As soon as no other exit routine is being executed or scheduled, the exit routine is given control (if necessary, interrupting the main portion of the program).



A Summary of ACF/VTAM Application Program Exit Routines

Figure 7-2 summarizes exit routines by showing the purpose of each type of exit routine and how the address of each type is specified to ACF/VTAM.

Deciding Whether and How to Use Exit Routines

The use of exit routines is optional. An RPL exit routine is an alternative to having a routine that is branched to in the main program following the posting of an ECB by ACF/VTAM. EXLST exit routines are not absolutely required, although some are designed for common use and should normally be included in an application program. These EXLST exit routines are designed for common use:

LERAD

LOGON, in a primary application program when requests for connection (logons) can be expected

LOSTERM

NSEXIT

SCIP

SYNAD

TPEND

The following EXLST exit routine is required only if the facility associated with it is required:

RELREQ, in a primary application program if the application program is to be notified when another program requests control of a logical unit that is already connected to the first program

These EXLST exit routines are optional in that they are an alternatives to other facilities:

DFASY, rather than having to issue a RECEIVE specifying RTYPE=DFASY in the main program and branching to a related routine on completion

RESP, rather than having to issue a RECEIVE specifying RTYPE=RESP in the main program and branching to a related routine on completion

Type of Exit Routine	Purpose	How the Exit Routine's Address Is Specified	Type of Exit List That Routine's Name May Appear In
RPL Exit Routine	Any purpose	Code the address in the EXIT operand of an RPL macro or in the request that uses the RPL.	(Not applicable)
EXLST Exit Routines (Each type is listed below.)	Special purposes	Code the names of the exit routines in an EXLST macro instruction. The list that is created is then identified in either the E EXLST operand of an ACB or NIB macro.	
DFASY	Receive expedited-flow input (for example, a Quiesce at End of Chain command) from a logical unit without requiring an outstanding RECEIVE specifying RTYPE=DFASY.	Code the name in the DFASY operand of the EXLST macro.	ACB or NIB
LERAD	Handle logical errors that may occur as the result of a request.	Code the name in the LERAD operand of the EXLST macro.	ACB only
LOGON	Handle a request for connection to the application program that ACF/VTAM has received from a logical unit.	Code the name in the LOGON operand of the EXLST macro.	ACB only
LOSTERM	Handle the situation of a logical unit's being unexpectedly lost to the program, or notify the application program of other unusual conditions that can affect the session.	Code the name in the LOSTERM operand of the EXLST macro.	ACB only
NSEXIT	Handle a situation in which (1) a request for a procedure has been positively responded to but the procedure cannot be completed, (2) ACF/VTAM has initiated session termination because of a session outage, or (3) some other kind of network services request unit is received.	Code the name in the NSEXIT operand of the EXLST macro.	ACB only
RELREQ	Handle a request from another application program for a logical unit that is presently connected to the program that contains the RELREQ exit routine.	Code the name in the RELREQ operand of the EXLST macro.	ACB only
RESP	Receive a response from a logical unit without requiring an outstanding RECEIVE specifying RTYPE=RESP.	Code the name in the RESP operand of the EXLST macro.	ACB or NIB
SCIP	Receive and process one of the following session-control commands: Clear Start Data Traffic (SDT) Request Recovery (RQR) Set and Test Sequence Numbers (STSN) Bind Unbind	Code the name in the SCIP operand of the EXLST macro.	ACB or NIB
SYNAD	Handle a physical error or special condition that occurs as the result of a request.	Code the name in the SYNAD operand of the EXLST macro.	ACB only
TPEND	Handle the closing of the program that is required when the network operator halts the network or ACF/VTAM terminates abnormally.	Code the name in the TPEND operand of the EXLST macro.	ACB only

Figure 7-2. A Summary of Exit Routines

If an EXLST exit routine is not provided and the event or condition that the routine handles occurs, the user may never learn of the event or condition. In some cases, the user might learn of the event or condition through return codes or information in the RPL.

Note: *Only exit routines that can be recognized by ACF/VTAM can be specified in the EXLST macro instruction. Non-ACF/VTAM exit routines (such as VSAM exit routines) cannot be specified in the macro instruction.*

RPL Exit Routines

In general, it is easier to have ACF/VTAM schedule and enter an RPL exit routine than it is to write instructions that, after an ECB has been posted, determine which ECB was posted, keep track of which ECB to check next, wait if no ECB is yet posted, and branch to an appropriate routine when an ECB is found to be posted. There is another advantage to an RPL exit routine even if ECB-handling is also used: Certain requests can be given priority by having an RPL exit routine scheduled rather than an ECB posted. The RPL exit routine can be entered sooner than the same logic can be branched to after the main program discovers the posted ECB.

Specifying the DFASY, RESP, and SCIP Exit Routines in an ACB or NIB

Certain EXLST exit routines—DFASY, RESP, and SCIP—can be in a list that is associated either with an ACB (identified in the EXLST operand of an ACB) or with a NIB (identified by the EXLST operand of a NIB). ACB-specified exit routines are used by ACF/VTAM for all logical units connected to the program represented by the ACB. NIB-specified exit routines are used by ACF/VTAM only for the logical unit whose NIB specifies the exit routine when the logical unit is connected. For details on how ACF/VTAM handles DFASY and RESP input, see Figures 7-3 and 7-4. Several logical units can share the same list of DFASY, SCIP, and RESP exit routines or the list can be unique for each logical unit.

Here is an example of the use of both ACB- and NIB-specified exit routines in the same program. Program A has a list of exit routines that are common to all logical units connected to the program. These include the LOGON, TPEND, LERAD, NSEXIT, SYNAD, LOSTERM, and SCIP exit routines. These are defined in an EXLST macro instruction whose address is specified in the program's ACB macro instruction. In addition, because the program plans to handle DFASY and RESP input from some logical units differently from similar input from others, the program has two lists of DFASY and RESP exit routines. (One list will be used to process signals from logical units at one location and the other list will be used for signals received from another location.) Prior to connecting a logical unit, its logon message can help determine its location, and the related exit-list address can be placed in the EXLST field of the NIB, using MODCB.

DFASY Exit Routine

The DFASY exit routine provides a way for ACF/VTAM to notify an application program that an expedited-flow command has arrived. The expedited-flow commands that can be received by a primary application program are:

Quiesce at End of Chain (QEC)

Release Quiesce (RELQ)

Request Shutdown (RSHUTD)

Shutdown Complete (SHUTC)

Signal

Stop Bracket Initiation (SBI)

The expedited-flow commands that can be received by a secondary application program are:

Quiesce at End of Chain (QEC)

Release Quiesce (RELQ)

Shutdown (SHUTD)

Signal

Stop Bracket Initiation (SBI)

For information on these commands, see Appendix B.

If a DFASY exit routine is specified, either in the EXLST operand of the NIB used for connection or of the ACB (and PROC=DFASYX was specified in the NIB), whenever an expedited-flow command arrives from the logical unit associated with the NIB, ACF/VTAM schedules the DFASY exit routine. The manner in which ACF/VTAM handles an expedited-flow command is shown in Figure 7-3.

Using a DFASY exit routine is an alternative to getting each expedited-flow command with a RECEIVE macro instruction that contains RTYPE=DFASY or that includes DFASY among other RTYPE parameters. The program can maintain an active RECEIVE with RTYPE=DFASY to get each expedited-flow command, but use of the RECEIVE requires an active RPL before the command arrives. Use of the RECEIVE also requires the program to include coding that, after completion of the RECEIVE, determines which command has been received and branches to the routine that processes that command.

Using a DFASY exit routine may be more convenient. This frees the application program from having to issue the RECEIVE and from tying up storage for the active RPL. The disadvantage is that scheduling an exit routine requires more ACF/VTAM execution time than posting an ECB. However, since DFASY input occurs less frequently than normal-flow input, that disadvantage may be outweighed by the convenience of having the exit routine.

Whether or not the application program must send a response to the expedited-flow command depends on the setting of a PROC option in the NIB when the connection was made:

If PROC=APPLRESP was specified in the NIB at connection, the application program sends the response, using SEND ...,STYPE=RESP,CONTROL=*command code of received command*,RESPOND=(*response operands*). Sense information may also be returned for a negative response.

If PROC=SYSRESP was specified in the NIB at connection, ACF/VTAM automatically sends the response before presenting the command to the application program.

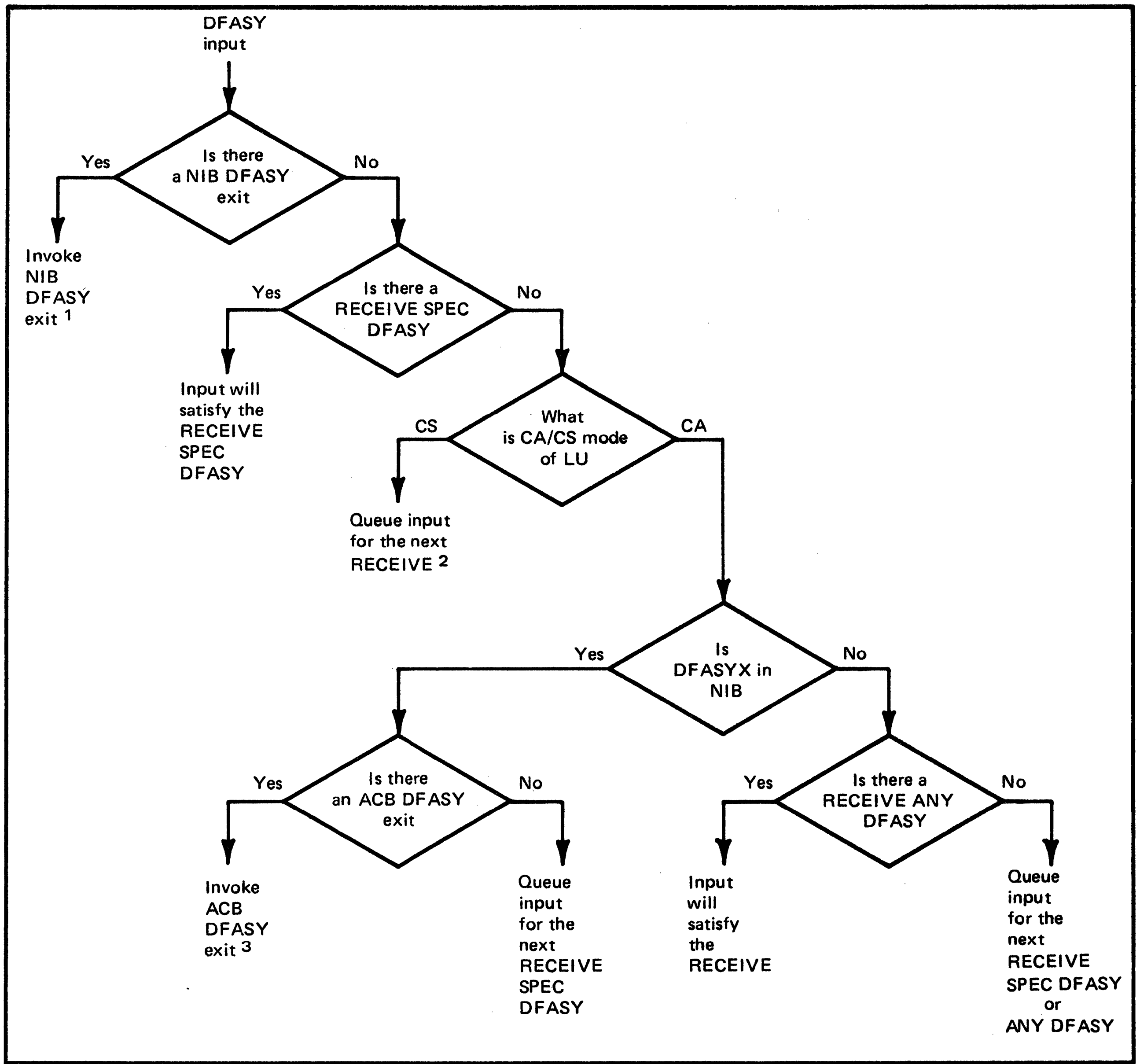
For a DFASY exit routine, information on the expedited-flow command that has been received is available in a read-only RPL provided by ACF/VTAM. The location of the read-only RPL is provided in the parameter list passed to the exit routine when the routine is scheduled.

Registers Upon Entry: When the DFASY exit routine receives control, register 1 contains the address of a 5-word parameter list (the parameter list is summarized in Figure 7-5):

The first word contains the address of the ACB of the application program to which the expedited-flow command was sent.

The second word contains the CID of the logical unit that sent the command.

The third word contains whatever has been placed in the USERFLD field of the NIB associated with that logical unit.



DFASY input
Is there a NIB DFASY exit
Yes
No
Invoke NIB DFASY exit 1
Is there a RECEIVE SPEC DFASY
Yes
No
Input will satisfy the RECEIVE SPEC DFASY
What is CA/CS mode of LU
CS
CA
Queue input for the next RECEIVE 2
Is DFASYX in NIB
Yes
No
Is there an ACB DFASY exit
Yes
No
Invoke ACB DFASY exit 3
Queue input for the next RECEIVE SPEC DFASY
Is there a RECEIVE ANY DFASY
Yes
No
Input will satisfy the RECEIVE
Queue input for the next RECEIVE SPEC DFASY or ANY DFASY

1 The exit routine is scheduled if no other exit routine (including the NIB DFASY exit routine) is currently running. If another exit routine is running, the input is queued for the NIB DFASY exit routine.

2 The input will satisfy a RECEIVE SPEC DFASY. The input can also be obtained by a RECEIVE ANY DFASY if the mode has been switched to CA for the session with the logical unit.

3 The exit routine is scheduled if no other exit routine (including the ACB DFASY exit routine) us currently running. If another exit routine is running, the input is queued for the ACB DFASY exit routine.

Figure 7-3. How ACF/VTAM Handles DFASY (Expedited-Flow) Input

The fourth word is reserved.

The fifth word contains the address of an ACF/VTAM-supplied, read-only RPL. Other than the fact that it resides in read-only ACF/VTAM storage and cannot be used by an RPL-based macro instruction, the read-only RPL is identical to any other RPL. The application program can examine the read-only RPL fields with SHOWCB and TESTCB macro instructions or by using assembler instructions and the IFGRPL DSECT. The read-only RPL feedback fields are set exactly as they would be following a RECEIVE (RTYPE=DFASY) macro instruction, except that the REQ field is not set. A CHECK macro instruction must not be issued against the read-only RPL.

Other general purpose registers contain the following:

Register 14: The address in ACF/VTAM to which the DFASY routine must branch when it has finished processing. ACF/VTAM will handle the return of control to the instruction in the application program that was about to be executed when the DFASY interruption occurred.

Register 15: The address of the DFASY routine.

Registers 0 and 2-13: Unpredictable.

RESP Exit Routine

The RESP exit routine provides a way for ACF/VTAM to notify an application program when a response to a normal-flow message (data or command) has arrived.

Using a RESP exit routine is one of three ways an application program can be notified of receipt of a normal-flow response. The other two ways are:

Specifying POST=RESP in the macro instruction used to send the normal-flow message. If this is done, the macro instruction is not completed until the response is received.

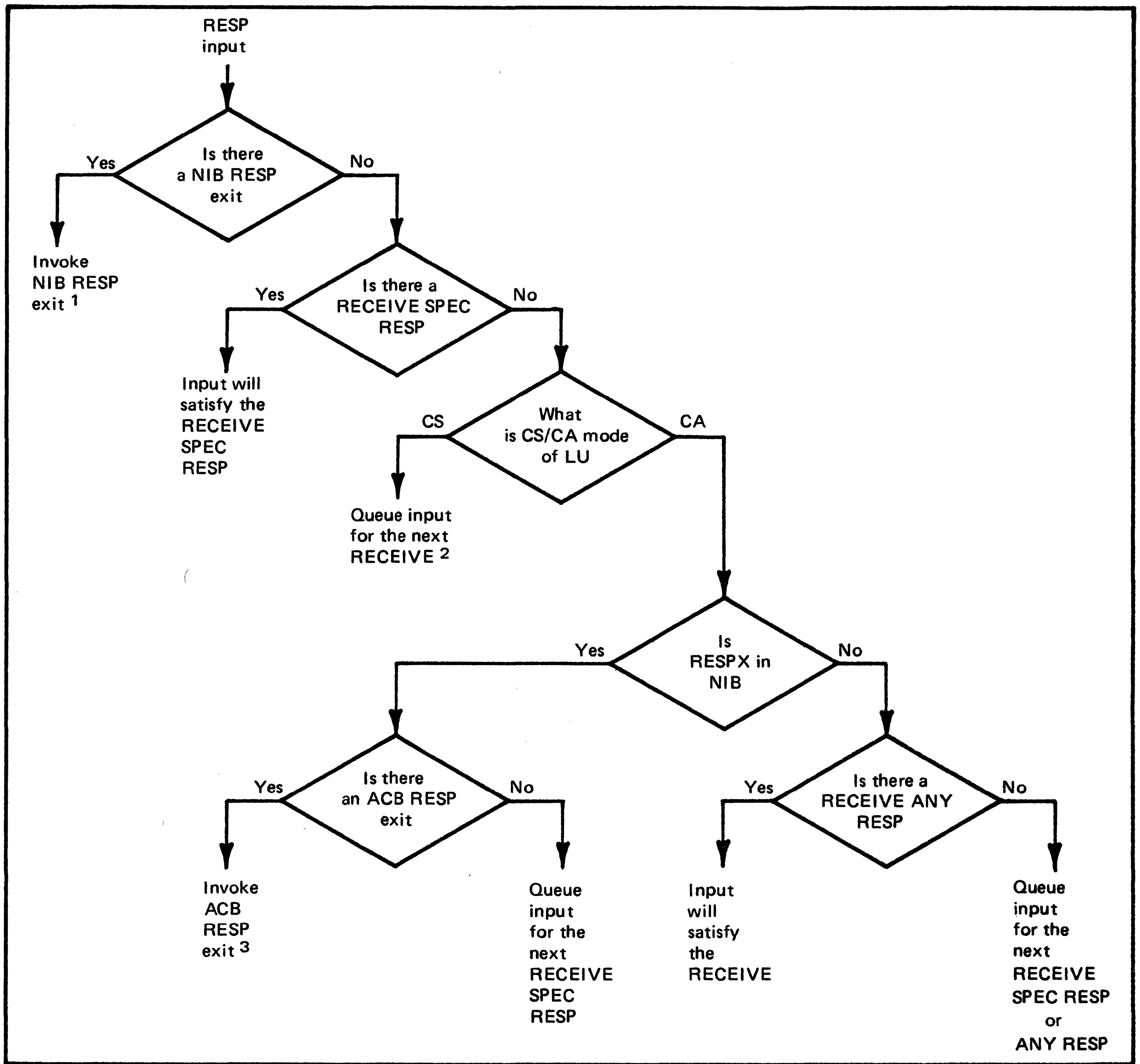
Maintaining an active RECEIVE with RTYPE=RESP. The RECEIVE is completed (and reissued) each time a normal-flow response is received.

If a RESP exit routine is specified (either in the EXLST operand of the NIB used for connection or of the ACB) and PROC=RESPX was defined in the NIB at connection, ACF/VTAM schedules the RESP exit routine whenever a normal-flow response is received from the logical unit associated with the NIB. This frees the main part of the application program from having to issue a RECEIVE with RTYPE=RESP or, if a RECEIVE is issued that will receive any kind of input (RECEIVE with RTYPE=(DFSYN,DFASY,RESP)), from having to distinguish the type of input that was received and branching to the appropriate routine. The disadvantage of having a RESP exit routine is that the system must execute more instructions to schedule an exit routine than to post an ECB. Howver, since RESP input occurs less frequently than normal-flow input; this disadvantage may be outweighted by the convenience of having a RESP exit routine. The way in which ACF/VTAM handles RESP input is shown in Figure 7-4. A RESP exit routine is shown in Appendix D.

For a RESP exit routine, information on the normal-flow response that has been received is available in a read-only RPL provided by ACF/VTAM. The location of the read-only RPL is provided in the parameter list passed to the exit routine when the routine is scheduled.

Registers Upon Entry: When the RESP exit routine receives control, the register contents are the same as those described above for the DFASY exit routine. That is:

Register 1: The address of a parameter list containing the ACB address, the logical unit's CID, the USERFLD data, a reserved word, and the address of the read-only RPL. The parameter list is summarized in Figure 7-5.


RESP input
Is there a NIB RESP exit
Yes
No
Invoke NIB RESP exit 1
Is there a RECEIVE SPEC RESP
Yes
No
Input will satisfy the RECEIVE SPEC RESP
What is CS/CA mode of LU
CS
CA
Queue input for the next RECEIVE 2
Is RESPX in NIB
Yes
No
Is there an ACB RESP exit
Yes
No
Invoke ACB RESP exit 3
Queue input for the next RECEIVE SPEC RESP
Is there a RECEIVE ANY RESP
Yes
No
Input will satisfy the RECEIVE
Queue input for the next RECEIVE SPEC RESP or ANY RESP

1 The exit routine is scheduled if no other exit routine (including the NIB RESP exit routine) is currently running. If another exit routine is running, the input is queued for the NIB RESP exit routine.

2 The input will satisfy a RECEIVE SPEC RESP. The response can also be obtained by a RECEIVE ANY RESP if the mode has been switched to CA mode for the session with the logical unit.

3 The exit routine is scheduled if no other exit routine (including the ACB RESP exit routine) is currently running. If another exit routine is running, the input is queued for the ACB RESP exit routine.

Figure 7-4. How ACF/VTAM Handles RESP (Normal-Flow Response) Input

Exit Routine	1st Word	Register 1 Parameter List 2nd Word	3rd Word	4th Word	5th Word
LERAD	None (Register 1 contains the RPL address for the request that failed)				
SYNAD	None (Register 1 contains the RPL address for the request that failed)				
DFASY	ACB address	CID	USERFLD data	Unused	Read-only RPL address
RESP	ACB address	CID	USERFLD data	Unused	Read-only RPL address
SCIP	ACB address	CID except for Bind. For Bind, this word is reserved.	USERFLD data except for Bind. For Bind, this word is reserved.	Unused except for Bind. For Bind, address of session parameters.	Read-only RPL address
TPEND	ACB address	Reason-terminated code			
RELREQ	ACB address	Address of the terminal's symbolic name			
LOGON	ACB address	Address of the terminal's symbolic name	Unused	Length of logon message	
LOSTERM	ACB address	CID	USERFLD data	Reason-lost code	
NSEXIT	ACB address	(Contents depend on type of network services request unit received. See description of NSEXIT exit routine.)			Read-only RPL address

Figure 7-5. Summary of Parameter Lists Passed to Exit Routines

Register 14: The address in ACF/VTAM to which the RESP exit routine must return when it is finished processing. ACF/VTAM returns control to the instruction in the application program that was about to be executed when the RESP interruption occurred.

Register 15: The address of the RESP exit routine.

Registers 0 and 2-13: Unpredictable.

LERAD Exit Routine

A LERAD exit routine is included in an application program (and identified in an ACB exit list) when the application program wants a routine to be automatically invoked when a logical error (in contrast to a physical error) is detected.

Generally, a logical error results when an RPL-based request is made that is inherently contradictory–like attempting to use an invalid CID. (Errors that occur because of hardware malfunctions, for example, are not logical errors. These errors are handled by the SYNAD exit routine.)

If the SYN option code is in effect when the logical error occurs or if the request cannot be accepted because of a logical error, the LERAD exit routine is entered immediately; otherwise, if the ASY option code is in effect, the routine is not scheduled until a CHECK macro instruction is issued for the operation in which the error occurred. One exception: If the ASY option code is set, the request is accepted by ACF/VTAM, and then ACF/VTAM determines that it cannot post the RPL (perhaps because the ACB has been overwritten), ACF/VTAM abnormally terminates the application program.

Before the LERAD exit routine is given control, ACF/VTAM sets a recovery action return code of 20 or 24 (decimal) in register 0 and in the RTNCD field of the RPL and sets a specific error return code in the FDBK2 field indicating the specific cause of the error. These return codes are explained in Appendix C of *ACF/VTAM Macro Language Reference.*

If the application program has no LERAD exit routine and a logical error occurs, ACF/VTAM simply returns control to the next sequential instruction. ACF/VTAM places a return code of 4 in register 15 and a recovery action return code of 20 or 24 (decimal) in register 0 and in the RTNCD field of the RPL. It also sets a specific error return code in the FDBK2 field of the RPL indicating the specific cause of the error. These codes are explained in Appendix C of *ACF/VTAM Macro Language Reference.*

If the application program issues RPL-based requests in both the main program and in the exit routines, the LERAD exit routine may be reentered by ACF/VTAM. The routine may likewise be reentered if any RPL-based requests are issued in the LERAD exit routine itself. In these situations, the exit routine must be reeneterable.

When the LERAD exit routine returns control to ACF/VTAM, ACF/VTAM leaves registers 0 and 15 intact so that the routine can pass information back in these registers to the main part of the application program.

Registers Upon Entry: When the LERAD routine receives control, the general purpose registers contain the following:

Register 0: A recovery action return code (refer to Appendix C in *ACF/VTAM Macro Language Reference).*

Register 1: The address of the RPL associated with the request. If the recovery action return code in register 0 is set to 24 (decimal), ACF/VTAM was unable to place an indicator in the FDBK2 field specifying the reason for the error. This happens in three cases: A macro has been issued whose RPL is already in use, CHECK has been issued for a request whose RPL exit routine has not yet been scheduled, or an invalid RPL was specified (for example, the RPL address is invalid or the RPL is overlaid). For descriptions of the return codes placed in FDBK2, see Appendix C in *ACF/VTAM Macro Language Reference.*

Register 13: The address of an 18-word save area supplied by the programmer when the macro instruction that caused the logical error was issued. If the exit routine is going to return control via register 14, it must not change anything in the save area. This means that if any macro instruction is issued in the exit routine, register 13 must first be loaded with the address of a new save area. Furthermore, before control is returned via register 14, register 13 must be restored with the value it had when the exit routine was invoked.

Register 14: The address in ACF/VTAM to which the LERAD exit routine can branch when it has finished processing. When the exit routine branches to this address, ACF/VTAM handles the retuning of control to the next sequential instruction in the application program following the request (or following the CHECK macro instruction issued for the request). The LERAD exit routine can branch to any part of the main program because the routine is executed under the same system task control block as the main program. (Care should be taken, however, to eventually branch to the register 14 address if LERAD was entered from an RPL-based request issued in another exit routine.) If the routine returns control to the next sequential instruction by branching on the register 14 address, ACF/VTAM restores the registers from the save area whose address is in register 13.

Register 15: The address of the LERAD routine.

Registers 2-12: Unmodified; whatever was in them when the macro instruction was issued is still there.

SYNAD Exit Routine

A SYNAD exit routine is included in an application program (and identified in an ACB exit list) when a routine is to be automatically invoked when a physical error is detected. A physical error is an unrecoverable input or output error or other unusual condition that occurs during an I/O operation. The SYNAD exit routine, if specified, is entered for all recovery action return codes of 4, 8, 12, and 16 (decimal).

If the SYN option code is in effect when the error occurs or if the request cannot be accepted, the SYNAD exit routine is entered immediately; otherwise, if the ASY option code is in effect, the routine is not invoked until a CHECK macro is issued for the operation in which the error occurred.

The SYNAD exit routine can examine the REQ field of the RPL and determine the type of request that caused the routine to be invoked. Each RPL-based macro instruction (except CHECK and EXECRPL) has its own REQ code. The SYNAD exit routine can analyze the FDBK2 field and attempt to recover from the error.

If the application program has no SYNAD exit routine and a physical error occurs, ACF/VTAM simply returns control to the next sequential instruction with return codes in registers 0 and 15.

If the application program issues RPL-based requests in both the main program and the exit routines, the SYNAD exit routine may be reentered by ACF/VTAM. The routine may likewise be reentered if RPL-based requests are issued in the exit routine itself. In these situations, the exit routine must be reenterable.

When the SYNAD exit routine returns control to ACF/VTAM, ACF/VTAM leaves registers 0 and 15 intact; this enables the routine to pass information back in those registers to the main part of the application program.

Registers Upon Entry: When the SYNAD routine receives control, the general purpose registers contain the following:

Register 0: A recovery action return code (see Appendix C in *ACF/VTAM Macro Language Reference*).

Register 1: The address of the RPL associated with the request.

Register 13: The address of an 18-word save area supplied by the programmer when the macro instruction that caused the physical error was issued. If the exit routine is going to return control via register 14, it must not change anything in the save area. This means that if any macro instruction is issued in the exit routine, register 13 must first be loaded with the address of a new save area. Furthermore, before control is returned via register 14, register 13 must be restored with the value it had when the exit routine was invoked.

Register 14: The address in ACF/VTAM to which the SYNAD exit routine can branch when it has finished processing. When the exit routine branches to this address, ACF/VTAM handles the return of control to the next sequential instruction following the request (or following the CHECK macro issued for the request). The SYNAD exit routine can branch to any part of the main program. (Care should be taken, however, to eventually return to the register 14 address if SYNAD was entered from an RPL-based request issued in another exit routine.) If the application program eventually returns to the next sequential instruction by branching on the register 14 address, ACF/VTAM restores the registers from the save area whose address is in register 13.

Register 15: The address of the SYNAD routine.

Registers 2-12: Unmodified; whatever was in them when the macro instruction was issued is still there.

Special Considerations for LERAD and SYNAD Exit Routines

LERAD and SYNAD exit routines are not required. If a macro instruction that specifies an RPL is issued, one of these two exit routines, if present, is entered if an error occurs. If the exit routine does not exist, ACF/VTAM in any case provides feedback information in registers 0 and 15 and in appropriate RPL fields. The return code in register 0 enables the next sequential instruction in the program to determine whether a logical error or one of several other general types of errors occurred; the program can itself then branch to an appropriate routine. The chief advantage in using LERAD and SYNAD exit routines is that they provide a convenient way to organize sets of error and special condition-handling logic that serve all requests in the program.

The same name can be specified for the program's LERAD and SYNAD exit routines. The common exit routine can determine after it is entered whether a logical or some other error or special condition occurred.

A discussion of the kinds of logic that these routines might contain is provided in Chapter 9, "Handling Errors and Special Conditions." Coded LERAD and SYNAD exit routines are shown in Appendix D.

LOGON Exit Routine

A program that expects a logon from one or more logical units can handle the logon either by having ACF/VTAM complete a pending OPNDST specifying OPTCD=ACCEPT in the main program or by having ACF/VTAM schedule a LOGON exit routine. The LOGON exit routine enables the program to examine a logon message or make other inquiries of ACF/VTAM, using the INQUIRE macro instruction, before connecting the logical unit with an OPNDST specifying OPTCD=ACCEPT. If the logical unit's request for connection is to be rejected, the LOGON exit routine may wish to connect the logical unit, send a message, and then disconnect the logical unit. (Even if the logical unit is not temporarily connected with an OPNDST, rejection of its logon must include disconnecting it, using a CLSDST.)

VTAM queues a logon if (1) the logical unit has issued an Initiate command or a character-coded logon, (2) a terminal issues a logon via the network solicitor, (3) another application program to which the logical unit is currently connected issues a CLSDST macro instruction with OPTCD=PASS, (4) the application program issues a SIMLOGON macro instruction on behalf of the logical unit, (5) the user has specified automatic logon for the logical unit, when the network was defined or the network operator has specified an automatic logon using the VARY command, or (6) another application program acting as a secondary end of a session has issued a REQSESS macro instruction. These cause the LOGON exit routine to be scheduled if SETLOGON with OPTCD=START is in effect.

Note: *When a logical unit logs on, ACF/VTAM first checks for an outstanding OPNDST request (that is, OPNDST with ACCEPT and Q) that has not yet been completed. If there is no outstanding OPNDST request, ACF/VTAM and schedules a LOGON exit routine if an active one exists. Thus, a logon will not cause a LOGON exit routine to be scheduled if there is a pending OPNDST with ACCEPT. If no LOGON exit routine exists, the logon is queued.*

Regardless of the mechanism by which the LOGON exit routine is scheduled, the routine is in effect being asked to connect the logical unit to the application program. The routine's principal task therefore is to determine whether it should honor the request and, when it determines that it should, issue an OPNDST macro instruction to establish connection with the logical unit. If the request is not to be honored, the routine should issue the CLSDST macro instruction for the logical unit (which removes the logical unit from the logon queue). If neither OPNDST nor CLSDST is issued, the logical unit may remain unconnected to any application program.

If MACRF=LOGON was specified in the ACB and SETLOGON with OPTCD=QUIESCE has not been issued, logons are queued for the application program regardless of whether a LOGON exit routine is available. A logon remains queued until the program issues OPNDST or CLSDST for the logical unit. Note that the "queuing" of a logon does not necessarily mean that the logon is queued for eventual scheduling of the LOGON exit routine; it merely means that the logon is queued for an eventual OPNDST with OPTCD=ACCEPT macro instruction (or CLSDST).

The LOGON exit routine can issue an INQUIRE macro instruction to obtain the session parameters and the user logon message supplied by the logical unit that is logging on. If the routine determines from the session parameters and the logon message that connection with the logical unit is acceptable, it may wish to establish that connection. This is accomplished by using information passed to the LOGON exit routine, along with information obtained with the INQUIRE macro instruction, to build or modify a NIB and an RPL, and by then issuing the OPNDST macro instruction with ACCEPT and SPEC option codes.

The LOGON exit routine is entered only if MACRF=LOGON was specified for the ACB, and the application program has issued the SETLOGON with OPTCD=START macro instruction.

Registers Upon Entry: When the LOGON exit routine receives control, register 1 contains the address of a 4-word parameter list (the parameter list is summarized in Figure 7-5):

The first word contains the address of the ACB to which the logon request was directed. The ACB address should be specified for the ACB operand of an INQUIRE macro instruction used to obtain the data portion of the logon.

The second word contains the address of the 8-byte symbolic name of the logical unit requesting logon. This name should be placed in the NAME field of the NIB used to establish connection with the logical unit. The symbolic name being pointed to here is the same as the name that was specified in the NAME field of the definition statement for the logical unit. LU, TERMINAL, and COMP are ACF/VTAM definition statements used by the user to define logical units and terminals.)

The third word is reserved.

The fourth word contains the length of the data portion of the logon sent by the logical unit. This length should be used with the LENGTH operand of INQUIRE macro instruction to obtain the data portion of the logon.

Other registers contain the following:

Register 14: The address in ACF/VTAM to which the LOGON exit routine should branch when it is through processing. ACF/VTAM handles the return of control to the application program instruction that was about to be executed when the LOGON interruption occurred.

Register 15: The address of the LOGON exit routine.

Registers 0 and 2-13: Unpredictable.

A LOGON exit routine is shown in Appendix D.

LOSTERM Exit Routine

A LOSTERM exit routine is scheduled by ACF/VTAM when contact with a logical unit has been lost, when a logical unit has requested a logoff, when certain errors are detected in transmission, or when a logical unit is temporarily unavailable. As noted below, the application program may or may not issue CLSDST to disconnect the logical unit. (If the application program fails to issue CLSDST, the logical unit will remain unavailable for connection to any other application program.)

If a LOSTERM exit routine is not provided, ACF/VTAM posts any outstanding requests associated with affected logical units with an appropriate return code. If there are no outstanding requests, whenever the program makes the next request, it is posted with a lost-terminal return code. If there is a LOSTERM exit routine, the program can disconnect the logical unit. (When a logical unit is lost, ACF/VTAM stops sending to that unit but does not disconnect it.) As with LERAD and SYNAD exit routines (LOSTERM might be thought of as a special form of SYNAD exit routine), its advantage is having a more convenient and immediate way to have control passed to this part of the program.

A LOSTERM exit routine is especially recommended for an application program that does not issue specific-mode I/O requests for its logical units, but is driven instead by input arriving as the result of RECEIVE macro instructions issued in the any-mode. Use of the exit routine is also recommended for an application program that issues specific-mode I/O requests when there is the possibility that the logical unit may fill ACF/VTAM's buffers faster than the application program is emptying them with RECEIVE macro instructions.

When ACF/VTAM determines that it can attempt to restart a logical unit, the LOSTERM exit routine may be entered twice. (This action can occur only when the application program does not have an NSEXIT exit routine.) On first entry, a reason code of 24 is passed to the LOSTERM exit routine, indicating that ACF/VTAM has begun its attempt to restart the logical unit. After the attempt is completed, ACF/VTAM gives control to the LOSTERM exit routine a second time (unless the CLSDST macro instruction has already been completed). On this second entry, the reason code is either 16 (the logical unit has been successfully restarted) or 12 (the attempt was unsuccessful and contact with the logical unit has been lost). Even when the restart is successful, the application program must issue a CLSDST macro instruction to end the session that was disrupted and then, if desired, establish connection again with the logical unit that was successfully restarted. A coded LOSTERM exit routine is shown in Appendix D.

If the application program has an NSEXIT exit routine, the conditions listed below for reason-codes 12, 16, and 24, are reported to the NSEXIT exit routine (with a network services cleanup request unit) instead of to the LOSTERM exit routine. If the application program does not have an NSEXIT exit routine but does have a LOSTERM exit routine, those conditions are reported to the LOSTERM exit routine.

Registers Upon Entry: When the LOSTERM exit routine receives control, register 1 contains the address of a 4-word parameter list (the parameter list is summarized in Figure 7-5):

The first word contains the address of the ACB of the application program to which the logical unit or terminal is connected.

The second word contains the session's CID. The ARG field of an RPL used for CLSDST must contain this CID.

The third word contains whatever had been placed in the USERFLD field of the NIB associated with the logical unit or terminal.

The value contained in the fourth word (called the *reason code*) indicates why the LOSTERM exit routine was entered:

LOSTERM Reason Code (Decimal)	Meaning
0	A dial-line disconnection occurred for a dial-in BSC or start-stop terminal. A CLSDST macro instruction is required.
4	A dial-line disconnection occurred for a dial-out BSC or start-stop terminal. If no data from the terminal remains in ACF/VTAM buffers, a READ or WRITE (OPTCD=SPEC) macro instruction will redial the terminal. If redialing fails (causing the LOSTERM exit routine to be rescheduled), the CLSDST macro instruction should be issued for the terminal.

LOSTERM Reason Code (Decimal)	Meaning
8	Reserved.
12	Contact with a BSC terminal, start-stop terminal, or logical unit was permanently lost for one of the following reasons: (1) The network operator has issued a VARY INACT command for the logical unit or terminal. For VARY INACT,F or R, the LOSTERM exit routine is entered twice: once with a code of 24 and once with a code of 12. (*Note:* If the device lost was a PU or a 3705, the VARY INACT,R causes the LOSTERM exit routine to be scheduled with a code of 16 instead of 12.) (2) The communications controller's NCP has begun an automatic network shutdown or has abended and cannot be restarted. (3) There has been a permanent channel failure between the CPU and the communications controller or locally attached terminal. (4) There has been a failure in the network path between the communications controller and the remotely attached terminal. (5) The network operator has issued a HALT NET,QUICK command. (6) A test request message has been received from the terminal other than a 3270 (see *ACF/VTAM TOLTEP,* SC38-0283, for more information about test request messages). For logical units, ACF/VTAM automatically issues an Unbind command. For local 3270, BSC, and start-stop terminals, the program can issue READ macro instructions to obtain data already sent from the terminal. A CLSDST macro instruction is required unless a CLSDST has already been executed successfully. (*Note:* If the program has an NSEXIT exit routine, these conditions are in some cases reported instead to that exit routine.)
16	The logical unit has been successfully recontacted. ACF/VTAM has automatically issued an Unbind command for the application program. Issue a CLSDST macro instruction. If desired, the program can issue an OPNDST or SIMLOGON macro instruction to re-acquire the logical unit. (*Note:* If the program has an NSEXIT exit routine, this condition is reported instead to that exit routine.) Note: *Once the CLSDST macro instruction has been issued, reconnection of the terminal is subject to the normal rules for acquisition. Therefore, if another application program has a connection request queued for the logical unit, or the logical unit has issued a connection request for another application program, the logical unit may not be immediately reconnected to the releasing application program.*
20	An unconditional Terminate command, an unconditional character-coded logoff, or an unconditional TERMSESS macro instruction has been issued by the logical unit. ACF/VTAM automatically issues an Unbind command for the application program. A CLSDST macro instruction is required.
24	Contact with the logical unit has been lost but ACF/VTAM may be able to reestablish it. Stop output to the logical unit and either return to ACF/VTAM or issue a CLSDST macro instruction. If the program does not issue a CLSDST, it must return to ACF/VTAM; ACF/VTAM will attempt to recontact the logical unit. If recontact is successful, ACF/VTAM reschedules the LOSTERM exit routine with a return code of 16. If recontact is unsuccessful, the LOSTERM exit routine is rescheduled with a return code of 12. If the program issues a CLSDST for the logical unit, the LOSTERM exit routine might not be rescheduled and the program might not get return code 12 or 16. (*Note:* If the program has an NSEXIT exit routine, this condition is not reported to the LOSTERM exit routine.)
28	Reserved.
32	A conditional Terminate command, a conditional character-coded logoff, or a conditional TERMSESS macro instruction has been issued by the logical unit. The application program may take any action it desires including issuing a CLSDST for the logical unit.
36	The buffer limit defined for a logical unit has been exceeded. ACF/VTAM automatically issues a Clear command for the application program. Any data for which the application program has not issued a RECEIVE will be discarded. The application program may resynchronize sequence numbers, or the data may be retransmitted after issuing SESSIONC (CONTROL=SDT) if appropriate for the transmission services profile specified by the session parameters.
40	The operator at a BSC 3270 or local 3270 terminal has hit the Test Request Key. A CLSDST macro instruction is required.

Note: *For any of the LOSTERM reason codes that require or recommend a CLSDST macro instruction, do not issue a second CLSDST if one has already been issued to the same logical unit or terminal but for a different reason.*

Other general purpose registers contain the following:

Register 14: The address in ACF/VTAM to which the LOSTERM exit routine must branch when it is through processing. ACF/VTAM handles the return of control to the point in the application program where the LOSTERM interruption occurred.

Register 15: The address of the LOSTERM exit routine.

Registers 0 and 2-13: Unpredictable.

NSEXIT Exit Routine

The NSEXIT exit routine is entered whenever a network services request unit arrives for an application program. Since an application program can specify only one NSEXIT exit routine, this same routine must serve both when the program is the primary end of a session and when it is the secondary end. The action taken by the exit routine depends on the type of network services request unit received by the program.

An application program can receive either of two types of network services request units:

The program receives a *network services procedure error request unit* if, after the program has issued a session establishment request and the request has been posted complete, something happens that makes it impossible to perform the next step in setting up the session. (See "Network Services Procedure Error" below.)

The program receives a *network services cleanup request unit* when a connection with a logical unit has been broken because of a session outage (for example, a link failure or an NCP failure) or because the network operator has issued a VARY INACT,F or VARY INACT,R command for the logical unit. (See "Cleanup Conditions" below.)

When the exit routine is scheduled, ACF/VTAM provides it with the address of a read-only RPL. The AREA field of the RPL contains the address of the request unit that was received, and the RECLEN field of the RPL tells the number of bytes in the request unit. The exit routine examines the request unit to determine which type of network services request unit was received, which determines what action it should take.

In any future releases of ACF/VTAM, other types of network services request units may be passed to the NSEXIT exit routine. For that reason, the exit routine should be coded to determine the particular type of request unit received and to take action for each type. The exit routine should also take particular action when it receives a request unit other than one of the types it expects to receive. (In other words, the exit routine should not, by default, do nothing when it receives a request unit other than a type that is expected.) If the exit routine receives a request unit other than a procedure error request unit or a cleanup request unit, the exit routine should set register 0 to 0 and register 15 to 4 and then return control to ACF/VTAM.

Network Services Procedure Error

As indicated above, a network services procedure error (NSPE) request unit can arrive at an application program when, after having received a positive response to a session establishment request, the program is awaiting the next event in the session establishment procedure. Here are some examples of conditions that cause a network services procedure error request unit to be generated and delivered to an application program:

1. A secondary application program has issued a REQSESS macro instruction, and the macro has been completed successfully (indicating that a positive response to the request was sent back). The primary application program then rejects the logon by

issuing a CLSDST macro instruction. Issuance of the CLSDST macro causes a network services procedure error request unit to be sent to the secondary application program.

2. A secondary application program has issued a REQSESS macro instruction, and the macro has been completed successfully. The primary application program is then abnormally terminated before it can process the logon which resulted from the REQSESS. ACF/VTAM sends a network services procedure error request unit to the secondary program.
3. A primary application program issues a SIMLOGON macro instruction for a logical unit, and the macro is completed successfully (indicating that a logon for the logical unit has been created by ACF/VTAM and queued for the application program that issued the macro). Before the logon can be processed, the network operator deactivates the logical unit. This causes ACF/VTAM to send the application program a network services procedure error request unit.
4. Application program A issues a CLSDST macro instruction with OPTCD=PASS to pass a logical unit to application program B. The macro completes successfully, indicating that a logon has been created and queued for application program B. When application program B processes the logon, it either (1) rejects the logon by issuing a CLSDST macro or (2) issues an OPNDST to the logical unit, but the logical unit rejects the Bind command by sending a negative response. In either case, ACF/VTAM sends a network services procedure error request unit to application program A. The request unit signals application program A that, even though the CLSDST with OPTCD=PASS was posted complete, the session that was requested cannot be accomplished.
5. When a primary application program issues an OPNDST with OPTCD=ACQUIRE and for some reason ACF/VTAM cannot establish the session, ACF/VTAM may send an NSPE to the primary application program. Since the application program may receive the NSPE either before or after the processing associated with the OPNDST is completed (either successfully or unsuccessfully), the application program's NSEXIT exit routine should be written to take appropriate error recovery regardless of when the NSPE is received.

The format of the network services procedure error request unit is shown in Figure 7-6.

In some situations, such as conditions 1 and 2 described above, the application program may want to issue another session establishment request immediately or may want to wait and issue the request at a later time. Even if no other action is taken, the NSEXIT exit routine should set registers 0 and 15 to 0 before returning control to ACF/VTAM. This is done so that those registers can be used for return codes when processing other types of network services request units in any future releases of ACF/VTAM.

Cleanup Conditions

When a session is interrupted either by a session outage or by a VARY INACT command with the F (forced) or R (reactivate) operand, ACF/VTAM sends the primary application program a network services cleanup request unit. A network services cleanup request unit is also sent to the secondary end of the session when the secondary end is an application program. When the secondary end of the session is a device-type logical unit, ACF/VTAM performs a cleanup sequence consisting of deactivation of the logical unit and an attempt to reactivate the logical unit.

Arrival of the cleanup request unit at an application program causes that program's NSEXIT exit routine to be scheduled (if one exists). Because ACF/VTAM has already terminated the session, the exit routine does not take any action to end the session (that is, does not issue a CLSDST macro instruction or send a Request Shutdown command). The exit routine may want to clean up control blocks for the session. The exit routine

Byte **Contents**

0 X'01'

1 X'06'

2 X'04'

3 Reason code

The meaning of the bits in this code are:

0123 4567	
1...	An internal processing error occurred in trying to reach the primary logical unit.
.1..	A Bind error occurred in reaching the secondary logical unit.
..1.	Initiation was rejected at the primary logical unit.
...1	Initiation was rejected at the secondary logical unit.
.... 0...	A setup procedure error occurred.
.... .0..	Reserved.
.... ..1.	Initiation was rejected at the system services control point.
....1	The request unit is in the comprehensive format (rather than the condensed format).

In the current release of ACF/VTAM, bits 4 and 7 (setup error and comprehensive format) are always 0 and 1, respectively. If bit 4 is not 0 or bit 7 is not 1, set register 0 to 0 and register 15 to 4 and return to ACF/VTAM.

4-5 System sense data
(if applicable)

6-7 User sense data
(if applicable)

The system and user sense data, if applicable, is from the step in the procedure that caused the setup failure. For the meaning of the system sense data, see Appendix C in *ACF/VTAM Macro Language Reference.*

8 X'06'

9-n Identification of the logical units involved in the failed procedure, as follows:

1 Byte	1 Byte	1-8 Bytes	1 Byte	1 Byte	1-8 Bytes
X'F3'	*l1*	Symbolic name of primary logical unit (1-8 characters)	X'F3'	*l2*	Symbolic name of secondary logical unit (1-8 characters)

l1 is the length (number of characters) of the symbolic name of the primary logical unit.

l2 is the length (number of characters) of the symbolic name of the secondary logical unit.

Figure 7-6. Format of a Network Services Procedure Error Request Unit

may also want to attempt to reestablish the session, and the attempt may be successful if the session outage has been repaired or bypassed and the desired logical unit is available. The format of the network services cleanup request unit is shown in Figure 7-7.

When the primary application program involved in the broken session does not have an NSEXIT exit routine, that program's LOSTERM exit routine (if one exists) is scheduled to report loss of the session. The LOSTERM exit routine is scheduled first with reason code 24 (contact lost) and is later rescheduled with reason code 16 (logical unit successfully restarted) or reason code 12 (restart was unsuccessful).

Even if no other action is taken, the NSEXIT exit routine should set registers 0 and 15 to 0 before returning to ACF/VTAM. This is done so that those registers can be used for return codes when processing other types of network services request units in any future releases of ACF/VTAM.

Registers Upon Entry: When the NSEXIT exit routine receives control, register 1 contains the address of a 5-word parameter list (the parameter list is summarized in Figure 7-5):

The first word contains the address of the ACB for the application program to which the network services request unit was sent.

Word 2 contains the CID for the session referred to in the cleanup request unit. Word 2 is not applicable for a network services procedure error request unit.

Word 3 contains the data that was placed in the USERFLD field of the NIB when the session referred to in the cleanup request unit was established (that is, when the OPNDST macro was issued by the primary application program or when the OPNSEC macro was issued by the secondary application program). Word 3 is not applicable for a network services procedure error request unit.

The fourth word is reserved.

The fifth word contains the address of an ACF/VTAM-supplied, read-only RPL. Other than the fact that it resides in read-only storage and cannot be used by an RPL-based macro instruction, the read-only RPL is identical to any other RPL. The application program can examine the read-only RPL fields with SHOWCB and TESTCB macro instructions or with assembler instructions.

Other general purpose registers contain the following:

Register 14: The address in VTAM to which the NSEXIT routine must branch when it has finished processing. VTAM will return control to the instruction in the application program that was about to be executed when the NSEXIT interruption occurred.

Register 15: The address of the NSEXIT exit routine.

Registers 0 and 2-13: Unpredictable.

TPEND Exit Routine

The TPEND exit routine is entered when the network operator issues a HALT command, when ACF/VTAM is halting itself in an orderly fashion because of an internal problem, or when ACF/VTAM is being abnormally terminated. The reason for entry to the exit routine is indicated by a code in the second word of the parameter list passed to the exit routine.

For a standard HALT command (a HALT command without the QUICK or CANCEL operand), indicated by code 0 in the parameter list, the program is allowed to continue communications with connected logical units, but the program should end those communications in an orderly fashion as soon as it can. It should issue an asynchronous

Byte	Contents
0	X'81'
1	X'06'
2	X'29'
3-4	Reserved
5	Reason code In the current release of ACF/VTAM, this byte always contains X'03', indicating that the session has been taken down. In any future releases of ACF/VTAM, additional reason codes may be returned.
6	X'06'
7-n	Identification of the logical units involved in the session, in this format:

1 Byte	1 Byte	1-8 Bytes	1 Byte	1 Byte	1-8 Bytes
X'F3'	*l1*	Symbolic name of primary logical unit (1-8 characters)	X'F3'	*l2*	Symbolic name of secondary logical unit (1-8 characters)

l1 is the length (number of characters) of the symbolic name of the primary logical unit.

l2 is the length (number of characters) of the symbolic name of the secondary logical unit.

Figure 7-7. Format of a Network Services Cleanup Request Unit

CLSDST macro instruction for each connected logical unit, return to its main program, and issue a CLOSE macro instruction. (A CLOSE macro instruction cannot be issued in an exit routine.)

For a HALT QUICK command or when ACF/VTAM is halting itself (code 4), pending data-transfer operations are stopped, but they are marked as completed and canceled (ACF/VTAM sets a flag in the FDBK2 field of each RPL to indicate that the operation was canceled). For code 4 (as for code 0), the application program should issue a CLSDST macro instruction for each connected logical unit and then issue the CLOSE macro instruction.

For a HALT CANCEL command or ACF/VTAM abnormal termination (code 8, which appears only in an OS/VS system), pending operations are interrupted (without being marked as completed or canceled), and no ACF/VTAM request except the CLOSE macro instruction is accepted. The TPEND exit routine should return to the main program for immediate issuance of the CLOSE macro instruction without any attempt to disconnect the logical units.

See Chapter 4 for more information on actions to be taken by the TPEND exit routine. A coded TPEND exit routine is shown in Appendix D.

Registers Upon Entry: When the TPEND exit routine receives control, register 1 contains the address of a 2-word parameter list (the parameter list is summarized in Figure 7-5):

The first word contains the address of the ACB of the application program being shut down.

The value in the second word indicates the reason for the shutdown:

0 The network operator issued a standard HALT command to close the network normally.

4 The network operator issued a HALT QUICK command, or ACF/VTAM detected an internal problem and is halting itself.

8 The network operator issued a HALT CANCEL command, or ACF/VTAM has abnormally terminated.

Other general purpose registers contain the following:

Register 14: The address in ACF/VTAM to which the TPEND exit routine must branch when it is through processing. ACF/VTAM returns control to the instruction in the application program that was to be executed when the TPEND interruption occurred.

Register 15: The address of the TPEND exit routine.

Registers 0 and 2-13: Unpredictable.

RELREQ Exit Routine

The RELREQ exit routine is entered when one application program (a set of instructions associated with one ACB) or TOLTEP (the teleprocessing online test executive program) requests connection to a logical unit that is connected to another application program (a set of instructions associated with a different ACB). The requesting program requests connection with a SIMLOGON macro instruction that specifies OPTCD=(RELRQ,Q). As a result, ACF/VTAM schedules and causes entry to the RELREQ exit routine of the application program currently connected to the logical unit and acting as the primary end of the session. The RELREQ exit routine can either ignore the request (that is, remain connected to the logical unit and make the requesting program wait) or take action to immediately release the requested logical unit.

If the exit routine decides to release the logical unit, it may want to determine whether there are any pending (incomplete) data-transfer requests for the logical unit and release it only after those data-transfer operations have been completed. To disconnect and release the logical unit, the application program issues the CLSDST macro instruction with the RELEASE option. After execution of the CLSDST macro, the logical unit is made available to the application program that has the oldest pending request for the logical unit. (Note that the application program to which the logical unit is made available may be a different application program from the one that caused the current entry to the RELREQ exit routine. This will be the case when another application program made an earlier connection request and the request was queued.)

If the exit routine decides to ignore the RELREQ request, it takes no action and continues communication with the logical unit. The connection request from the other application program remains pending and is queued (behind any other pending connection request for the logical unit) until the logical unit is released.

If an application program does not have a RELREQ exit routine, the program cannot be notified of another program's request. If the other program issued its SIMLOGON request with the NQ option, ACF/VTAM rejects the request. If the other program issued the request with the Q option, the request remains pending until the logical unit is released.

The application program that caused entry to the RELREQ exit routine may have issued its SIMLOGON request with the CONANY option in effect and provided a list of NIBs from which any one logical unit is acceptable. This kind of request is satisfied if one of the logical units is immediately available or when the first of the logical units is released and thus becomes available. When no logical unit is immediately available, ACF/VTAM invokes the RELREQ exit routine (or queues the connection request) for each application program currently connected to one of the logical units. Thus, for a SIMLOGON with the RELREQ and CONANY options, many RELREQ exit routines may be invoked. In this situation, if another application program releases one of the logical units first, but your program also releases one of the logical units, the logical unit released by your program may remain unconnected. To prevent this, the CLSDST that releases the logical unit should be followed by an OPNDST (OPTCD=ACQUIRE) or SIMLOGON request to attempt to reacquire the logical unit that was just released. Then, if the released logical unit is being ignored, the program that released it gets it back.

Registers Upon Entry: When the RELREQ exit routine receives control, register 1 contains the address of a 2-word parameter list (the parameter list is summarized in Figure 7-5):

The first word of the parameter list contains the address of the ACB through which the logical unit is currently connected to an application program.

The second word of the parameter list contains the address of the symbolic name of the requested logical unit. The name is 8 bytes long and padded on the right with blanks, if necessary.

The other registers contain the following:

Register 14: The address in ACF/VTAM to which the RELREQ routine must branch when it is through processing. ACF/VTAM will return control to the instruction in the application program that was about to be executed when the RELREQ interruption occurred.

Register 15: The address of the RELREQ exit routine.

Registers 0 and 2-13: Unpredictable.

SCIP Exit Routine

The SCIP exit routine is entered when any of the following session-control commands is received by an application program:

Clear

Start Data Traffic (SDT)

Request Recovery (RQR)

Set and Test Sequence Numbers (STSN)

Bind

Unbind

For the Clear, RQR, and Unbind commands, ACF/VTAM automatically sends a response before the command is presented to the exit routine. For the STSN and Bind commands, the application program must send its own response. For the SDT command, either the application program or ACF/VTAM sends the response, depending on the NIB's SDT operand for the session. For all six commands, if the application program has no SCIP exit routine, ACF/VTAM automatically sends a negative response with sense information indicating that the request was rejected because the function is disabled.

Five of the commands—Clear, SDT, STSN, Bind, and Unbind—are sent only from the primary end of the session (the primary application program) to the secondary end of the session (a device-type logical unit or a secondary application program). Thus, in an application program, those five commands can only be received and processed in a SCIP exit routine in a secondary application program. The other command—RQR—is sent only from the secondary end of a session to the primary end. Thus, that command is only received and processed in a SCIP exit routine in a primary application program.

For a SCIP exit routine, information on the command that has been received is available in a read-only RPL provided by ACF/VTAM. The location of the read-only RPL is provided in the parameter list passed to the exit routine when the routine is scheduled.

Clear Command: The Clear command is sent by the primary end of the session (either ACF/VTAM or the primary application program) when the flow of data messages, data-flow control commands, and responses is to be stopped, either because the primary end is terminating the session or because the primary end wants to take some recovery action. The Clear command does these things:

Informs the secondary end of the session to stop transmitting messages and responses

Causes the inbound and outbound sequence numbers for both ends of the session to reset to 0

Causes all incoming and outgoing messages, commands, and responses pertaining to the session and not yet delivered to be discarded.

For certain logical units[1], issuance of the CLSDST macro instruction in a primary application program causes ACF/VTAM to generate a Clear command and send it to the secondary end of the session. For certain logical units[1], ACF/VTAM also generates a Clear command when it receives a Terminate command from the secondary end of the session. A primary application program usually issues a Clear command (using the SESSIONC macro with CONTROL=CLEAR) when the SCIP exit routine in that program receives a Request Recovery (RQR) command from the secondary end of the session.

[1] ACF/VTAM sends a Clear command before an Unbind command when the logical unit being disconnected is (1) in another domain, or (2) in the same domain and attached to a communications controller (3704 or 3705) that contains Release 5 of the NCP.

As the result of receiving the Clear command, the SCIP exit routine in a secondary application program should take action to stop the program from sending any more messages, responses, or commands.

For more information on the Clear command, see "Controlling Flow" in Chapter 6. For the role of the Clear command in various command sequences, see Figures C-9, C-12, C-13, C-14, C-18, C-21, C-22, C-23, and C-24 in Appendix C.

Start Data Traffic (SDT) Command: When required by the session parameters, a Start Data Traffic (SDT) command is sent from the primary end of the session to the secondary end of the session at the beginning of the session and within a session after successful sequence number resynchronization has occurred. In both cases, the command informs the secondary end of the session that the flow of data messages, data flow commands, and responses can be started (or resumed).

After receipt of a Start Data Traffic command, the SCIP exit routine in a secondary application program should inform the rest of the program that transmissions can be started or pass control to the part of the program that handles beginning-of-session activities.

For more information on the Start Data Traffic command, see "The Node Initialization Block (NIB)" and "Connection with a Secondary Application Program" in Chapter 5, and see "Controlling Flow" in Chapter 6. For the role of the Start Data Traffic command in various command sequences, see Figures C-1, C-2, C-3, C-9, C-15, C-16, C-17, and C-18 in Appendix C.

Request Recovery (RQR) Command: The secondary end of the session sends the Request Recovery command to the primary application program to request the primary program to start sequence number resynchronization. In most cases, the command is sent when the secondary end discovers a discrepancy in the sequence numbers of incoming messages or a discrepancy between the sequence numbers it assigned to outgoing messages and the responses it is receiving to those messages. It might also send the command if it loses or is forced to discard some incoming messages before it can process them. (For example, input buffers are too full to hold the incoming messages.)

The Request Recovery command informs the primary application program that the secondary end of the session wants to resynchronize sequence numbers (that is, agree on which sequence numbers represent the last incoming and outgoing message to be successfully sent and received) and to resend some messages if necessary.

Upon receiving the Request Recovery command, the SCIP exit routine in the primary application program should start resynchronization. That action normally consists of issuing a Clear command (using SESSIONC with CONTROL=CLEAR), followed by one or more Set and Test Sequence Numbers (STSN) commands followed by an SDT command. To each STSN command it receives, the secondary end of the session sends a response indicating the action it has taken in relation to sequence numbers suggested by the primary application program. For a description of the procedure used to resynchronize sequence numbers, see "Controlling Flow" in Chapter 6. For examples of the use of the Request Recovery command, see Figures C-9 and C-18 in Appendix C.

Set and Test Sequence Numbers (STSN) Command: As indicated above, the Set and Test Sequence Numbers command is used by the primary application program in resynchronizing sequence numbers and in message recovery action. For more information on the command, see "Controlling Flow" in Chapter 6 and Figures C-3, C-9, and C-18 in Appendix C.

Bind Command: The Bind command is sent from the primary end of the session to the secondary end of the session during connection. The Bind command indicates that the primary application program wants to start a session with the secondary end, and the command contains the session parameters that the primary program proposes to be used for the session.

Upon receipt of a Bind command, the SCIP exit routine in a secondary application program should inspect the session parameters contained in the command. If the session parameters are acceptable and if the secondary program is willing to go into session with the primary application program, the secondary program should issue the OPNSEC macro instruction (which produces a positive response to the Bind command). If the session parameters are unacceptable or the secondary program does not want the session, it uses the SESSIONC macro to send a negative response to the Bind command. For more information on the Bind command, see Chapter 5, especially the section under "Connection with a Secondary Application Program." Also see Figures C-1, C-2, C-15, C-16, and C-17 in Appendix C.

Unbind Command: As part of the disconnection process (that is, after the primary application program has issued the CLSDST macro instruction), the primary end of the session sends the Unbind command to the secondary end of the session. (In some cases, the Unbind command is preceded by the Clear command–see "Clear Command" above.)

Upon receipt of the Unbind command, the SCIP exit routine in the secondary application program cleans up control blocks and other information pertaining to the session, because that information is no longer needed. For the role of the Unbind command in various command sequences, see Figures C-13, C-14, C-21, C-22, C-23, and C-24 in Appendix C.

Registers Upon Entry: When the SCIP exit routine receives control, register 1 contains the address of a 5-word parameter list (the parameter list is summarized in Figure 7-5):

The first word contains the address of the ACB of the application program to which the command was sent.

The second word contains the CID of the application program that sent the command. (For a Bind command, the contents of this word have no significance.)

The third word contains whatever was placed in the USERFLD field of the NIB at the time the connection was established; that is, when the command is received by a primary application program, this word contains the USERFLD data that was in the NIB associated with the OPNDST macro, and when the command is received by a secondary application program, this word contains the USERFLD data that was in the NIB associated with the OPNSEC macro. (For a Bind command, the contents of this word have no significance.) routine.

The fourth word contains no meaningful data except for receipt of a Bind command. For a Bind command, this word contains the beginning address of the session parameters.

The fifth word contains the address of an ACF/VTAM-supplied, read-only RPL. Other than the fact that it resides in read-only storage and cannot be used by an RPL-based macro instruction, the read-only RPL is identical to any other RPL. The information in the read-only RPL can be examined by using the IFGRPL DSECT or by using the SHOWCB and TESTCB macro instructions. The particular command that caused scheduling of the exit routine can be determined by examining the CONTROL field of the read-only RPL. A CHECK macro instruction must not be issued against the read-only RPL.

Other general purpose registers contain the following:

Register 14: The address in ACF/VTAM to which the SCIP routine must branch when it has finished processing. ACF/VTAM will return control to the instruction in the application program that was about to be executed when the SCIP interruption occurred.

Register 15: The address of the SCIP routine.

Registers 0 and 2-13: Unpredictable.

Summary of Exit Routines Involved in Session Initiation, Session Outages, and Session Termination

Various exit routines play significant roles in session initiation, session outages, and session termination. The involvement of exit routines in those activities is summarized in Figures 7-8, 7-9, and 7-10.

Using Exit Routines When Multitasking

When multitasking, both RPL exit routines and EXLST exit routines are scheduled under the task that opens the ACB with which the exit routines are associated. For example, if the ACB is opened in the main task and a SEND is issued in a subtask that specifies a response to be returned, using a RESP exit routine, the RESP exit routine is scheduled under the main task. The exit routine and the subtask may thus require communication with each other (perhaps by posting and checking an ECB located in a common area).

Procedures to Follow in Writing Exit Routines

Figure 7-11 summarizes the addressability and save-area requirements for the main program and for exit routines. In most cases, LERAD and SYNAD exit routines must be reenterable. Figure 7-12 shows the situations in which LERAD and SYNAD exit routines do not have to be reenterable. Figure 7-13 shows the situations in which they must be reenterable.

Entry Procedures: In general, when an exit routine is entered, the following apply:

- Register 1 contains the address of a parameter list.
- Register 14 contains an address for returning control to ACF/VTAM.
- ACF/VTAM's registers do not have to be saved; register 13 does not contain the address of an ACF/VTAM save area. However, when a LERAD or SYNAD exit routine is entered, register 13 does contain the address of an 18-word save area in the application program.
- ACF/VTAM does not provide a save area for the application program's general registers. If any executable ACF/VTAM macro is issued in the exit routine, the address of the exit routine's own 18-word save area must be in register 13 when the macro is issued.

Cautions, Restrictions, and Techniques: These cautions and techniques can be used when writing exit routines:

- Be sure to establish addressability for each exit routine and for all of the storage that the exit routine uses. There are two techniques for addressing control blocks:
  - Define constants and literals that are within the range of the USING statement by using LTORG.
  - Use A-type address constants for storage that must be shared among the main program and exit routines or that cannot economically be duplicated (for example, save areas and the ACB).

- For DOS/VS users, certain system macros, such as DUMP and PDUMP, cannot be used in an exit routine.
- The OPEN and CLOSE macro instructions cannot be used in an exit routine.
- Although most exit routines cannot be interrupted to be reentered, there are some exceptions. An exit routine must be reenterable if it is associated with ACBs opened by different tasks in the same job step. A LERAD or SYNAD exit routine in most cases must be reenterable. It must be reenterable if:
  - It issues an RPL-based macro.
  - It is being executed because of an error in the main program, but is interrupted to process an error encountered in an exit routine.

  In a reenterable exit routine, storage must be obtained dynamically for control blocks (an RPL, for example) and data.
- If an RPL-based macro is issued in a LERAD or SYNAD exit routine (such as CLSDST, SEND, or EXECRPL), a flag should be set so that, in the event of an error or special condition, the LERAD or SYNAD exit routine will recognize that it has been reentered. This flag can be set in the leftmost bit of any register between register 2 and register 12 that is used to point to the RPL when the request is issued. For example:

```
ST     R2,WORKAREA       R2 CONTAINS RPL ADDR
OI     WORKAREA,X'80'    SET RECURSION FLAG
L      R2,WORKAREA       PUT FLAGGED ADDR BACK IN R2
SEND   RPL=(R2)
```

- If the exit routine issues a macro instruction and completion is awaited in the same routine, the main program as well as the exit routine will wait until the requested operation is completed. To avoid such delays, consider using an RPL exit routine for notification of completion.

Exit Procedures: An exit routine can branch to any location in the main program. The main program, however, must return control to the exit routine (except for LERAD and SYNAD routines). Then, when the exit routine is finished, the following conventions must be observed:

- Except for LERAD and SYNAD, exit routines must return control with a BR 14 after register 14 has been restored with the address it contained when the exit routine was entered (an address within ACF/VTAM).
- For LERAD and SYNAD exit routines, if the program returns control with a BR 14, it must not issue any macro that would change the contents of the 18-word save area whose address is in register 13 unless it first specifies a new save area. Then, when ready to return control, it puts the address of the old save area back into register 13. In other words, when the program returns control with a BR 14, register 13 must be pointing to the same save area it was pointing to at the time the LERAD or SYNAD exit routine was entered.
- A LERAD or SYNAD exit routine can use registers 0 and 15 to pass information to the main program.

Action or Event Causing Connection Request	Exit Routine[1] of Event That Occurs... ...In Primary Application Program	...In Secondary Application Program
Device-type logical unit sends Initiate command or character-coded logon to primary application program.	LOGON exit routine is scheduled.	(Not applicable)
Secondary application program issues REQSESS macro.	LOGON exit routine is scheduled.	If primary application program accepts logon by issuing OPNDST OPTCD=ACCEPT, Bind command is received in SCIP exit routine. If, after REQSESS is posted complete, the session cannot be initiated, NSEXIT exit routine is scheduled with network services procedure error request unit. If primary application program rejects logon by issuing CLSDST macro, NSEXIT exit routine is scheduled with network services procedure error request unit.
Primary application program issues OPNDST OPTCD=ACCEPT.		Bind command received in SCIP exit routine.
Primary application program issues OPNDST OPTCD=ACQUIRE for secondary application program.	If the session cannot be initiated, NSEXIT exit routine is scheduled with network services procedure error request unit. This can occur before or after posting of the OPNDST OPTCD=ACQUIRE.	Bind command received in SCIP exit routine.
Primary application program issues SIMLOGON to create simulated logon.	LOGON exit routine is scheduled to process the simulated logon. If, after the SIMLOGON is posted complete, the session cannot be initiated, NSEXIT exit routine is scheduled with network services procedure error request unit.	If SIMLOGON names a secondary application program and if primary application program accepts the logon by issuing OPNDST OPTCD=ACCEPT, Bind command is received in SCIP exit routine of secondary program.
Primary application program is controlling application program for a device-type logical unit (LOGAPPL=*this program name* in logical unit's definition statement) and another application program releases the logical unit.	LOGON exit routine is scheduled to process the logon created by ACF/VTAM.	
Network operator issues VARY LOGON command to make a primary application program the controlling application program for a device-type logical unit.	LOGON exit routine of primary application program named in VARY LOGON command is scheduled when logical unit is available—immediately if the logical unit is not in session or when the logical unit is released.	
Primary application program A issues a CLSDST OPTCD=PASS to pass a logical unit to primary application program B.	Application program B's LOGON exit routine is scheduled to process the logon. If, after the CLSDST OPTCD=PASS is posted complete, the session cannot be initiated, application program A's NSEXIT exit routine is scheduled with a network services procedure error request unit.	The SCIP exit routine will be entered twice; once by the Unbind received from application A, again by the Bind received from application B.
Primary application program A issues a SIMLOGON with OPTCD=RELREQ to request primary application program B to release a logical unit to which it is currently connected.	Application program B's RELREQ exit routine is scheduled.	

[1] If the program does not have the exit routine, no notification occurs.

Figure 7-8. Summary of Exit Routines Involved in Session Initiation

	Method of Notification	
Action or Event Causing Session Outage[1]	...For Primary Application Program[2]	...For Secondary Application Program[2]
Session outage occurs (for example, a link failure, an NCP failure, or a dial-line disconnection). (The following LOSTERM decimal reason codes are session outages: 0, 4, 12, 16, 24, and 40. When the conditions represented by these reason codes cause scheduling of the NSEXIT exit routine, these codes are not part of the network services cleanup request unit passed to the exit routine; the NSEXIT exit routine receives only the request unit.)	If primary application program has NSEXIT routine, that exit routine is scheduled with a cleanup request unit. Otherwise, the LOSTERM exit routine is scheduled with a reason code. Note that events that can cause double entry to the LOSTERM exit routine (first entry with code 24 and second entry with code 12 or 16) do not cause double entry to the NSEXIT exit routine. Instead, NSEXIT exit routine is entered once with network services cleanup request unit indicating that the session has been lost (no CLSDST is needed because ACF/VTAM has ended the session).	NSEXIT exit routine is scheduled with a cleanup request unit.
Session-type error detected (buffer limit for logical unit exceeded or invalid segmented request received from logical unit).	LOSTERM exit routine is scheduled with a reason code.	(same as primary)
Network operator issues a VARY NET,INACT,I (immediate) command to deactivate a device-type logical unit in the same domain or to deactivate a cross-domain resource (CDRSC) that is a device-type logical unit in another domain.	LOSTERM exit routine is scheduled with a reason code.	(Not applicable)
Network operator issues a VARY NET,INACT,I (immediate) command to deactivate a cross-domain resource (CDRSC) that is a secondary application program or primary application program in another domain.	LOSTERM exit routine is scheduled with a reason code.	Unbind command is received in SCIP exit routine.
Network operator issues a VARY NET,INACT,F or R command to deactivate a device-type logical unit in the same domain or to deactivate a cross-domain resource (CDRSC) that is a device-type logical unit in another domain.	If primary application program has an NSEXIT exit routine, that exit routine is scheduled with a cleanup request unit. Otherwise, LOSTERM exit routine is scheduled with a reason code.	(Not applicable)
Network operator issues a VARY NET,INACT,F or R command to deactivate a cross-domain resource (CDRSC) that is a secondary application program or primary application program in another domain.	If primary application program has an NSEXIT exit routine, that exit routine is scheduled with a cleanup request unit. Otherwise, LOSTERM exit routine is scheduled with a reason code.	NSEXIT exit routine is scheduled with a cleanup request unit.

[1] A session outage is any action or event that causes a path between a primary application program and its logical unit to be broken or that causes loss of one of the participants in the session.

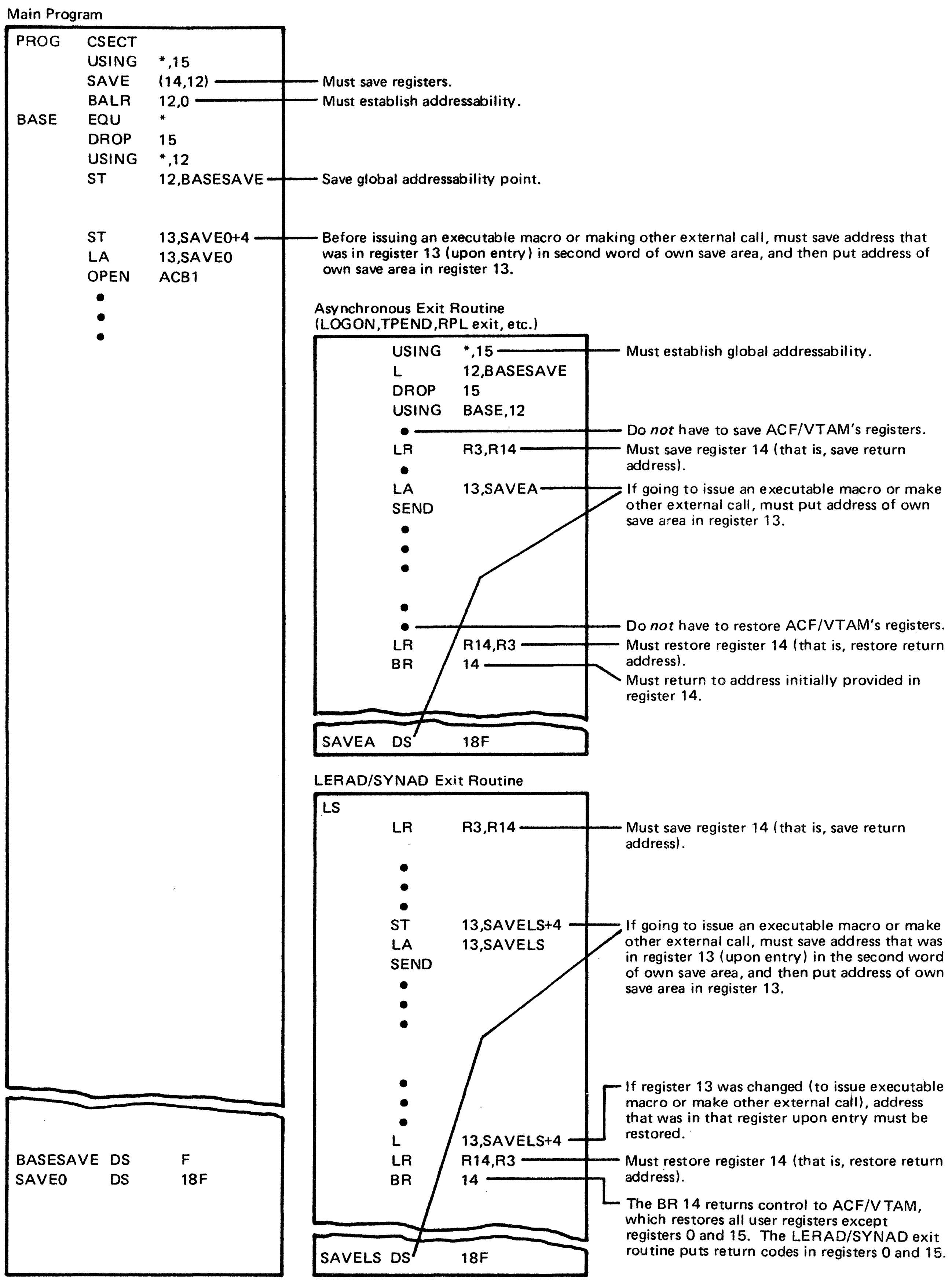
[2] If program does not have an NSEXIT or LOSTERM exit routine, no notification occurs. However, in those cases in which the LOSTERM exit routine is scheduled when there is no NSEXIT exit routine, no notification occurs only when there is neither an NSEXIT nor a LOSTERM exit routine.

Figure 7-9. Summary of Exit Routines Involved in Session Outages

Action or Event Causing Session Termination	Method of Notification...	
	...For Primary Application Program[1]	...For Secondary Application Program[1]
Device-type logical unit requests session termination.	LOSTERM exit routine is scheduled with a reason code.	(Not applicable)
Secondary application program issues TERMSESS macro.	LOSTERM exit routine is scheduled with a reason code.	When primary application program issues CLSDST to end the session, secondary application program receives Unbind command in SCIP exit routine.
Primary application program issues CLSDST or CLOSE	(Not applicable)	Unbind command received in SCIP exit routine.
Secondary application program issues a CLOSE	NSEXIT or LOSTERM exit routine is scheduled.	(Not applicable)
ACF/VTAM servicing the primary application program is being terminated, recognizes an internal error, or receives a HALT command from the operator.	TPEND exit routine is scheduled.	If secondary application program is in same domain as primary application program, secondary program's TPEND exit routine is scheduled.
Network operator issues standard HALT, HALT QUICK, or HALT CANCEL command to halt the ACF/VTAM that is servicing the secondary application program.	If primary application program is in same domain as secondary application program, primary program's TPEND exit routine is scheduled.	TPEND exit routine is scheduled.
ACF/VTAM servicing the secondary application program enters halt-quick processing because of an internal error or the ACF/VTAM is being abnormally terminated.	If the primary application program is in the same domain as the secondary application program, the primary program's TPEND exit routine is scheduled.	TPEND exit routine is scheduled.

[1] If program does not have required exit routine, no notification occurs.

Figure 7-10. Summary of Exit Routines Involved in Session Termination



Main Program

```
PROG     CSECT
         USING   *,15
         SAVE    (14,12)          Must save registers.
         BALR    12,0             Must establish addressability.
BASE     EQU     *
         DROP    15
         USING   *,12
         ST      12,BASESAVE      Save global addressability point.

         ST      13,SAVE0+4       Before issuing an executable macro or making other external call, must save address that
         LA      13,SAVE0         was in register 13 (upon entry) in second word of own save area, and then put address of
         OPEN    ACB1             own save area in register 13.
         •
         •
         •

BASESAVE DS      F
SAVE0    DS      18F
```

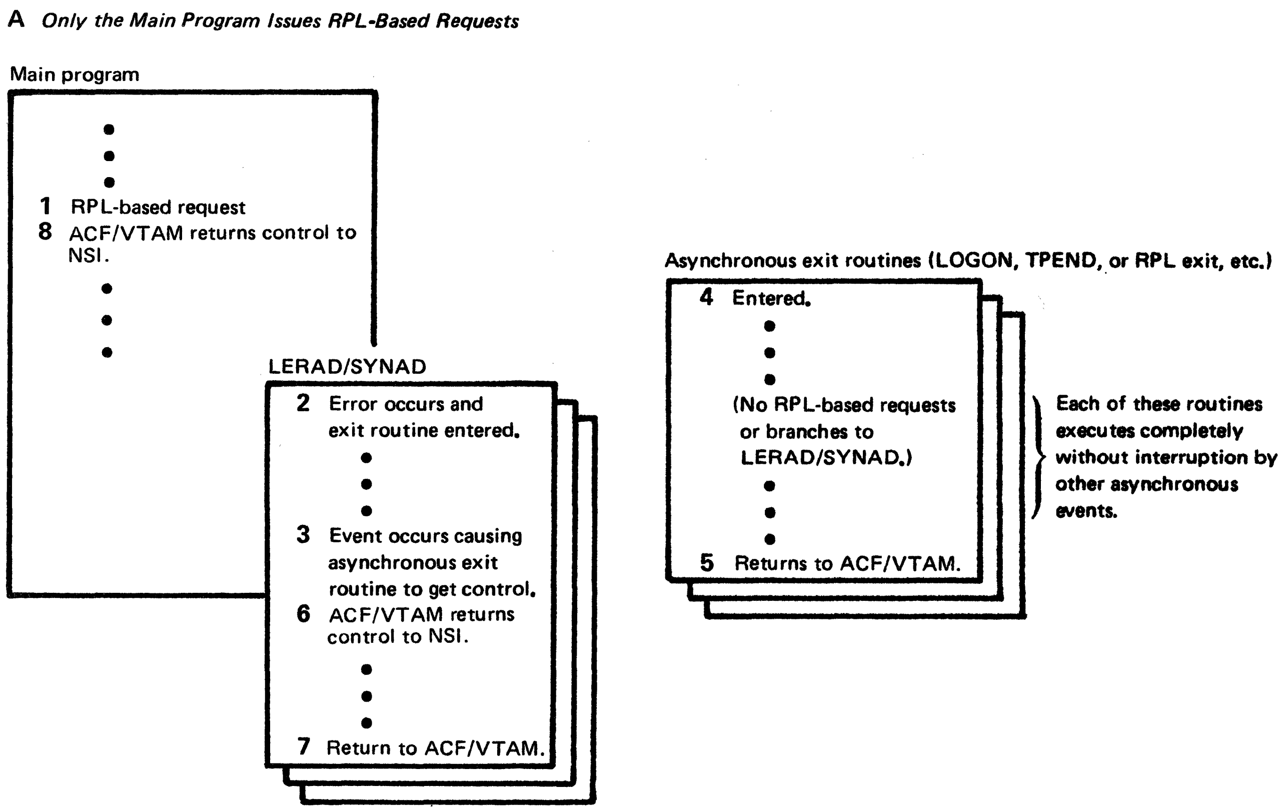
Asynchronous Exit Routine
(LOGON,TPEND,RPL exit, etc.)

```
         USING   *,15             Must establish global addressability.
         L       12,BASESAVE
         DROP    15
         USING   BASE,12
         •                        Do not have to save ACF/VTAM's registers.
         LR      R3,R14           Must save register 14 (that is, save return
         •                        address).
         LA      13,SAVEA         If going to issue an executable macro or make
         SEND                     other external call, must put address of own
         •                        save area in register 13.
         •
         •

         •
         •                        Do not have to restore ACF/VTAM's registers.
         LR      R14,R3           Must restore register 14 (that is, restore return
         BR      14               address).
                                  Must return to address initially provided in
                                  register 14.

SAVEA    DS      18F
```

LERAD/SYNAD Exit Routine

```
LS
         LR      R3,R14           Must save register 14 (that is, save return
                                  address).
         •
         •
         •
         ST      13,SAVELS+4      If going to issue an executable macro or make
         LA      13,SAVELS        other external call, must save address that was
         SEND                     in register 13 (upon entry) in the second word
         •                        of own save area, and then put address of own
         •                        save area in register 13.
         •

         •                        If register 13 was changed (to issue executable
         •                        macro or make other external call), address
         •                        that was in that register upon entry must be
         L       13,SAVELS+4      restored.
         LR      R14,R3           Must restore register 14 (that is, restore return
         BR      14               address).
                                  The BR 14 returns control to ACF/VTAM,
                                  which restores all user registers except
                                  registers 0 and 15. The LERAD/SYNAD exit
SAVELS   DS      18F              routine puts return codes in registers 0 and 15.
```

Figure 7-11. A Summary of Addressability and Save-Area Requirements for the Main Program


A Only the Main Program Issues RPL-Based Requests
Main program
1 RPL-based request
8 ACF/VTAM returns control to NSI.
LERAD/SYNAD
2 Error occurs and exit routine entered.
3 Event occurs causing asynchronous exit routine to get control.
6 ACF/VTAM returns control to NSI.
7 Return to ACF/VTAM.
Asynchronous exit routines (LOGON, TPEND, or RPL exit, etc.)
4 Entered.
(No RPL-based requests or branches to LERAD/SYNAD.)
5 Returns to ACF/VTAM.
Each of these routines executes completely without interruption by other asynchronous events.

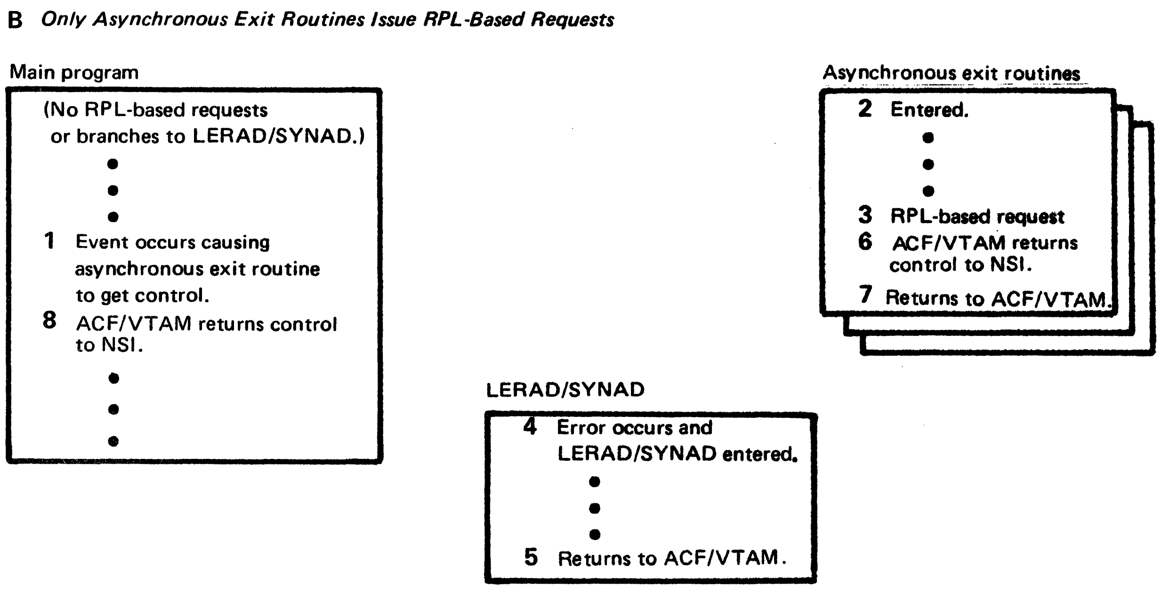

B Only Asynchronous Exit Routines Issue RPL-Based Requests
Main program
(No RPL-based requests or branches to LERAD/SYNAD.)
1 Event occurs causing asynchronous exit routine to get control.
8 ACF/VTAM returns control to NSI.
Asynchronous exit routines
2 Entered.
3 RPL-based request
6 ACF/VTAM returns control to NSI.
7 Returns to ACF/VTAM.
LERAD/SYNAD
4 Error occurs and LERAD/SYNAD entered.
5 Returns to ACF/VTAM.

Figure 7-12. Situations in Which LERAD and SYNAD Exit Routines Do not have to be Reenterable

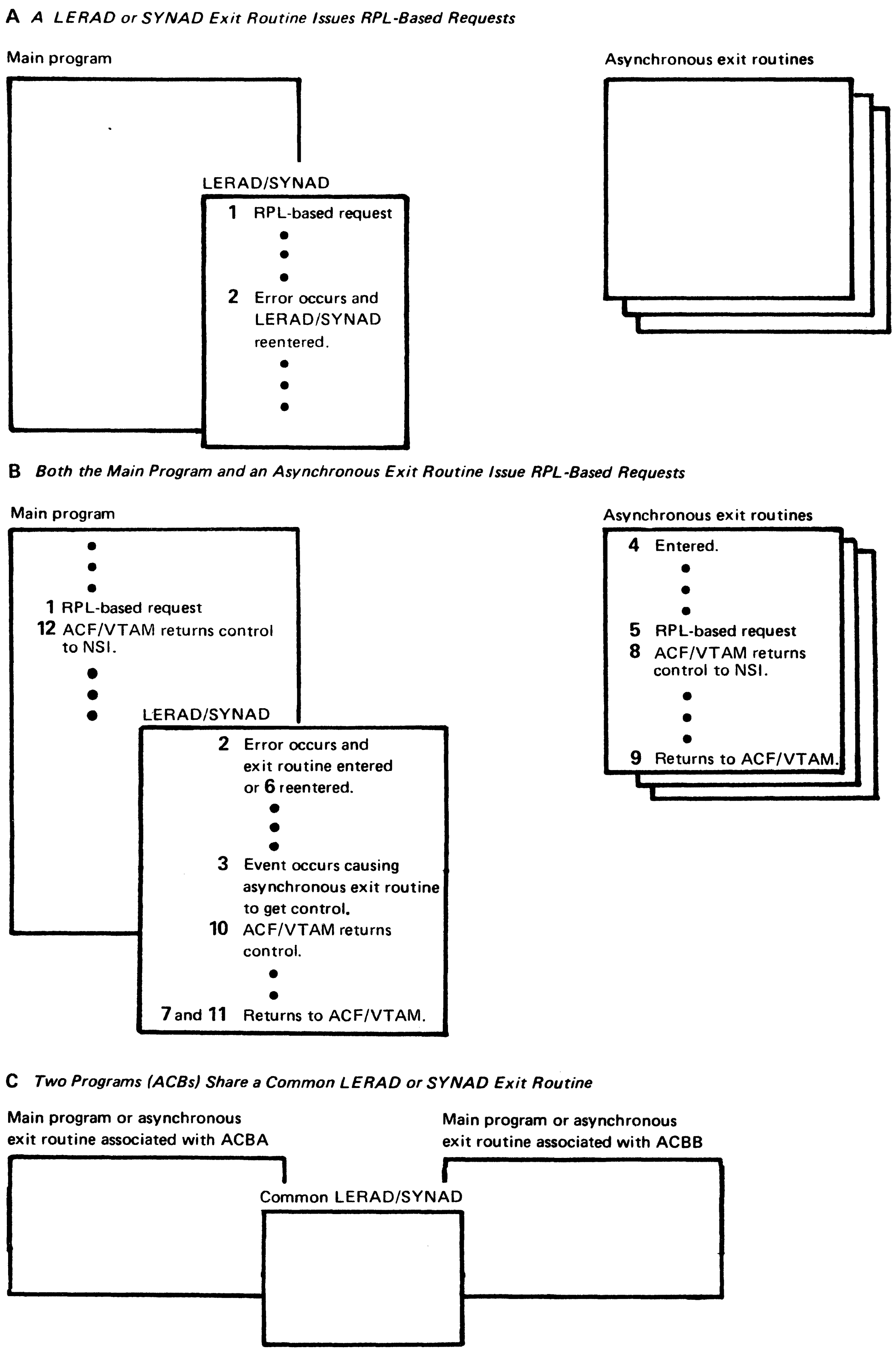

A A LERAD or SYNAD Exit Routine Issues RPL-Based Requests
Main program
Asynchronous exit routines
LERAD/SYNAD
1 RPL-based request
2 Error occurs and LERAD/SYNAD reentered.
B Both the Main Program and an Asynchronous Exit Routine Issue RPL-Based Requests
Main program
1 RPL-based request
12 ACF/VTAM returns control to NSI.
LERAD/SYNAD
2 Error occurs and exit routine entered or 6 reentered.
3 Event occurs causing asynchronous exit routine to get control.
10 ACF/VTAM returns control.
7 and 11 Returns to ACF/VTAM.
Asynchronous exit routines
4 Entered.
5 RPL-based request
8 ACF/VTAM returns control to NSI.
9 Returns to ACF/VTAM.
C Two Programs (ACBs) Share a Common LERAD or SYNAD Exit Routine
Main program or asynchronous exit routine associated with ACBA
Main program or asynchronous exit routine associated with ACBB
Common LERAD/SYNAD

Figure 7-13. Situation in Which LERAD and SYNAD Exit Routines Must be Reenterable

Chapter 8. Manipulating Control Blocks

The ACF/VTAM application program places values in the control blocks—ACB, EXLST, NIB, and RPL—that are used by ACF/VTAM when the application program requests ACF/VTAM to perform actions on its behalf. When the request is accepted or completed, ACF/VTAM places values in the ACB, NIB, or the RPL control block, as appropriate. The application program tests these values to determine the outcome of the request. This chapter discusses ways in which the ACF/VTAM application program can set and test these control block values.

Setting and Testing Control Block Values

Control block values can be set:

By defining them in an ACB, EXLST, NIB, or RPL macro instruction

By specifying values in operands of RPL-based macro instructions

By using the manipulative macro instructions GENCB and MODCB

By using the DSECT macro instructions and assembler instructions to move values into specified fields

By using the INQUIRE macro instruction with OPTCD=TERMS to generate a NIB or list of NIBs

Control block values can be tested:

By using the manipulative macro instructions TESTCB and SHOWCB

By using the DSECT macro instructions and assembler instructions to test values in specified fields

Using the Manipulative Macro Instructions

The macro instructions that manipulate application program control blocks are:

GENCB

MODCB

SHOWCB

TESTCB

The advantages of these macro instructions are that they:

Provide in one instruction what would require several assembler language instructions to provide

Allow symbolic references to be made to control blocks and their fields without having to be concerned with their relative storage locations

Can create control blocks in storage obtained dynamically, thereby allowing the application program to be reenterable

Make it possible to avoid reassembly should ACF/VTAM control blocks be changed in future releases

The GENCB Macro Instruction

GENCB builds and initializes a NIB, ACB, RPL, or EXLST. To use GENCB, this information is specified:

- The kind of control block to be built: ACB, NIB, RPL, or EXLST.
- The fields to be initialized and the values to be set in each field. For example, to build an RPL and initialize the OPTCD field to SYN, specify:

```
GENCB      BLK=RPL,AM=VTAM,OPTCD=SYN
```

- The number of copies of the control block to be built. Each copy is initialized with the same values. Each copy can later be modified as particular requests are made.
- Where the control block is to be built. The program defines an area where ACF/VTAM is to build the control block. If an area is not specified, ACF/VTAM gets the storage from the system dynamically. When storage is obtained dynamically, ACF/VTAM returns the address of the created control block in register 1 and the length in register 0.

Example 1: Build an ACB dynamically; initialize the EXLST field.

```
GENCB      BLK=ACB,AM=VTAM,EXLST=MYLST
```

When GENCB is completed, register 1 contains the address of the new ACB; register 0 contains its length.

Example 2: Build 50 copies of a NIB dynamically. Initialize the processing options field to DFASYX and RESPX so that expedited-flow messages and response input from the logical units connected with the NIBs will cause a DFASY or RESP exit routine to be scheduled.

```
GENCB      BLK=NIB,AM=VTAM,PROC=(DFASYX,RESPX),
           MODE=RECORD,COPIES=50
```

The program can calculate the length of each NIB by dividing the length in register 0 by the number of copies and use this value to refer to each NIB from an RPL.

Example 3: A storage management technique is used whereby, for each logical unit, the program obtains a storage area to contain an RPL, a logical unit work area, and a data area. It issues GETMAIN (or GETVIS in DOS/VS) to get storage for the entire area. Then, using a DSECT to map the area, it issues a GENCB to build an RPL. The WAREA operand of the GENCB macro uses an S-type constant to point to a real storage area using the DSECT as a map:

```
           GETMAIN    R,LV=LEN
           LR         2,1
           USING      TWA,2
           GENCB      BLK=RPL,AM=VTAM,WAREA=(S,RPL),
                      LENGTH=RPLLEN
           .
           .
           .
TWA        DSECT
CHAIN      DS         F              CHAINING POINTER
RPL        DS         0F
           IFGRPL     AM=VTAM,DSECT=YES
RPLEND     EQU        *
WORKAREA   DS         10F            LOGICAL UNIT WORK-AREA
DATA       DS         100C           DATA AREA
END        EQU        *
LEN        EQU        END-TWA        LENGTH OF ELEMENT
RPLLEN     EQU        RPLEND-RPL     LENGTH OF RPL
```

The SHOWCB macro can be used to examine the fields of this RPL:

```
          SHOWCB    AREA=SHOWAREA,RPL=(S,RPL),
                    FIELDS=FDBK2,LENGTH=4,AM=VTAM
            .
            .
            .
SHOWAREA  DS        F
```

If the program does not specify any fields to be initialized in the new control block, GENCB does one of the following:

Builds a blank (all zeros) ACB and assumes NLOGON for the MACRF operand

Builds an RPL using default values for all fields

Builds an EXLST with all entries flagged as inactive (no exit routines provided)

Builds a NIB using default values for all fields

The MODCB Macro Instruction

MODCB modifies the contents of an existing ACB, NIB, RPL, or EXLST. To use MODCB, this information is specified:

The access method (ACF/VTAM)

The kind of control block to be modified

The symbolic name of the control block or a register that contains the address of the control block

The fields to be modified

A common use of MODCB is to modify a NIB during execution of a LOGON exit routine. Here are some examples:

Example 1: A LOGON exit routine has been entered to put the symbolic name of the logical unit requesting connection into the NIB prior to connecting it. A pointer to the symbolic name of the logical unit is in the parameter list pointed to by register 1 when the exit routine is entered. Since the NIB NAME field must have the symbolic name itself and not its address, the programmer codes:

```
L         R4,4(R1)        POINT TO THE SYMBOLIC NAME
                          OF THE LOGICAL UNIT
MODCB     AM=VTAM,NIB=NIB1,NAME=(*,0(R4))        PUT IN NIB
```

Example 2: The entry for the LOGON exit routine in an exit list (labeled EX1) is to be changed to point to a routine named LOGON1.

```
MODCB     AM=VTAM,EXLST=EX1,LOGON=LOGON1
```

Example 3: A pool of 50 RPLs has been created using GENCB. The address of that pool is in register 6. Later, to modify the OPTCD field in the first RPL, this macro is issued:

```
MODCB     AM=VTAM,RPL=(6),OPTCD=SYN
```

To modify the second RPL in the same way, the program divides the number of copies (50) into the total length (contained in register 0) to obtain the length of one RPL. Assume that register 4 contains the length of one RPL.

```
AR        6,4             GET TO NEXT RPL
MODCB     AM=VTAM,RPL=(6),OPTCD=SYN
```

There are two restrictions governing the use of MODCB:

An open ACB or an RPL for a request that is being processed cannot be modified.

New entries cannot be added to an EXLST; only addresses of existing entries can be changed.

The SHOWCB Macro Instruction

SHOWCB copies the values of selected fields in an ACB, NIB, RPL, or EXLST into a designated area. In using SHOWCB, this information is specified:

The access method (ACF/VTAM).

The kind of control block: ACB, NIB, RPL, or EXLST.

The symbolic name of the particular control block or a register that contains the address of the control block.

The fields to be copied. For example, FDBK and FDBK2 fields in an RPL, the CID field in a NIB, or the ERROR field in an ACB can be specified. The fields must be in the same control block.

The name and length of a storage area in which ACF/VTAM will place the contents of the named fields. This area must begin on a fullword boundary.

Example 1: Extract the 4-byte CID from a NIB whose address is in register 2; put the CID in an area defined as CIDAREA.

```
SHOWCB     AM=VTAM,AREA=CIDAREA,NIB=(2),FIELDS=CID,
           LENGTH=4
```

Example 2: Extract the 4-byte address of the application program identification from the ACB labeled ACB1 and put it into an area labeled MYID.

```
SHOWCB     AM=VTAM,AREA=MYID,ACB=ACB1,FIELDS=APPLID,
           LENGTH=4
```

The application program identification can be used, for example, in an output message to a logical unit.

Example 3: Extract the contents of the FDBK and FDBK2 field from the RPL whose address is in register 7 and store the contents contiguously at HERE:

```
SHOWCB     AM=VTAM,AREA=HERE,RPL=(7),FIELDS=(FDBK,FDBK2),
           LENGTH=8
```

The TESTCB Macro Instruction

TESTCB tests the value of a specific field in an ACB, NIB, RPL, or EXLST. In using TESTCB, this information is specified:

The access method (ACF/VTAM)

The kind of control block: ACB, NIB, RPL, or EXLST

The name of the control block or a register that contains the address of the control block

The keyword for the field to be tested

The value against which the field is to be tested

Optionally, the name of a routine to be given control if ACF/VTAM cannot compare the two values

To test the results of a TESTCB, the TESTCB macro instruction can be followed with a branching instruction such as BE or BNE.

Some common uses of TESTCB are to test the ACB error flags when an ACB does not open properly and to test the FDBK field in the RPL after a data-transfer operation.

Example: If an ECB within the RPL is specified for posting, TESTCB can be used to determine which RPL has had its I/O request completed. Use TESTCB to test the IO field of each RPL:

```
LOOP     TESTCB   AM=VTAM,RPL=(8),IO=COMPLETE
         BE       OUT
*        INCREMENT REGISTER 8 TO NEXT RPL ADDRESS
         B        LOOP
```

Using the DSECT Macro Instructions and Assembler Instructions

ACF/VTAM provides macro instructions that generate a map of the fields and possible field values for each of the application program control blocks. Each macro instruction generates a DSECT instruction, a DS instruction for each field, and EQU instructions for certain predefined values. These macro instructions and associated assembler instructions can be used as an alternative to or in combination with the manipulative macro instructions. Whereas the manipulative macro instructions provide a more convenient way to set and test control block values, the DSECT macro instructions and assembler instructions that use the generated labels require the execution of fewer instructions.

Defining the DSECTs

Appendix H of *ACF/VTAM Macro Language Reference* shows the DSECT fields and equated values generated by the macro instructions. These are the DSECT macro instructions:

IFGACB for an ACB

IFGEXLST for an EXLST

ISTDNIB for an NIB

IFGRPL for an RPL

ISTDBIND for building a set of session parameters in an area in the application program or for checking a set of session parameters

In addition, one or both of these macro instructions can be used if ISTDNIB is not used:

ISTDPROC for the processing options field of the NIB

ISTDVCHR for the device characteristics field of the NIB

This field can be used if IFGRPL is not used:

ISTUSFBC for the FDBK2 field of an RPL

In coding these macro instructions, follow these rules:

Except for ISTDNIB, specify AM=VTAM as the sole operand.

Follow the macro instruction with a CSECT statement or another DSECT statement unless it is the last instruction before ending the program or unless the map is to be extended intentionally. (A terminal work area could be mapped following IFGRPL, for example.)

Ensure that no labels that are generated or reserved for future use are used elsewhere in the program. Do not use elsewhere in the program any label beginning with:

ACB	IFG	RPL
BIN	IST	RSV
DEV	NIB	USF
EXLST	PRO	

The DSECT produced by the ISTDBIND macro instruction is used to set up session parameters in an area in the application program or is used to examine a set of session parameters. For more information about the ISTDBIND DSECT, see Chapter 5 of this publication and Appendix J of *ACF/VTAM Macro Language Reference.*

Using the DSECTs

Having used the DSECT macro instructions to define one or more control block maps, the program can obtain storage for a control block from an assembled pool or dynamically from the system. The address of this storage should be placed in a register and specified in a USING statement before setting or testing values using the statements generated by the macro instruction. For example, suppose the address of an RPL is in register 5 and that, after a message containing data has been received, it is necessary to determine the value of the RECLEN field of the RPL. If an RPL-mapping macro instruction is coded:

```
IFGRPL          AM=VTAM
```

This statement:

```
USING           IFGRPL,5
```

allows assembler language instructions to refer to the label RPLRLEN to obtain the record length. (The labels are shown in Appendix H and Appendix J of *ACF/VTAM Macro Language Reference*.)

Using INQUIRE with OPTCD=TERMS to Generate NIBs

The INQUIRE macro instruction with OPTCD=TERMS can be used to build a single NIB or a list of NIBs. When the macro instruction is issued, the RPL must point to a NIB whose NAME field contains the name of the single logical unit or terminal or group of logical units or terminals for which the NIB or NIBs are to be created. The NAME field of the NIB can contain the symbolic name of a PU, LU, TERMINAL, LINE, CLUSTER, or GROUP definition statement. A NIB is built for each logical unit or terminal represented by the definition statement.

When the macro instruction is issued, the RPL's AREA and AREALEN fields must designate the location and length of the work area where the NIBs are to be built. Before the macro instruction is issued, the work area must be set to binary zeros.

If the application program wants the NIBs to be built in dynamically allocated storage (storage obtained by the application program during execution), the INQUIRE should be issued twice. For the first INQUIRE, set AREALEN to 0. This INQUIRE will be completed with RTNCD=0 and FDBK2=5 (insufficient length), and RECLEN will indicate the required length. Obtain the required storage and then issue INQUIRE again with AREALEN set to the proper length.

After the macro instruction is completed, the RPL's RECLEN field contains the total length (number of bytes) for all NIBs generated by the macro instruction. The NAME field of each NIB contains the symbolic name of the logical unit or terminal for which the NIB was generated, and each NIB contains the device characteristics for the logical unit or terminal it represents. The LISTEND indicator is YES in the last NIB generated; all terminal it represents. The LISTEND indicator is YES in the last NIB generated; all preceding generated NIBs contain LISTEND=NO. Using the symbolic names and device characteristics, the application program can set PROC options and the MODE field in each NIB, and it can set other fields to desired values. The NIBs are then ready to be used for connection. For an example of using INQUIRE with OPTCD=TERMS, see "Acquiring Logical Units" in Chapter 5.

Chapter 9. Handling Errors and Special Conditions

The ACF/VTAM application program requests that ACF/VTAM perform an operation, passing control to ACF/VTAM. ACF/VTAM returns control to the program with information about the operation. For RPL-based requests, depending on how the request was specified, ACF/VTAM tells the program (1) whether the request is accepted and the operation is underway or (2) how the operation was completed. If the operation was completed successfully, ACF/VTAM provides information such as the identity of the logical unit and the length of a message received. This kind of information for successful completion is discussed throughout this book.

This chapter discusses how to analyze information for errors and special conditions and what to do, in general, when the error or special condition is identified. Identifying and acting upon errors and special conditions are discussed separately for:

OPEN/CLOSE macro instructions

Manipulative macro instructions

RPL-based requests (such as SEND, RECEIVE, and CHECK)

ACF/VTAM software errors

For RPL-based requests, errors and special conditions are discussed separately in these categories:

Exception conditions (including exception messages and negative responses with related sense information)

Retriable completions

Data-integrity-damage completions

Environment errors

Logical errors

The information that ACF/VTAM returns to the application program is organized so that only a minimum amount of checking need be done. For most macro instructions, if register 15 contains 0 on return to the program, no further checking need be done; the program proceeds normally. If register 15 contains a nonzero value, checking proceeds until the condition is defined and appropriate action is taken. For certain RPL-based macro instructions, register 0 must be examined for a special condition even if register 15 contains 0. Since fewer errors and special conditions can occur as the result of an OPEN/CLOSE or manipulative macro instruction (GENCB, MODCB, SHOWCB, or TESTCB) than for RPL-based macro instructions, the coding needed to handle the OPEN/CLOSE and manipulative errors is simpler to write.

OPEN/CLOSE Errors and Special Conditions

Before issuing an OPEN or CLOSE macro instruction, DOS/VS users should clear register 15. For DOS/VS and OS/VS, after the OPEN or CLOSE, register 15 should be tested. If the return code in register 15 is 0, all ACBs have been opened or closed as requested. If the return code does not equal 0, one or more ACBs were not properly opened or closed. When this occurs, the TESTCB macro can be used to test the OFLAGS field of each ACB to see if it is open:

```
TESTCB   AM=VTAM,ACB=(3),OFLAGS=OPEN
```

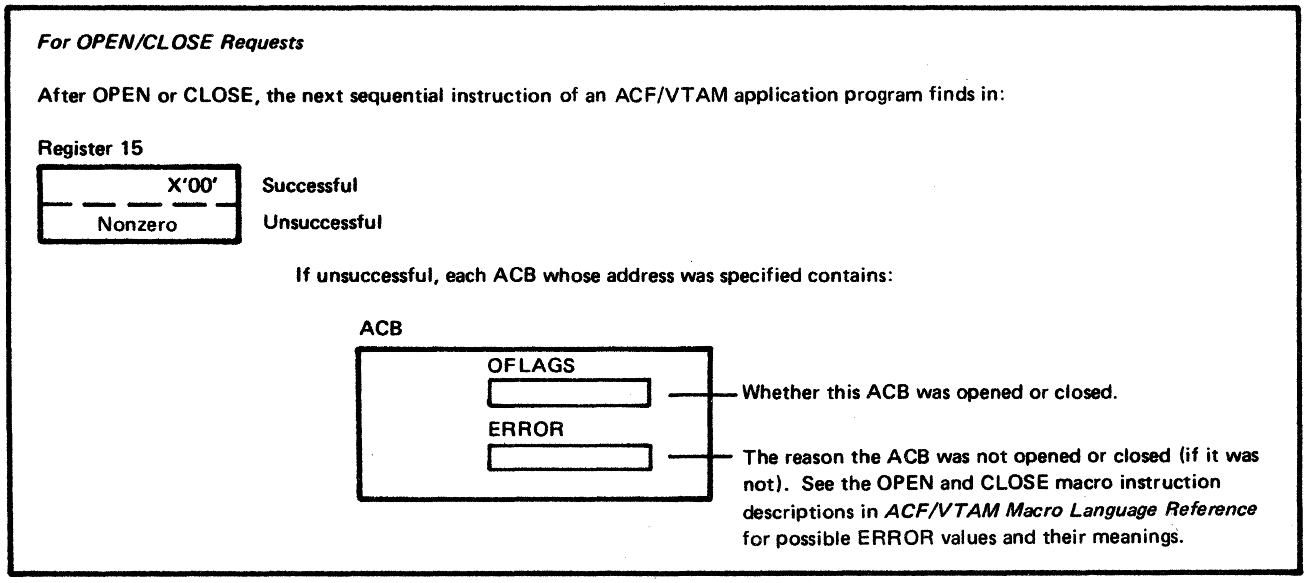
The address of an ACB is in register 3. If an OPEN macro failed, the failing ACB will not have OFLAGS=OPEN, and the ACB will still be closed. If CLOSE failed, the failing ACB will have OFLAGS=OPEN because the ACB was not closed. When the ACB is found, SHOWCB macro can be used to look at the error bits in the ERROR field of the ACB. *ACF/VTAM Macro Language Reference* describes the format and content of the ERROR field in its description of the OPEN and CLOSE macro instructions. Figure 9-1 shows how OPEN/CLOSE error and special condition information is organized.

Most ERROR settings shown in *ACF/VTAM Macro Language Reference* indicate an error in program logic or some failure to match the name of an ACB as specified in the program with its name as specified during ACF/VTAM definition. A dump, program termination, and debugging is required. If multiple ACBs are being opened, and only some have been opened successfully, it may be possible to continue with the programs whose ACBs were opened.

Manipulative Macro Instruction Errors and Special Conditions

After issuing a GENCB, MODCB, SHOWCB, or TESTCB macro instruction, the return code in register 15 must be tested. If the return code is 0, the manipulative operation was successful. If register 15 contains hex 04 or hex 08, it was not successful. Hex 04 in register 15 is an indication to look for the specific error in register 0. (Thus, examination of a return code for a SHOWCB or TESTCB macro does not require examination of a control block field.) Hex 08 in register 15 indicates that an attempt has been made to use the execute form of the macro to enter a *new* item in the parameter list being modified (only existing items can be modified). In DOS/VS, hex 0C in register 15 has special meaning. Appendix D in *ACF/VTAM Macro Language Reference* shows the possible register 0 settings and their meanings. Figure 9-2 shows in general how manipulative macro instruction error and special condition information is organized.

In all cases except one, manipulative macro instruction errors and special conditions are due to faulty logic and require program termination and debugging; they should not occur once the program has been debugged. In one case, however (register 0 contains hex 08), GENCB can fail because of insufficient storage; the request can be reissued at a later time.


For OPEN/CLOSE Requests

After OPEN or CLOSE, the next sequential instruction of an ACF/VTAM application program finds in:

If unsuccessful, each ACB whose address was specified contains:

Whether this ACB was opened or closed.

The reason the ACB was not opened or closed (if it was not). See the OPEN and CLOSE macro instruction descriptions in *ACF/VTAM Macro Language Reference* for possible ERROR values and their meanings.

Figure 9-1. How OPEN/CLOSE Error and Special-Condition Information Is Organized

For Manipulative Macro Instruction Requests

After a GENCB, MODCB, SHOWCB, or TESTCB, the next sequential instruction* finds in:

Register 15

X'00'	Successful
X'04'	Error. See register 0.
X'08'	Error. No code in register 0. See Appendix D in *ACF/VTAM Macro Language Reference* for the meaning of X'08'.
X'0C'	Error. DOS/VS system control error. See register 0. See Appendix D in *ACF/VTAM Macro Language Reference* for the meaning of X'0C'.

If unsuccessful with X'04' in register 15 or with X'0C' in register 15 in a DOS/VS system, register 0 contains:

Register 0

Error Return Code	See Appendix D in *ACF/VTAM Macro Language Reference.* for possible return codes and their meanings.

*Alternately, for TESTCB, if an error occurs, control may be passed to the specified address of an ERET (error return code) routine.

Figure 9-2. How Manipulative Macro Instruction Error and Special-Condition Information Is Organized

RPL-Based Macro Instruction Errors and Special Conditions

There are two kinds of RPL-based operations: synchronous and asynchronous. For synchronous RPL-based operations, a single macro instruction is issued. On return to the ACF/VTAM application program, error or special-condition information is available about the requested operation.

For asynchronous RPL-based operations, two RPL-based macro instructions are required: a request macro instruction and a CHECK (completion) macro instruction. Error and special-condition information can thus be returned at two different stages, as a result of the *request* being accepted or not accepted and, if the request is accepted, as a result of the operation completing successfully or unsuccessfully.

Following an RPL-based macro instruction, information is available to the program about the acceptability of the request or about the completion of the operation. This information may be provided by ACF/VTAM or, if an error or special condition was detected and ACF/VTAM scheduled the program's LERAD or SYNAD exit routine, from the LERAD or SYNAD exit routine. The information consists of a return code in register 15, in some cases a return code in register 0, and information in the RPL. Figures 9-3, 9-4, and 9-5 show how this information is organized.

Reg 15	Meaning	Reg 0
X'00'	Normal or conditional completion, or acceptance.	0 or conditional completion return code
X'04'	Request not accepted or completed abnormally. No exit found.	RPL recovery action return code
X'20'	ACB not open.	RPL request code

Figure 9-3. Register 15 and Register 0 Return Codes Following an RPL-Based Request

For RPL-Based Requests

After a SEND, RECEIVE, CHECK, or other RPL-based request, the next sequential instruction finds in:

Register 15

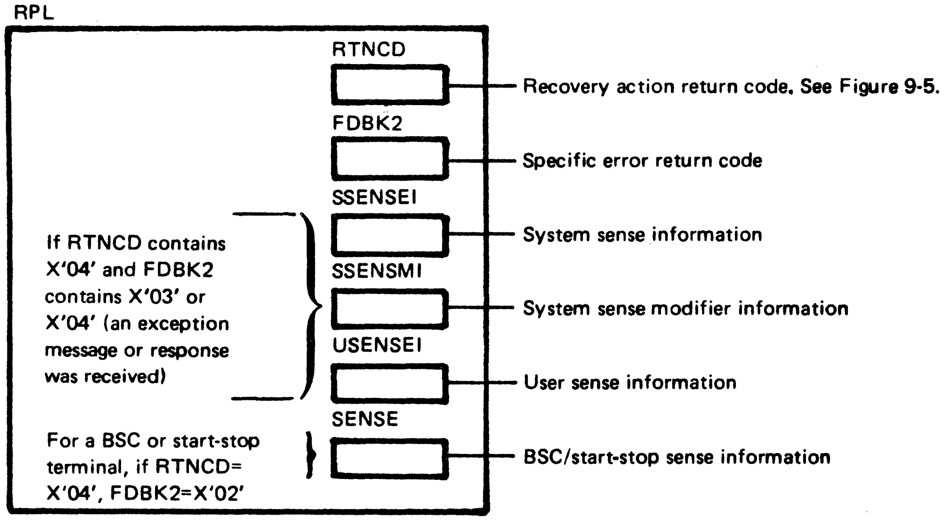
X'00'	The request or, for a synchronous request (including CHECK), the operation was successful.
Some other value	The request or the operation was not successful.

Depending on the request and whether or not it was successful,* it may be necessary to test:

Register 0

If register 15 is X'00', register 0 indicates for certain macros whether success was conditional. If register 15 is not X'00', register 0 can contain a return code from a LERAD or SYNAD exit routine or, if there is no LERAD/SYNAD, a recovery action return code (generally the code from the RTNCD field of the RPL).

If a request or operation was unsuccessful or conditionally successful, these RPL fields can be examined (in either the issuing routine or in a LERAD or SYNAD exit routine):



RPL
RTNCD
Recovery action return code. See Figure 9-5.
FDBK2
Specific error return code
SSENSEI
System sense information
SSENSMI
System sense modifier information
USENSEI
User sense information
If RTNCD contains X'04' and FDBK2 contains X'03' or X'04' (an exception message or response was received)
SENSE
For a BSC or start-stop terminal, if RTNCD= X'04', FDBK2=X'02'
BSC/start-stop sense information

In addition, other RPL fields that contain feedback information normally used following completion of a requested operation, such as SEQNO, CHAIN, and CHNGDIR, may be used in determining how to handle an error or special condition.

RPL fields are described under RPL and other macro instruction descriptions and summarized in Appendix A of *ACF/VTAM Macro Language Reference.*

Possible RPL RTNCD, FDBK2, and sense information settings and their meanings are in Appendix C of *ACF/VTAM Macro Language Reference.*

For Arrival of a Logical Unit Status command

After receiving input with a RECEIVE specifying RTYPE=DFSYN, if CONTROL=LUS, one or more of the RPL sense fields (SSENSEI, SSENSMI, and USENSEI) will contain error or special-condition information from the logical unit.

*Register 0 is of interest in these circumstances:

- For certain macros with certain options set (DO, INQUIRE, INTRPRET, OPNDST, READ, RECEIVE with OPTCD=NQ, RESET, and WRITE), if register 15 contains X'00', success may be conditional. Register 0 should be examined to see if there is a condition (and what it is).
- If an error or special condition occurred for an RPL-based request and no LERAD or SYNAD exit routine is available, register 15 contains X'04', and register 0 contains a recovery action return code.
- If a LERAD or SYNAD exit routine is available, register 15 can be set to X'00' to indicate "Error corrected—request or operation successful." If not corrected, register 15 should be nonzero and a return code can be passed from LERAD/SYNAD in register 0.

Figure 9-4. How RPL-Based Macro Instruction Error and Special-Condition Information Is Organized

Recovery Action Return Code (In RPL RTNCD Field)	LERAD or SYNAD Exit Scheduled	Type of Completion	Programmer Action
(Hex) 00		Normal or conditional	Program for at NSI.
04	SYNAD	Exception condition	Analyze RPL to choose logic path.
08	SYNAD	Retriable completion	Use EXECRPL macro to retry if desired.
0C	SYNAD	Data integrity damage	Execute user program error recovery coding.
10	SYNAD	Environment error	Call for external intervention.
14	LERAD	User logic error	Dump program status and continue or abend.
18 (in register 0 but not in RTNCD field)	LERAD	User logic error, no RPL feedback	Dump program status and continue or abend. Do not reuse this RPL.
Others	LERAD or SYNAD	RPL overwritten	Same as above.

Figure 9-5. Recovery Action Return Codes and Their General Meanings

If an error or special condition occurred, it can be analyzed and handled in either or both of two places:

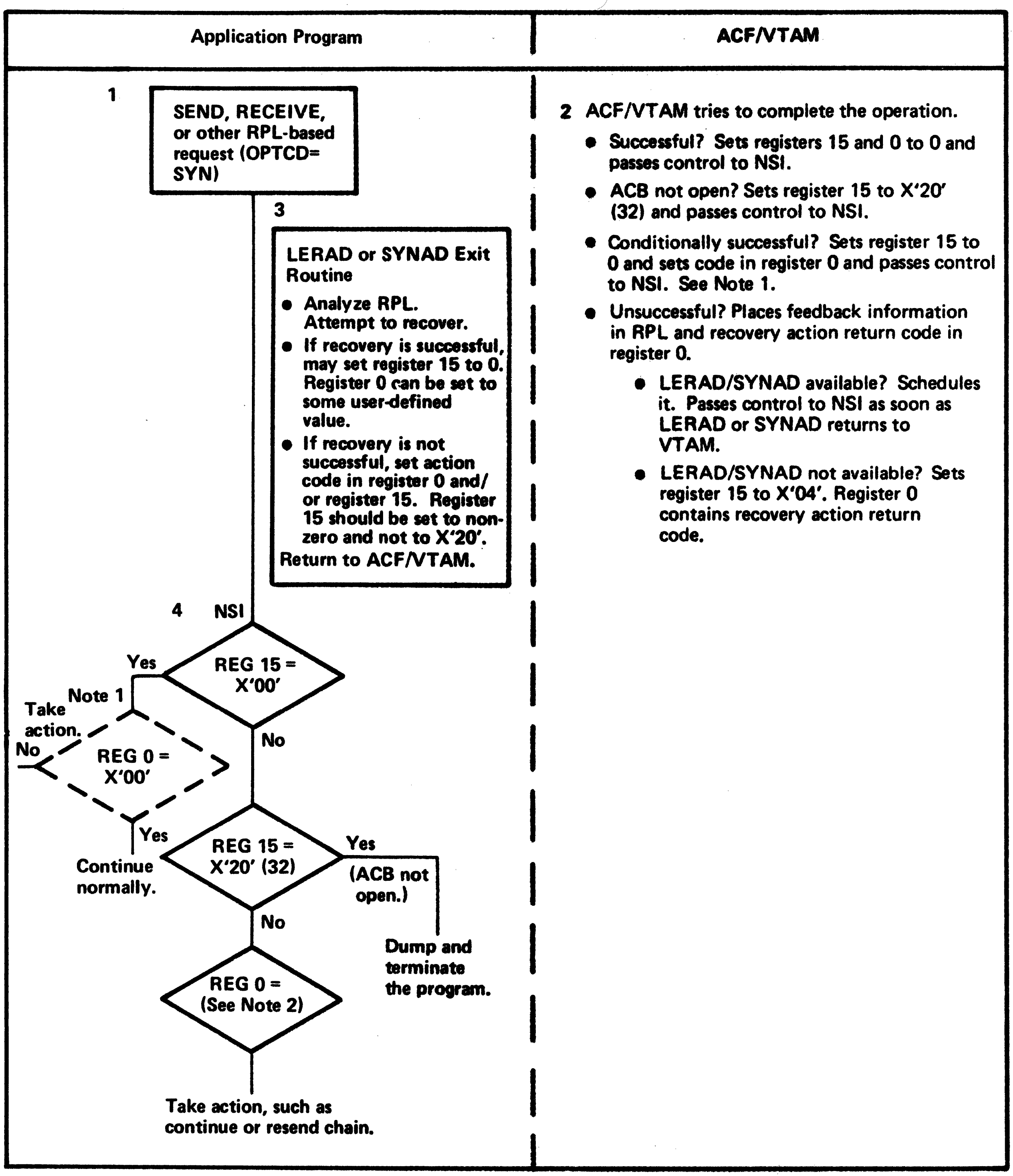
In the main program or exit routine where the RPL-based request was issued

In a designated LERAD or SYNAD exit routine

It is convenient to use LERAD and SYNAD exit routines to handle error and special conditions for all RPL-based requests in a program. A LERAD or SYNAD exit routine can be entered for an asynchronous operation at two different times: after the request fails and, if the request is accepted, after the operation fails. The LERAD or SYNAD exit routine can set register 15 to 0 so that the request-issuing part of the program is not aware that an error or special condition occurred and will continue normally. If the request-issuing part of the program must be made aware that an error or special condition occurred, register 15 can be set to a nonzero value and a user-specified return code placed in register 0 (in this case, register 15 can also be used as a return code register between the LERAD or SYNAD exit routine and the issuing part of the program).

If a LERAD or SYNAD exit routine is not used, errors and special conditions can be handled in the inline coding that follows the request or can be processed in a common subroutine. The same information available to the LERAD/SYNAD exit routine is available to the requesting part of the program, and the discussion of this information in "Coding LERAD or SYNAD Exit Routines" applies (apart from the discussion of passing return codes back to the requesting part of the program).

Figures 9-6 and 9-7 show the relationship between the requesting part of the program and LERAD and SYNAD exit routines if present. These figures can also be used to code the instructions that check the results of requests in the requesting parts of the program.



1 SEND, RECEIVE, or other RPL-based request (OPTCD= SYN)

2 ACF/VTAM tries to complete the operation.

- Successful? Sets registers 15 and 0 to 0 and passes control to NSI.
- ACB not open? Sets register 15 to X'20' (32) and passes control to NSI.
- Conditionally successful? Sets register 15 to 0 and sets code in register 0 and passes control to NSI. See Note 1.
- Unsuccessful? Places feedback information in RPL and recovery action return code in register 0.
  - LERAD/SYNAD available? Schedules it. Passes control to NSI as soon as LERAD or SYNAD returns to VTAM.
  - LERAD/SYNAD not available? Sets register 15 to X'04'. Register 0 contains recovery action return code.

3 LERAD or SYNAD Exit Routine

- Analyze RPL. Attempt to recover.
- If recovery is successful, may set register 15 to 0. Register 0 can be set to some user-defined value.
- If recovery is not successful, set action code in register 0 and/ or register 15. Register 15 should be set to non-zero and not to X'20'.

Return to ACF/VTAM.

Figure 9-6 (Part 1 of 2). A Summary of Error and Special-Condition Handling with Synchronous Operations

Note 1 If a DO, INQUIRE, INTRPRET, OPNDST, READ, RECEIVE specifying NQ, or WRITE is issued, register 0 may be set to indicate some condition that may or may not require further action, such as getting a larger input area for a logon message and reissuing INQUIRE. For other requests, register 0 need not be checked.

Note 2 Register 0 can contain:

- A value set by the LERAD or SYNAD exit routine. This can indicate one of several actions to take, such as reissuing the request or setting up the resending of a chain. Register 15 can also be used to pass information.
- If the issuer is handling errors inline (no LERAD or SYNAD is available), the RTNCD value from the RPL. The value is one of these:

X'04' – Exception condition X'08' – Retriable completion X'0C' – Data integrity damaged X'10' – Environment error X'14' – Logical error	See Appendix C of *ACF/VTAM Macro Language Reference* for a discussion of these recovery action return codes.

The FDBK2 field of the RPL will contain a specific error code.

Or register 0 can contain:

X'18' – Logical error: Invalid RPL; or an operation was attempted on an RPL already in use; or a CHECK macro instruction has been used for a request whose RPL exit routine has not yet been scheduled.

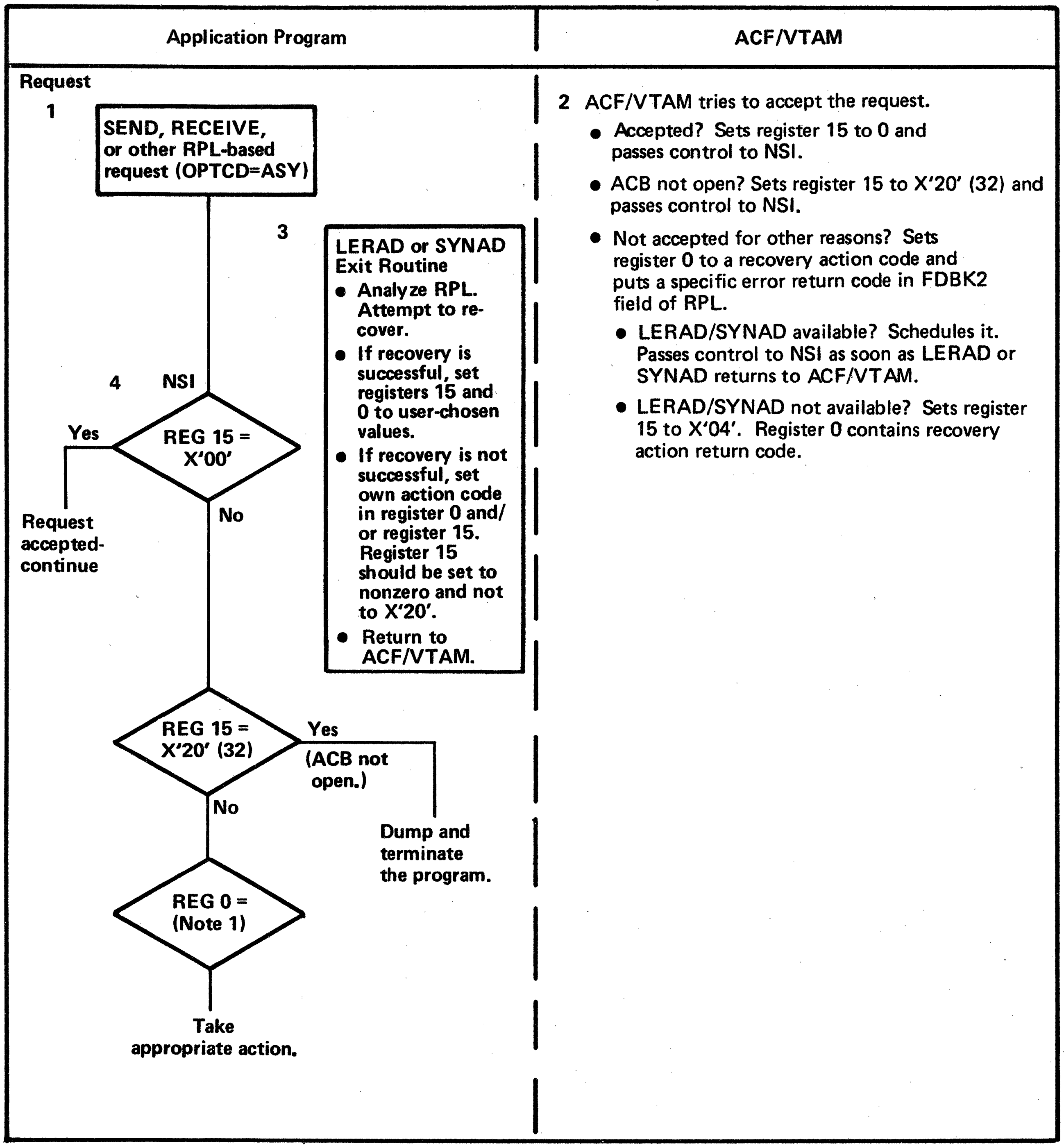
Figure 9-6 (Part 2 of 2). A Summary of Error and Special-Condition Handling with Synchronous Operations

If an output operation (SEND macro instruction) that specifies *scheduling* of message output is requested, error information can be viewed as returning at two different times if the request is for a synchronous operation: first, when the scheduling of the requested operation is completed and secondly, if a response is requested, when a RECEIVE that specifies RTYPE=RESP is completed or a RESP exit routine is scheduled, with the response that has arrived confirming that the scheduled message was received. (The RECEIVE specifying RTYPE=RESP can be seen as an operation by itself.) Error information can be viewed as occurring at three different times if the request is for an asynchronous operation: first, when the *request* to schedule the operation is accepted; second, when the scheduling of the operation is completed; and third, when a response confirms arrival of the message. Figure 9-8 can be used to understand this sequence of events.

Coding LERAD and SYNAD Exit Routines

LERAD and SYNAD exit routines can have a common entry point; the logic at the common entry point can determine whether the error is a logical error or a physical error and branch to the appropriate error-handling instructions.

Figure 9-9 shows the use of registers on entering and leaving a LERAD/SYNAD exit routine. Addressability and save area requirements are described in Chapter 7. Examples of LERAD and SYNAD exit routines are shown in the sample ACF/VTAM application program in Appendix D.



LERAD or SYNAD Exit Routine

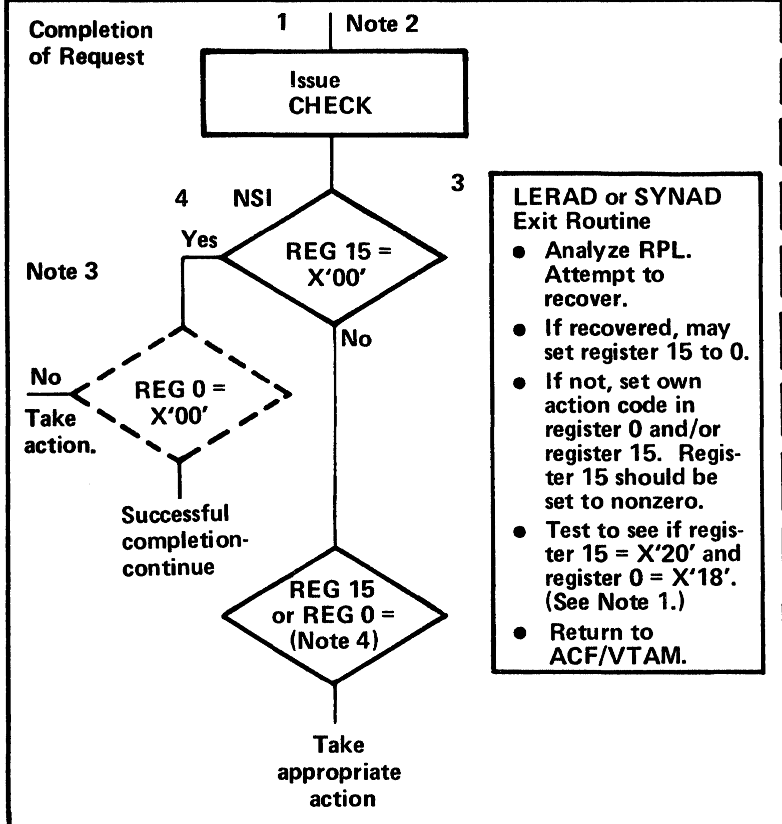
- **Analyze RPL. Attempt to recover.**
- **If recovery is successful, set registers 15 and 0 to user-chosen values.**
- **If recovery is not successful, set own action code in register 0 and/or register 15. Register 15 should be set to nonzero and not to X'20'.**
- **Return to ACF/VTAM.**

2 ACF/VTAM tries to accept the request.

- Accepted? Sets register 15 to 0 and passes control to NSI.
- ACB not open? Sets register 15 to X'20' (32) and passes control to NSI.
- Not accepted for other reasons? Sets register 0 to a recovery action code and puts a specific error return code in FDBK2 field of RPL.
  - LERAD/SYNAD available? Schedules it. Passes control to NSI as soon as LERAD or SYNAD returns to ACF/VTAM.
  - LERAD/SYNAD not available? Sets register 15 to X'04'. Register 0 contains recovery action return code.

Figure 9-7 (Part 1 of 2). A Summary of Error and Special-Condition Handling with Asynchronous Operations

Application Program	ACF/VTAM



LERAD or SYNAD Exit Routine

- **Analyze RPL. Attempt to recover.**
- **If recovered, may set register 15 to 0.**
- **If not, set own action code in register 0 and/or register 15. Register 15 should be set to nonzero.**
- **Test to see if register 15 = X'20' and register 0 = X'18'. (See Note 1.)**
- **Return to ACF/VTAM.**

2
- After the operation is completed, ACF/VTAM posts an ECB or schedules an RPL exit routine. After CHECK is issued, if the operation was successful, ACF/VTAM sets registers 15 and 0 to 0 and passes control to NSI.
- If the operation was conditionally successful, register 15 is set to 0 and register 0 is set to a special code (see note 4). ACF/VTAM passes control to NSI.
- If the request or the operation was unsuccessful, ACF/VTAM puts the recovery action return code (RTNCD) in register 0 and feedback information in the RPL.
  - LERAD or SYNAD available? Schedule it. Pass control back to NSI when exit-routine returns to ACF/VTAM.
  - LERAD or SYNAD not available? Set register 15 to X'04'. Register 0 contains the recovery action return code.

Note 1 Register 0 can contain:

- A value set by the LERAD or SYNAD exit routine. This can indicate one of several actions to take, such as reissuing the request. Register 15 can also be used to pass information.
- If no LERAD or SYNAD is available, the RTNCD value from the RPL. The value is one of these:

X'04' – Exception condition
X'08' – Retriable completion
X'0C' – Data integrity damaged
X'10' – Environment error
X'14' – Logical error

See Appendix C of *ACF/VTAM Macro Language Reference.*

Or register 0 can contain:

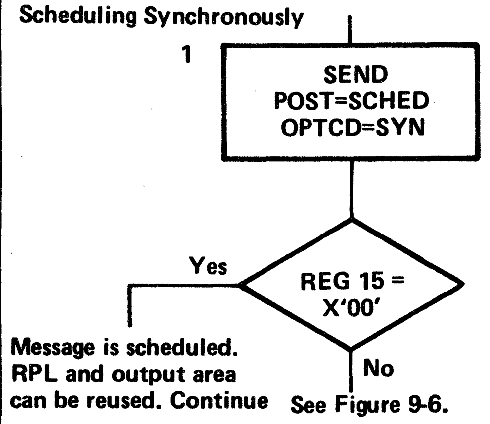
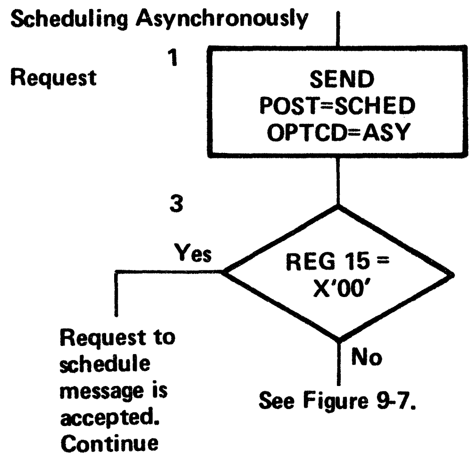
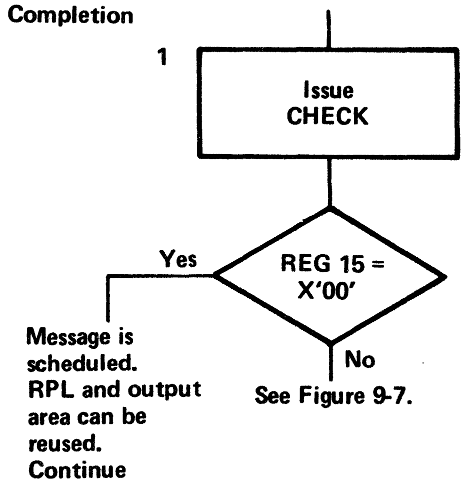
X'18' – Logical error: Invalid RPL; or an operation was attempted on an RPL already in use; or a CHECK macro instruction has been issued for a request whose RPL exit routine has not yet been scheduled.

Note 2 The CHECK macro can be issued following discovery of a posted ECB, or to wait for it to be posted, or after scheduling of an RPL exit routine. The description here applies whether ECB-posting or RPL exit-routine scheduling is used. Rather than issue a CHECK immediately, the program can look at the RPL feedback fields. CHECK is required sooner or later, however, to free the RPL for reuse.

Note 3 If a DO, INQUIRE, INTRPRET, OPNDST, READ, RECEIVE (specifying NQ), or WRITE is issued, register 0 may be set to indicate some condition that may or may not require further action, such as getting a larger input area for a logon message and reissuing INQUIRE. For other operations, register 0 need not be checked.

Note 4 Register 0 and, if necessary, register 15 can be used to convey information from the LERAD or SYNAD exit routine about what to do next. These routines can reduce the alternatives that follow the CHECK to two or three things, such as sending a special message, disconnecting the logical unit, or terminating the program.

Figure 9-7 (Part 2 of 2). A Summary of Error and Special-Condition Handling with Asynchronous Operations

Application Program	ACF/VTAM
Scheduling Synchronously  (See Note 1 on how to know whether the message arrived successfully.)	2 ACF/VTAM tries to complete this operation (that is, schedule the message for sending, moving the message to a ACF/VTAM buffer). • Completed? Sets registers 15 and 0 to 0 and passes control to the next sequential instruction (NSI). The operation is completed. • Not completed? See Figure 9-6.
Scheduling Asynchronously 	2 ACF/VTAM tries to accept the request. • Accepted? Sets registers 15 and 0 to 0 and passes control to NSI. • Not accepted? See Figure 9-7.
Completion 	2 When the operation is completed (that is, the scheduling of sending the message), ACF/VTAM posts an ECB or schedules an RPL exit routine. After CHECK is issued, if the operation was successful, registers 15 and 0 are set to 0. If unsuccessful, ACF/VTAM puts feedback information in the RPL. See Figure 9-7.

1
SEND POST=SCHED OPTCD=SYN
REG 15 = X'00'
Yes
Message is scheduled. RPL and output area can be reused. Continue
No
See Figure 9-6.

Request
1
SEND POST=SCHED OPTCD=ASY
3
REG 15 = X'00'
Yes
Request to schedule message is accepted. Continue
No
See Figure 9-7.

1
Issue CHECK
REG 15 = X'00'
Yes
Message is scheduled. RPL and output area can be reused. Continue
No
See Figure 9-7.

(See Note 1 on how to know whether the message arrived successfully.)

Figure 9-8 (Part 1 of 2). A Summary of Error and Special-Condition Handling with Scheduling of Messages

Note 1 After scheduling a message successfully, the program can determine its arrival by:

1. Receiving a positive response to it (requested by specifying RESPOND=NEX, FME or NEX,RRN). This response is received either by completion of a RECEIVE with RTYPE=RESP or by the program's gaining control at its RESP exit routine. The SEQNO field of the RPL associated with the response will match the sequence number of the message being replied to. If a RESP exit routine is entered, it can post an ECB to tell the main program that the message arrived.
2. Receiving a positive response to the last message in a chain (or to a message that is the only one in a chain) indicates that all messages in the chain were received.
3. Receiving a subsequent message from the logical unit that indicates by its content that the message was successfully received. (This might mean that if it had *not* been successfully received, the content of a subsequent message would so indicate.)

Figure 9-8 (Part 2 of 2). A Summary of Error and Special-Condition Handling with Scheduling of Messages

On Entering A LERAD or SYNAD Exit Routine	
Register	Contains
0	The recovery action return code (RTNCD field of RPL)
1	The address of the RPL
2-13	The contents as they were when RPL-based request was issued
14	The address at which control may be returned to ACF/VTAM
15	The address of the LERAD or SYNAD exit-routine
On Leaving a LERAD or SYNAD Exit Routine	
Register	Contains
0	The user return code from LERAD or SYNAD
1	The address of the RPL (although ACF/VTAM will restore it)
2-12	Any value (ACF/VTAM will restore the contents as they were when the request was issued)
13	The address of the save area from the request-issuing part of the program
14	The address to which control is being passed
15	An optional user return code (in addition to register 0)

Figure 9-9. A Summary of Register Usage on Entering and Leaving a LERAD or SYNAD Exit Routine

The recovery action return code (in register 0 and in the RTNCD field of the RPL unless it is unusable) can be used to branch to separate subroutines in the LERAD/SYNAD exit routine. This might be coded:

```
         LR   R2,R0      SINCE R0 NOT USABLE AS INDEX REG
         B    *(R2)      BRANCH TO APPROPRIATE BRANCHING
*                        INSTRUCTION
         B    EXCEPTN    EXCEPTION CONDITION (R2='04')
         B    REISSUE    RETRIABLE COMPLETION (R2='08')
         B    DINTDAM    DATA INTEGRITY DAMAGE (R2='0C')
         B    ENVERR     ENVIRONMENT ERROR (R2='10')
         B    USLOGIC    USER LOGIC ERROR (R2='14')
         B    NOTINRPL   USER LOGIC ERROR BUT INFORMATION
*                        IS NOT IN THE RPL (R2='18')
```

The logic that these subroutines, each representing a general category of recovery action, might contain is discussed below.

Handling Exception Conditions (R0=X'04')

This section covers:

Handling exception messages

Handling negative responses

Handling certain exceptional conditions for BSC and start-stop terminals

Handling Exception Messages

If a message arrives from the other end of the session for which ACF/VTAM detects an error (for example, the message arrived with a sequence number that was not one greater than the previous sequence number), ACF/VTAM puts hex 04 in register 0, hex 04 in the recovery action return code field (RTNCD) of the RPL, and hex 03 in the specific error return code field (FDBK2) of the RPL. It also puts values in the SSENSEI, SSENSMI, and USENSEI fields of the RPL. Unless the other end of the session does not want a response (the RESPOND field contains NFME,NRRN), the ACF/VTAM application program either in this part of the SYNAD exit routine or perhaps later in some other part of the program, must send a response. An exception to this rule is this: if a chain of messages is being received, and a negative response has already been returned to a message that was received as part of the chain, and remaining elements of the chain are being received as exception messages, only the first negative response should be sent and the remaining elements should be disregarded.

Before sending the negative response, the values in the SSENSEI and SSENSMI fields must be transferred to the SSENSEO and SSENSMO fields of the RPL. (SHOWCB cannot be used to make this transfer; a combination of TESTCBs, to determine the values, and MODCB, to set the determined values in SSENSEO and SSENSMO, must be used.)

Further Action by a Primary Application Program: In addition to transferring these values for sending response information, the SYNAD exit routine in a primary application program may want to analyze these fields to determine what to do next. It may want to:

Begin sequence number resynchronization, using the SESSIONC macro instruction.

Await a Request Recovery (RQR) command from the logical unit (which would cause scheduling of the program's SCIP exit routine) and then begin sequence number resynchronization. In this case, the logical unit must be capable of deciding whether to shut down or to request recovery.

If in the process of receiving a chain of messages, set a flag to purge the buffer that contains messages in this chain that have been previously received. It can set a flag to indicate to itself that the rest of the messages in this chain are to be disregarded. When the last message arrives, the program can turn off this flag.

The sense information return codes and their meanings are discussed in Appendix C of *ACF/VTAM Macro Language Reference.*

Further Action by a Secondary Application Program: When a secondary application program learns of a sequence number error in an incoming message, it too transfers the values in the SSENSI and SSENSMI fields to the SSENSEO and SSENSMO fields of the RPL before sending the negative response with that RPL. In addition, the SYNAD exit routine in a secondary application program may want to:

Send a Request Shutdown (RSHUTD) command to the primary application program if sequence number resynchonization is not possible.

Send a Request Recovery (RQR) command to ask the primary application program to begin sequence number resynchronization. In this case, the secondary application program will await a Clear command followed by a Set and Test Sequence Numbers (STSN) command from the primary program.

If in the process of receiving a chain of messages, set a flag to purge the buffer that contains messages in this chain that have been previously received. It can set a flag to indicate to itself that the rest of the messages in this chain are to be discarded. When the last message arrives, the program can turn off this flag.

The exact action to be taken by each end of the session in the event of a sequence number error must be agreed upon by the programmers coding the primary and secondary application programs before they are coded.

Further information on sequence number resynchronization action can be found under "Controlling Flow" in Chapter 6 and in Figures C-3, C-9, and C-18 in Appendix C.

Handling Negative Responses

This logic can be entered as the result of one of the following conditions:

A pending RECEIVE that specifies RTYPE=RESP is completed. The program determines that a negative response was received and branches to the negative response handling routine or reaches it as a result of issuing a CHECK macro.

The program's RESP exit routine is entered as the result of a response being received. A CHECK is issued against the user-specified RPL (not the read-only RPL furnished to the exit routine by ACF/VTAM), or a direct branch is made to the negative response handling routine.

A SEND that specifies POST=RESP is completed. A CHECK is issued or a direct branch is made after it is determined that a negative response has been received.

A synchronous SEND with POST=RESP is issued resulting in a negative response being returned. ACF/VTAM schedules the SYNAD exit routine.

In this logic, the SSENSEI, SSENSMI, and possibly the USENSEI fields must be analyzed to determine what recovery action is possible. Appendix C of *ACF/VTAM Macro Language Reference* describes the possible settings of these fields and their meanings. Appendix H of *ACF/VTAM Macro Language Reference* shows the DSECT labels and possible values of these fields. In general, certain settings require that the logical unit be disconnected; other settings indicate that the situation is recoverable and that either sequence number synchronization or some other action can be taken.

Handling Retriable Completion (R0=X'08')

This return code indicates that an operation was not successful but should be requested again. The RPL does not contain any further information. The program should issue an EXECRPL macro instruction. Note that the parameters in the EXECRPL's RPL do not apply to the EXECRPL request itself but to the original request already specified in the REQ field of the RPL (in other words, the RPL associated with the request that is to be retried). The EXECRPL does not require a CHECK; it simply reinitiates the request in the portion of the program whose issuance caused entry to the SYNAD exit routine.

An error on retrying an operation may cause LERAD or SYNAD to be entered again. So that this situation can be recognized, a flag can be set before issuing the EXECRPL or other RPL-based macro. (See "Procedures to Follow in Writing Exit Routines" in Chapter 7.)

Note: *If an OPNDST ACCEPT fails because of insufficient storage, do not reissue the OPNDST ACCEPT. Instead, attempt to connect the logical unit with a SIMLOGON macro or VARY LOGON operator command before retrying the OPNDST ACCEPT request. For more information, see Appendix C of the* ACF/VTAM Macro Language Reference.

Handling Data Integrity Damage (R0=X'0C')

A number of errors fall into this category; the action to be taken depends on conditions that are unique to each program. For example, a program that has no means of recovering data that is lost on its way to a logical unit may decide that the logical unit will have to revert to some earlier point in a conversation or inquiry (may have to reenter its inquiry) and sends it a message to that effect. The logical unit can then reenter its request and start the process at an initial stage. On the other hand, a program may want to always keep a copy of its latest transmission to a logical unit so that it can retransmit it if required.

Handling Environment Errors (R0=X'10')

In general, this category requires intervention by a network or terminal operator or program support representative because the logical unit or some communication element between the application program and the logical unit is permanently or temporarily unavailable. The intervention action can consist of one or more of these actions:

Sending a message to the network operator

Sending a message to any log that is being kept in addition to the error logs being kept by ACF/VTAM and the NCP

Sending a message to a master terminal or logical unit

Sending a message to a logical unit that is associated with the logical unit for which intervention is required

If the error is temporary, the program can set a flag and retry the operation at a later time or set up an ECB that when posted will indicate that external intervention has taken place and the operation can be resumed. If the error is permanent, the logical unit (and perhaps other logical units associated with the failing logical unit) must be disconnected. If the program is dependent on the logical unit, the program must terminate.

Handling Logical Errors (R0=X'14' and X'18')

Most return codes that cause entry to a LERAD exit routine are likely to occur when the program is being debugged. These errors require that the program save as much information as necessary for debugging, perhaps request a dump of storage, and terminate the program at that point. (See Chapter 10, "Debugging a Program," for suggestions on what information to save.) Once the program has been debugged and it seems unlikely that any of these logical errors will occur, a message to the operator can be substituted for program termination in the event that a logical error occurs.

If the LERAD exit routine is not terminating and wants the request-issuing portion of the program to continue normally, registers 15 and 0 must be set to hex 00. If the LERAD exit routine is to indicate some action that the request-issuing portion of the program is to take (such as disconnecting a logical unit or terminating the program), a return code can be set in register 0 or register 15, or both. In any case, register 15 must be set to a nonzero value other than hex 20.

Handling ACF/VTAM Software Errors

When ACF/VTAM encounters a software error, it attempts to determine whether the error resulted from user action, the operating system, or ACF/VTAM. If an abnormal termination occurs in ACF/VTAM while running under a user's task, ACF/VTAM attempts to recover. During the attempted recovery, all ACF/VTAM requests from the user are rejected including CLOSE ACB.

If ACF/VTAM cannot recover from an error, the following sequence of events can occur:

1. The abnormal termination process continues and the user's application program is shut down.

2. If the user has an STXIT AB (for DOS/VS), STAE (for OS/VS1 and OS/VS2 SVS), or ESTAE (for OS/VS2 MVS) exit routine in the application program, the system eventually passes control to the user.
3. The user can perform a retry or pass control back to the system. If a retry is performed, the user can inspect user files, clean up user resources, and terminate. Termination passes control back to the system to continue the abnormal termination processing. The user cannot issue any ACF/VTAM requests, because the application program is shut down.
4. The system eventually passes control to the ACF/VTAM resource manager to schedule ACF/VTAM cleanup.
5. The user's task terminates.
6. The user can log on after the application program has been reactivated.

Note: *If the user does not give control back to the system after attempting a retry, ACF/VTAM resources are not cleaned up.*

For additional information on the STXIT AB, STAE, or ESTAE exit routines, see *DOS/VS Supervisor and I/O Macros,* GC33-5373; *OS/VS1 Supervisor Services and Macro Instructions,* GC24-5103; for OS/VS2 MVS, *OS/VS2 Supervisor Services and Macro Instructions,* GC28-0683; or, for OS/VS2 SVS, *OS/VS Supervisor Services and Macro Instructions,* GT29-6979.

Chapter 10. Debugging a Program

Chapter 9 discusses the handling of errors and special conditions. One kind of error that the program must handle, especially when trying to get a program to run, is a logical error. Some logical errors become evident when the program fails to execute as planned; these errors may or may not be related to the program's use of ACF/VTAM. Other logical errors are discovered by ACF/VTAM, which analyzes the error and provides feedback in the RTNCD and FDBK2 fields of the RPL. When ACF/VTAM discovers such an error following an RPL-based request or following a subsequent CHECK macro instruction, it schedules the application program's LERAD exit routine. (Chapter 7, "Using Exit Routines," describes how to use a LERAD exit routine.) Of the logical errors detected by ACF/VTAM, some can be handled without terminating the program, either because the program can correct the situation (for example, get a larger output area) or because the problem affects only one logical unit, which can be disconnected. Other logical errors, however, especially in the initial stages of developing a program, require termination of the program and the taking of a dump so that the error can be analyzed. This chapter discusses getting a program to execute as the programmer intends.

Debugging is performed at two times: before executing the program and after unsuccessful execution of the program.

Debugging before Executing the Program

It is better to find and correct errors before trying to execute the program. Two kinds of errors can be found before trying to execute the program:

Assembly errors

Program logic errors

In addition to finding obvious errors, the programmer can insert coding in the program to help him locate errors if the program fails to execute.

Checking for Assembly Errors

After coding the program and prior to assembling it, a list of things to check for includes:

Spelling errors in names (that is, a name spelled one way in the name field of the statement that defines the name and another way in a statement that refers to the name).

Names longer than 8 characters.

Duplication of names in IBM-supplied DSECTs (if DSECTs are used). See "Using the DSECT Macro Instructions and Assembler Instructions" in Chapter 8.

Executable instructions that refer to undefined Equate statements, constants, or storage areas.

Syntactical errors, such as blank spaces between operands.

Improperly specified operands, such as register notation where it is not allowed.

Continuation lines that do not begin in column 16.

Punctuation errors, such as missing commas or parentheses.

Checking for Program Logic Errors

Before trying to run the program, it is a good idea to "walk through" the main logic flow of the program to see whether there are any obvious errors. The parameters and register contents that are passed between ACF/VTAM, the main program, and exit routines (especially LERAD and SYNAD exit routines) can be simulated to make sure the parameters and registers will contain what they should.

Program Logic errors include errors in using ACF/VTAM macro instructions, control blocks, and linkage conventions that cannot be detected during assembly.

A Checklist

Here is a list of some checks to make and errors to watch for:

- Check each RPL-based macro instruction to be sure that an omitted parameter with an assumed previous setting retains the assumed setting. For example, be sure that if a CS setting is desired, another macro instruction has not set it to CA. Note that a default value exits in an RPL field only until the value is first changed; after that, the changed value is in effect and the default value no longer applies to the field.
- Make sure that a CLOSE macro is not issued in a TPEND exit routine; CLOSE must be issued in the main program.
- In checking the effects of alternative logic paths on various RPL fields, remember that, in addition to SEND and RECEIVE macro instructions, OPNDST, INQUIRE, CLSDST, and other RPL-based macro instructions can change the contents of an RPL's fields.
- In a LERAD exit routine or other feedback analysis routine, be sure to handle all logical errors that ACF/VTAM might indicate (all relevant feedback information described in Appendix C of *ACF/VTAM Macro Language Reference).*
- Make sure that addressability is established on entry to all exit routines except the LERAD and SYNAD exit routines.
- Make sure that register and save area instructions are properly performed on entry to a LERAD or SYNAD exit routine. If the LERAD or SYNAD exit routine contains an RPL-based macro instruction, the exit routine must: (1) point to its own save area in register 13, (2) save the return address that was passed to it in register 14, and (3) reload this address in register 14 prior to returning from the exit routine. Each recursive entry to LERAD and SYNAD must provide a previously unused save area.
- Make sure that LERAD and SYNAD exit routines are reenterable, if this is required. (See Figure 7-18.) Other exit routines do not have to be reenterable unless they can be used by more than one ACF/VTAM application program (VTAM considers each open ACB to be a separate application program).
- After an OPNDST macro instruction, ACF/VTAM moves the address of the NIB or list of NIBs (specified by the programmer in the NIB field of the RPL prior to making the OPNDST request) into the AREA field of the RPL. (This frees the NIB field so that ACF/VTAM can place the CID in it.) Before issuing another request using this RPL, perhaps to write an initial message to a connected logical unit, be sure that the AREA field is reset to contain the data area address.
- Be sure that PROC=RESPX and DFASYX are coded in the NIB if the program is to receive responses in a RESP exit routine and expedited-flow input in a DFASY exit routine, respectively.
- An INQUIRE macro instruction may set the RECLEN field of the RPL; a subsequent SEND using this RPL should respecify a value for RECLEN.
- An exit routine may branch to the main program. However, with the exception of authorized path, exit routines other than LERAD and SYNAD must return control to ACF/VTAM after their processing is completed, using the address in register 14.
- Be sure that each RPL-based request uses an inactive RPL (one against which a CHECK has been issued if it was active previously for an asynchronous request).

- If using an RPL exit routine for asynchronous operations, ensure that the RPL associated with the exit routine has a CHECK issued for it. This can be done either in the exit routine or in the main program after discovering an ECB posted by the exit routine.
- Use TESTCB rather than SHOWCB to analyze the contents of fields that contain only bit settings (for example, the SSENSEI field).

Adding Debugging Aids to the Program

Here are some ways a programmer can insert instructions to help in debugging the program.

Requesting a Dump

An application program can request a dump of the supervisor and program area by using macro instructions as follows:

In DOS/VS, by issuing any of the DOS/VS DUMP macro instructions (DUMP, PDUMP, and JDUMP)

In OS/VS1, OS/VS2 SVS, and OS/VS2 MVS, by issuing the SNAP or ABEND macro instruction

For information on the DOS/VS DUMP, PDUMP, and JDUMP macro instructions, see the *DOS/VS ACF/VTAM Debugging Guide*, GC27-0021. For information on the OS/VS1 and OS/VS2 SNAP and ABEND macro instructions, see the ACF/VTAM Debugging Guide for the operating system you are using.

A dump can also be requested by the system operator by using commands as follows:

In DOS/VS, by using the DUMP or DSPLY command

In OS/VS1, OS/VS2 SVS, and OS/VS2 MVS, by using the DUMP or CANCEL command

For detailed information on these commands, see the Operator's Reference manual for the operating system you are using.

Loading the Current Address before Each Macro

A specific register can be loaded with the current location counter before issuing a macro instruction. Then, if abnormal termination occurs, the register contains the location of the macro that ACF/VTAM was attempting to execute when the LERAD or SYNAD exit routine gained control:

```
LA          R5,*          PUT CURRENT ADDRESS IN REGISTER 5
RECEIVE     RPL=RPL1      RECEIVE A REQUEST
```

Using a Special Code to Indicate Which ABEND or DUMP Macro Was Issued

A defined storage area can be used to contain a code that will indicate the particular macro instruction that caused a program termination and dump. For example, a program can contain DUMP macro instructions in the main program and in the LERAD, SYNAD, LOSTERM, NSEXIT, and RESP exit routines. Knowing which of these DUMP macros caused the program termination will help localize the error. For example, in a DOS/VS program, the LERAD exit routine might contain:

```
LA          R5,4            PUT DUMP ID IN REGISTER 5
ST          R5,DUMPID       STORE CODE 4 IN DUMPID
DUMP                        CAUSE DUMP AND TERMINATE PROGRAM
```

Since the programmer knows the displacement of DUMPID from the beginning of the program, he can locate this field and its contents in the dump. The value of 4 in DUMPID tells the programmer that the dump was requested in the LERAD exit routine.

Saving Register 1 (Which Points to the RPL) (DOS/VS)

The code generated by the DOS/VS DUMP macro instruction uses register 1 and thus destroys its contents. Since register 1 will probably point to the RPL associated with the most recent ACF/VTAM request, its contents are important for debugging. The contents of register 1 should be saved in a defined storage area prior to issuing DUMP. For example, the previous example could be coded:

```
LA          R5,4            PUT DUMP ID IN REGISTER 5
ST          R5,DUMPID       STORE CODE 4 IN DUMPID
ST          R1,R1CONTS      SAVE REGISTER 1 CONTENTS
DUMP
```

The programmer, knowing the displacement of R1CONTS, can locate the field in the dump; it will contain the address of the RPL that was pointed to at the time of termination.

Using the ABEND Completion Code (OS/VS)

An alternative in OS/VS to using a defined storage field, as described above, is to specify a unique completion code as a parameter in issuing an ABEND macro instruction in each routine in which it is issued. This completion code will be printed on the system output device if the job step is being terminated (the STEP operand is specified) or placed in the tas control block of the task that is terminating (if STEP is not specified). For example, the LERAD exit routine might contain:

```
ABEND       4,DUMP          4 INDICATES LERAD ABEND
```

Writing a Debugging Record That Can Be Printed

For a complex program, it might be worth the effort to write a small routine in the LERAD exit routine (and possibly other exit routines where errors and special conditions are handled) that logs an error record. If the program terminates abnormally during the debugging stage, this record can be used instead of or in addition to the dump. This kind of record can be used for recording any kind of abnormal program activity, as well as logic errors.

Debugging after Executing the Program

If the program is executed successfully, but does not do exactly what it was intended to do, the program's design and general logic must be examined. If the program is not executed successfully, the error or errors causing the failure must be located. The most common tool is a dump of the storage associated with the program.

Important Information in a Dump

The contents of these registers and fields may be useful:

- The completion code, either your own in a designated location in the program or an ABEND completion code.
- Register 1, which contains the address of the RPL associated with the most recent request prior to program termination. In DOS/VS, this address must be located by looking in a program area where the contents of register 1 will have been stored prior to issuing DUMP.
- If the program terminated in a LERAD or SYNAD exit routine, register 13 contains the address of the save area of the LERAD or SYNAD exit routine. The second word

of that save area contains the address of the main program's save area, which contains the register contents of the main program at the time the LERAD or SYNAD exit routine was entered.

- If the program terminated in a LERAD or SYNAD exit routine, the contents of register 14, the address the LERAD or SYNAD exit routine was to return to upon completion, were saved in a program area. The address in this program area indicates the macro instruction that caused the LERAD or SYNAD exit routine to be entered.
- The RPL, pointed to by register 1, provides more information on the failing operation. The RTNCD field contains the recovery action return code, and the FDBK2 field contains the specific error return code; possible codes are described in Appendix C of *ACF/VTAM Macro Language Reference.* The contents of other RPL fields, such as the REQ field, which indicates the requested operation, may be useful in determining the source of the error.
- Use Appendix H of *ACF/VTAM Macro Language Reference* to determine the displacement of RPL and other ACF/VTAM control-block fields.
- The USER field of the RPL may be used to point to a logical-unit-associated control block. This control block may contain information about the status of the logical unit.

Replacing the Dump with a Programmer Message

If a logic error involves only one logical unit and the program can continue execution, a message can be written to the console instead of terminating the program. In OS/VS, such a message can be sent with a WTO macro instruction that specifies routing code 11 (programmer message). The programmer can then determine whether the error is a logical error that must be corrected. This approach might be used after the program has been debugged and is in productive operation. This message can also be written to a programmer log or to a printer.

System and ACF/VTAM Debugging Guides

These system and ACF/VTAM publications contain more information about debugging:

DOS/VS Serviceability Aids and Debugging Procedures, GC33-5380

OS/VS1 Debugging Guide, GC24-5093

OS/VS2 SVS Component Release Guide, GC27-0053

OS/VS2 Debugging Guide, GC28-0663

DOS/VS ACF/VTAM Debugging Guide, SY27-8007

OS/VS1 ACF/VTAM Debugging Guide, SY27-8005

OS/VS2 SVS ACF/VTAM Debugging Guide, SY27-8008

OS/VS2 MVS ACF/VTAM Debugging Guide, SY27-8006

Part 3. Sample Programs

The sample programs in this section show how ACF/VTAM macro instructions are used in application programs. Most ACF/VTAM services and language features are illustrated in these sample programs. In the text that explains the programs, some alternative facilities or techniques are discussed, but they are not shown in the figures.

Sample Program 1 (Chapter 11) shows how simple an ACF/VTAM program can be; it is not designed to portray a typical program. A completely coded version of Sample Program 1 is shown in Appendix D.

Sample Program 2 (Chapter 12) illustrates additional ACF/VTAM programming concepts and techniques and contains more detail than Sample Program 1. Sample Program 2 communicates with logical units in a 3601 Finance Communication Controller and with 3270 Information Display System terminals. It communicates with both types of terminals using the record-mode macro instructions (SEND and RECEIVE).

The logic of each program is displayed in flowcharts that show the major decision points but do not show all program instructions. The key macro instructions and operands are shown, and they are discussed in notes.

To follow the sample programs, the reader must be familiar with programming techniques for acquiring and handling control block areas, posting ECBs, and scheduling processing routines.

Chapter 11. Sample Program 1

Figure 11-1 shows an ACF/VTAM application program that receives a logon (request for connection) for a logical unit, connects it, reads input from any connected logical unit, processes the input, prepares a reply for output, and then writes the output to the logical unit.

Sample Program 1 demonstrates use of:

A LOGON exit routine and the acceptance of a logon

A request to receive input from any connected logical unit

Synchronous I/O requests

Continue-any and continue-specific modes

A SEND macro instruction to send a response rather than data

A request to *schedule* output

A RESP exit routine to handle a response received from the logical unit each time the logical unit receives data

A TPEND exit routine

For simplicity, error recovery routines and other special routines are omitted; these routines are discussed and shown in coded form in Appendix D. Appendix D consists of a coded ACF/VTAM application program that is based on Sample Program 1's general logic.

Sample Program 1 might be usable for an application program in which each transaction between the program and the logical unit consists of a short inquiry and a short reply. Because the program waits for the processing for one logical unit to be completed before reissuing a request for input from any connected logical unit, the program might not adequately serve a large number of logical units communicating with the program at the same time.

These notes are keyed to Figure 11-1.

1 The ACB1, defined in the program with the ACB macro instruction, is opened with the OPEN macro instruction. For example, this might be coded:

```
OPEN      ACB1
```

ACB1 contains:

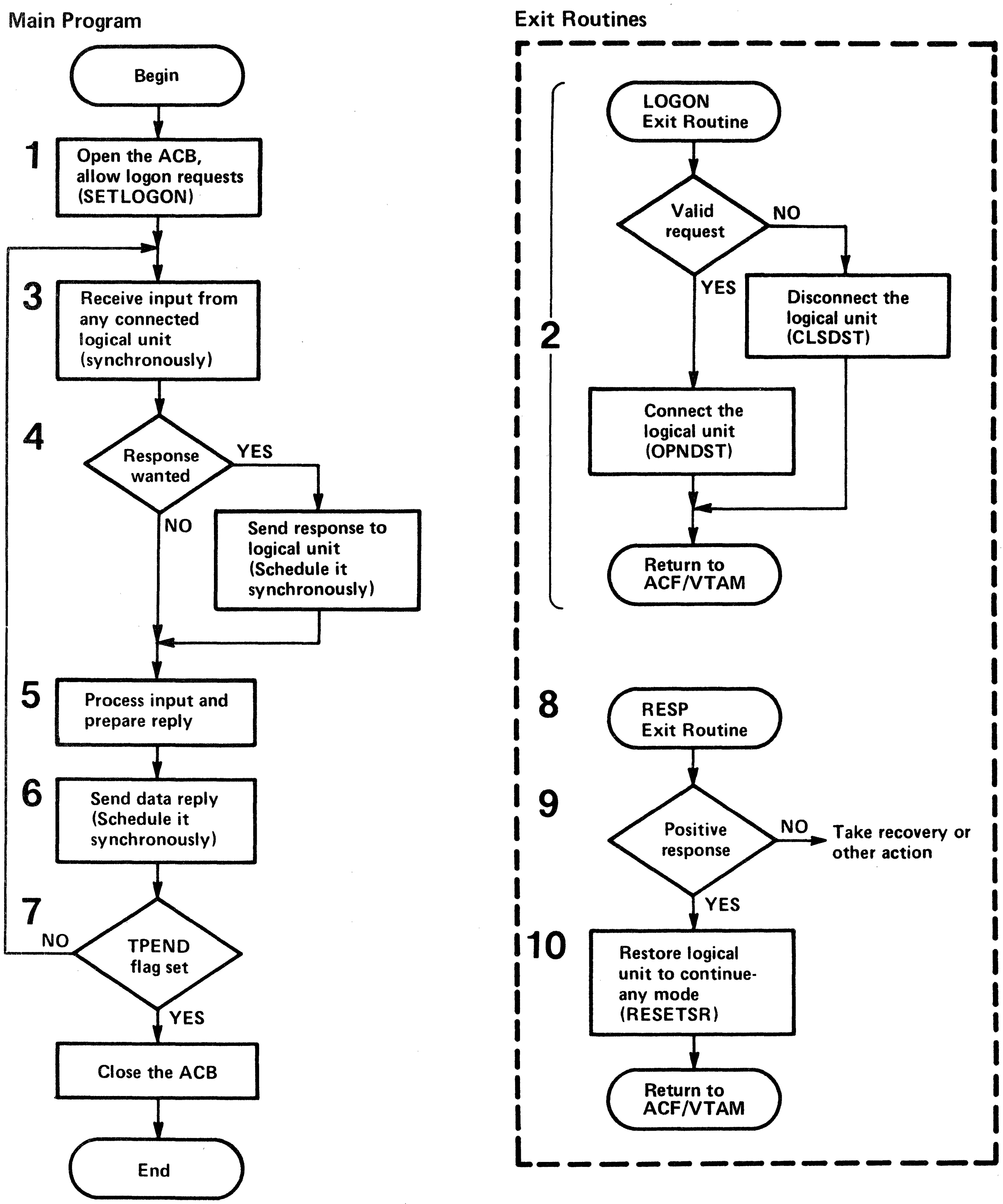
```
ACB1      ACB       AM=VTAM,APPLID=APPL1,EXLST=EXLST1,
                    MACRF=LOGON
```

APPL1 contains:

```
APPL1     DC        X'05'
          DC        C'PROG1'
```

PROG1 is the name of the APPL statement used to define the program during ACF/VTAM definition.

EXLST1 contains the names of the exit routines shown in Figure 11-1.



Main Program
Begin
1
Open the ACB, allow logon requests (SETLOGON)
3
Receive input from any connected logical unit (synchronously)
4
Response wanted
YES
NO
Send response to logical unit (Schedule it synchronously)
5
Process input and prepare reply
6
Send data reply (Schedule it synchronously)
7
NO
TPEND flag set
YES
Close the ACB
End
Exit Routines
2
LOGON Exit Routine
Valid request
NO
YES
Disconnect the logical unit (CLSDST)
Connect the logical unit (OPNDST)
Return to ACF/VTAM
8
RESP Exit Routine
9
Positive response
NO
Take recovery or other action
YES
10
Restore logical unit to continue-any mode (RESETSR)
Return to ACF/VTAM

Figure 11-1. The General Logic of Sample Program 1

After the OPEN macro instruction, a SETLOGON macro instruction is used to tell ACF/VTAM to begin processing logons for the program. The macro instruction might be coded:

```
SETLOGON   RPL=RPL1,OPTCD=START
```

2 The LOGON exit routine, whose address is specified in the LOGON operand of the EXLST macro instruction (whose address is in turn specified in the ACB macro instruction), is scheduled when ACF/VTAM receives a request for connection from or on behalf of a logical unit. Providing no other exit routine is being executed, the LOGON exit routine is given control. When the LOGON exit routine is completed, either the next-scheduled exit routine receives control or the main program regains control at its next sequential instruction.

An INQUIRE macro instruction can be used to obtain a user logon message when there is one.

```
INQ1       INQUIRE    RPL=RPL1,OPTCD=LOGONMSG,ACB=ACB1,
                      NIB=NIB1,AREA=LGNMSG.AREALEN=100
```

By examining the logon message in AREA, the exit routine determines whether the logical unit should be connected to the program. If so, an OPNDST macro instruction like this can be issued:

```
OPNDST     RPL=RPL1CONN,OPTCD=(ACCEPT,SPEC),NIB=NIB1
```

A NIB and an RPL are required for the OPNDST request; if the OPNDST is synchronous, the same NIB and RPL can be reused by the OPNDST each time the LOGON exit routine is entered. If necessary, the type of terminal and other communication characteristics can be obtained by using the INQUIRE macro, and the NIB can be properly initialized with this information prior to issuing the OPNDST macro. (The MODCB macro instruction can be used to initialize the NIB.)

If the logical unit is not to be allowed to use the program, a CLSDST macro instruction must be issued to notify ACF/VTAM that the logical unit's logon is being rejected. Although not shown in the flowchart, it may be desirable to connect the logical unit (using OPNDST), write an appropriate message to it, and then disconnect it.

3 In this example, the first request to receive input from any connected logical unit is issued in the main program. The request is synchronous; that is, the program indicates that it will wait until input is received from one of the logical units. In making the request, the program identifies an RPL and an input area. The macro instruction can be coded:

```
RECANY     RECEIVE    RPL=RPL1,AREA=AREA1,AREALEN=100,
                      RTYPE=DFSYN,OPTCD=(SYN,ANY,CS)
```

or it can be coded:

```
RECANY     RECEIVE    RPL=RPL1
```

where RPL is coded:

```
RPL1       RPL        AM=VTAM,ACB=ACB1,AREA=AREA1,
                      AREALEN=100,RTYPE=DFSYN,
                      OPTCD=(SYN,ANY,CS)
```

The input request can be coded with some operands appearing in the RPL and others in the macro instruction. If an operand that appears in the RPL is also coded in the macro instruction, the value in the macro instruction replaces the value in the RPL and is in effect not only for the current operation but for subsequent operations that use the RPL (unless changed by a manipulative macro instruction or by a subsequent request using the RPL).

RTYPE=DFSYN is specified so that the receipt of data or of a normal-flow command will complete the request. (The receipt of a response in this sample program causes the RESP exit routine to be entered.)

CS is specified so that the logical unit whose input is read by the RECEIVE will be put in continue-specific mode until its inquiry has been successfully answered. Thus, if the logical unit is still in continue-specific mode when the next RECEIVE with OPTCD=ANY is issued, input from that logical unit will not satisfy the continue-any-mode request. The logical unit will be put back in continue-any mode when a response has been received acknowledging that the reply to the inquiry arrived successfully; in this sample program, this is done in the RESP exit routine.

Assume that a logical unit just connected sends in the first inquiry to the program. The synchronous RECEIVE is completed. If register 15 contains 0, the operation was successful. If register 15 contains some value other than 0, an error or special condition occurred. A LERAD or SYNAD exit routine in the program may have been entered, and may have returned a code in register 15 or register 0 that indicates further action for the program to take. Whether or not a LERAD or SYNAD was entered, information is available in various feedback fields of the RPL for analysis. One of the errors that can occur is that the message arrived as an exception; this information is available as one of the feedback return codes in the RPL.

Assuming that the operation was successful, the identity of the logical unit whose input was received is provided in the ARG field of the RPL. Data is located in AREA1, and a SHOWCB or TESTCB can be used to determine its length (by examining the RECLEN field of the RPL).

4 The logical unit that sends the data can indicate that the program should issue a response to verify that the input has been received. There may be some occasions when the logical unit wants a response, perhaps to verify that a data base update message has been received so that it can free its buffers. On other occasions, such as an inquiry message, the logical unit may not want a response (or want a response only if an exception condition occurs); the answer to its inquiry will be forthcoming soon and will be implicit assurance that the inquiry arrived. The application program can examine the RESPOND field of the RPL to determine whether and under what conditions a response is required. If completion information following the RECEIVE indicates that the input was received normally and the RESPOND field indicates that a definite response is required, it is sent with a SEND macro instruction that can be coded:

```
SENDRESP   SEND        RPL=RPL1,STYPE=RESP,OPTCD=SYN
```

If completion information following the RECEIVE indicates that an exception message was received (in which case there will be no input to process), and the RESPOND field indicates that a response is requested, it is sent with a SEND macro instruction that might be coded:

```
SENDRESP   SEND        RPL=RPL1,STYPE=RESP,OPTCD=SYN,
                       RESPOND=EX
```

If the logical unit wants a response only in the event of an exception, the RESPOND field will already be set to EX and will not have to be reset in the SEND. Before issuing this SEND, the application program places sense information defining the exception in the RPL.

The same RPL used for the RECEIVE request can be reused for the SEND. Since a response is being sent (STYPE=RESP), no data area or length is needed. For a response, POST=SCHED is assumed. The operation is specified to be synchronous. However, because it is only being scheduled, the operation will take a relatively short time. As soon as the operation has been scheduled and the SEND is completed, the RPL can be reused.

5 The inquiry is analyzed and a reply is prepared by a processing routine. Disk I/O may be required. If so, the program waits until the reply is ready.

6 The reply is then sent with a SEND that requests the transmission of data (STYPE=REQ). In this sample program, the same area used to receive input is used for output. The macro instruction can be coded:

```
SENDDATA  SEND        RPL=RPL1,STYPE=REQ,RESPOND=(NEX,FME),
                      OPTCD=SYN,POST=SCHED
```

Since the RPL currently contains the value AREA=AREA1, this need not be respecified in the macro. So that the application program can determine whether or not the logical unit receives the data successfully, a response from the logical unit is requested (RESPOND=(NEX,FME)). The macro instruction requests control to be returned to the application program as soon as ACF/VTAM has completed scheduling the operation; the request will be completed synchronously (OPTCD=SYN) as soon as the sending of the output has been scheduled. Completion of the reply to the inquiry will be determined as a result of the program's receiving the definite response 1 in the RESP exit routine.

With the reply under way, a branch is made back to the RECEIVE so that input that may have been read into ACF/VTAM's buffers from another connected logical unit (that is not in continue-specific mode) will be read into the application program for processing.

7 If the ACF/VTAM network is being halted, the program's TPEND exit routine (not shown) is scheduled and entered by ACF/VTAM. This routine can indicate to the main program that the program is to terminate. The TPEND exit routine or the main program later can send final messages to connected logical units and do other close processing depending on whether the closedown is immediate or a routine end-of-day closedown. The main program, discovering the closedown requirement, must issue a CLOSE macro instruction to disconnect the application program from the network; any logical units not yet disconnected will, as a result of the CLOSE, be disconnected. Note that the CLOSE macro instruction must be issued in the main program and not in the TPEND exit routine. The program terminates by returning control to the operating system.

8 When a response to the data sent in 6 is received by ACF/VTAM, the RESP exit routine is scheduled and entered. On entry, register 1 points to a parameter list that points to a read-only RPL in ACF/VTAM's storage, whose feedback fields can be examined to determine the kind of response received.

9 If a negative response was received, the program can determine from sense information in the RPL whether or not to retry the operation, disconnect the logical unit, or take some other action.

10 If a positive response was received, the logical unit can be returned to continue-any mode so that the next request for input from any logical unit will include this logical unit as one whose input can be read by the application program. This is done by issuing:

```
RESETSR    RPL=RPL1R,OPTCD=CA,RTYPE=DFSYN
```

The RESP exit routine must have its own RPL available. The identity of the terminal to be placed back in continue-any mode must be put in the ARG field of the exit routine's RPL. The RESP exit routine then returns control to ACF/VTAM.

A completely coded version of Sample Program 1 is shown in Appendix D.

Chapter 12. Sample Program 2

Sample Program 2 is a more typical example of an ACF/VTAM program than Sample Program 1. Sample Program 1 (Chapter 11) should be read first.

Sample Program 2 communicates with logical units associated with 3600 Finance Communication Systems and SNA 3270 Information Display Systems. The logical units are associated with physical units that are connected to ACF/VTAM on nonswitched remote lines through a 3704 or 3705 Communications Controller with a network control program. Additionally, local 3270s are attached to ACF/VTAM through a channel. The ACF/VTAM application program uses the record mode to communicate with all of these terminals.

Figure 12-1 shows a possible configuration of terminals with which Sample Program 2 might communicate.

The application program in the 3601 controller can be written to perform certain functions that would otherwise have to be performed by the ACF/VTAM application program. For example, the 3601 application program can screen inquiries for correct format prior to forwarding them to the ACF/VTAM application program for processing, or it can collect inquiries from several terminals that form a work station and send them as one transmission to the ACF/VTAM application program.

The logic of Sample Program 2 is described at a high level in Figure 12-2 and accompanying notes. The logic of special routines is described in more detail in subsequent figures and accompanying notes.

Sample Program 2 uses the posting of ECBs (either by ACF/VTAM or within the program) and a central wait routine that discovers posted ECBs as a mechanism to handle a number of terminals alternately without having to suspend all program execution while waiting for I/O operation to be completed. After a request has been issued for an asynchronous operation, control is transferred to the wait routine, which discovers (or, if necessary, waits for) a posted ECB. The posted ECB may be associated with another terminal, and the wait routine branches to a point related to further processing for the terminal for which an operation has been completed. An understanding of the details of this technique is assumed in this discussion.

Although not discussed in detail, it is likely that Sample Program 2 would use a separate control block for each terminal that was actively using the program. This control block can include an input/output area. A separate RPL can be associated exclusively with each active terminal. The storage for these control blocks can be obtained from a fixed pool or be obtained dynamically and initialized with the GENCB macro instruction. This terminal control block and RPL area can be obtained and related to a terminal for the duration of its connection, for the duration of the program, for the duration of a transaction or conversation, or on some other basis. The ECB associated with a terminal can be located in the RPL or outside of it in some fixed relationship, perhaps just in front of it. In Sample Program 2, it is assumed that the storage for an ECB, RPL, and terminal control block is obtained and initialized in the LOGON exit routine and retained for the duration of the terminal's connection to the program.

The Organization and Flow of Sample Program 2

Figure 12-2 shows the principal routines in Sample Program 2; the notes below indicate how Sample Program 2 works. More detailed logic is shown and discussed in subsequent figures and notes.

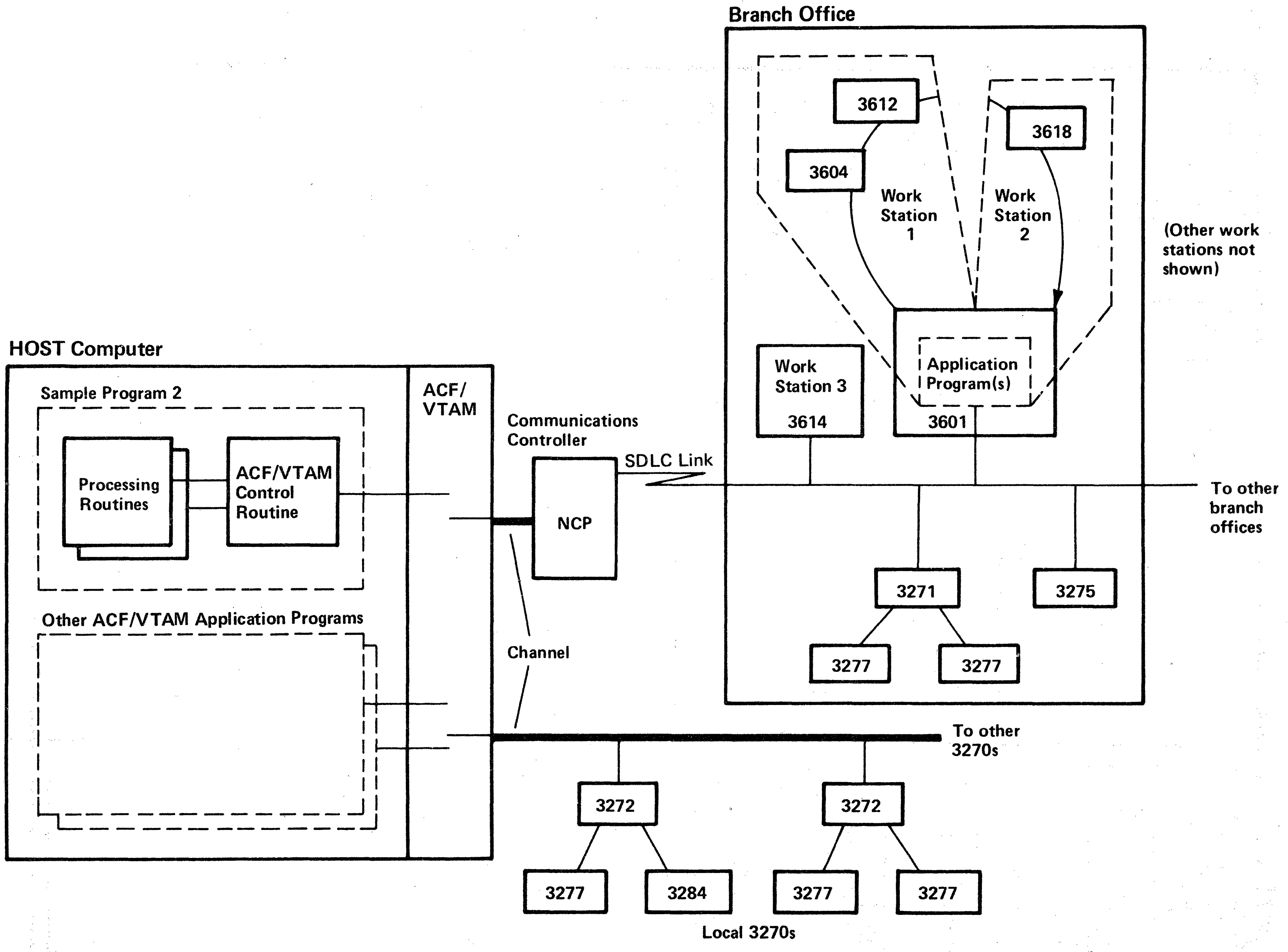


Figure 12-1. A Possible Data Communication System Configuration for Sample Program 2

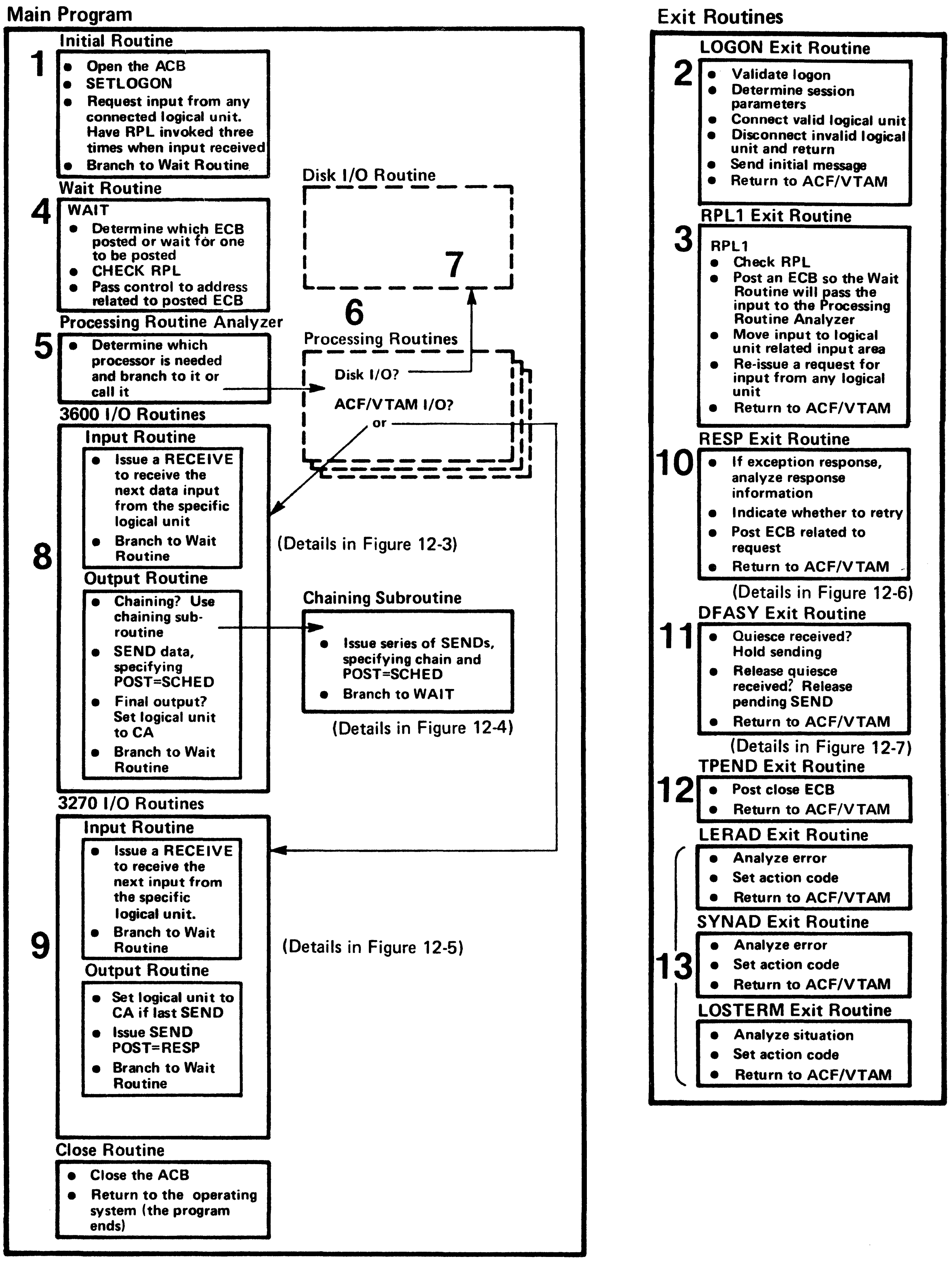

Main Program
Initial Routine
1
• Open the ACB
• SETLOGON
• Request input from any connected logical unit. Have RPL invoked three times when input received
• Branch to Wait Routine
Wait Routine
4
WAIT
• Determine which ECB posted or wait for one to be posted
• CHECK RPL
• Pass control to address related to posted ECB
Processing Routine Analyzer
5
• Determine which processor is needed and branch to it or call it
3600 I/O Routines
Input Routine
• Issue a RECEIVE to receive the next data input from the specific logical unit
• Branch to Wait Routine
8
Output Routine
• Chaining? Use chaining subroutine
• SEND data, specifying POST=SCHED
• Final output? Set logical unit to CA
• Branch to Wait Routine
3270 I/O Routines
Input Routine
• Issue a RECEIVE to receive the next input from the specific logical unit.
• Branch to Wait Routine
9
Output Routine
• Set logical unit to CA if last SEND
• Issue SEND POST=RESP
• Branch to Wait Routine
Close Routine
• Close the ACB
• Return to the operating system (the program ends)
Disk I/O Routine
7
6
Processing Routines
Disk I/O?
ACF/VTAM I/O?
or
(Details in Figure 12-3)
Chaining Subroutine
• Issue series of SENDs, specifying chain and POST=SCHED
• Branch to WAIT
(Details in Figure 12-4)
(Details in Figure 12-5)
Exit Routines
LOGON Exit Routine
2
• Validate logon
• Determine session parameters
• Connect valid logical unit
• Disconnect invalid logical unit and return
• Send initial message
• Return to ACF/VTAM
RPL1 Exit Routine
3
RPL1
• Check RPL
• Post an ECB so the Wait Routine will pass the input to the Processing Routine Analyzer
• Move input to logical unit related input area
• Re-issue a request for input from any logical unit
• Return to ACF/VTAM
RESP Exit Routine
10
• If exception response, analyze response information
• Indicate whether to retry
• Post ECB related to request
• Return to ACF/VTAM
(Details in Figure 12-6)
DFASY Exit Routine
11
• Quiesce received? Hold sending
• Release quiesce received? Release pending SEND
• Return to ACF/VTAM
(Details in Figure 12-7)
TPEND Exit Routine
12
• Post close ECB
• Return to ACF/VTAM
LERAD Exit Routine
• Analyze error
• Set action code
• Return to ACF/VTAM
SYNAD Exit Routine
13
• Analyze error
• Set action code
• Return to ACF/VTAM
LOSTERM Exit Routine
• Analyze situation
• Set action code
• Return to ACF/VTAM

Figure 12-2. The Organization and Flow of Sample Program 2

Figure 12-2 shows the main program and the exit routines as separate groups of routines. This is a logical rather than a physical separation; exit routines are distinctive because they are entered only when an event occurs that requires handling by an exit routine. When an exit routine is scheduled, ACF/VTAM suspends execution of the main program until the exit routine completes its processing and returns to ACF/VTAM. Only one exit routine can be executed at a time; if an exit routine event occurs while an exit routine is being executed, the second exit routine is scheduled for entry only after the first exit routine is completed. The LERAD and SYNAD exit routines are exceptions to this general rule; they can be entered as the result of an RPL-based operation, such as OPNDST, RECEIVE, or CHECK in another exit routine (in which case, they may be viewed as extensions of the exit routine that caused them to be entered).

Except for the LERAD and SYNAD exit routines, each exit routine must establish its own addressability, be executed, and then return to ACF/VTAM; ACF/VTAM's registers need not be saved or restored. A temporary branch to part of the main routine can be made from an exit routine—and common code can be shared,—but the exit routine would be considered to be in progress until control is returned to ACF/VTAM. The LERAD and SYNAD exit routines are furnished addressability as the result of loading registers (a user save area address is passed in register 13).

Except for an RPL exit routine, whose address is specified in the RPL or request macro instruction, the addresses of exit routines to be associated with the program are defined in an EXLST macro instruction. In addition, for DFASY, RESP, and SCIP exit routines, different exit lists can be defined for different terminals or sets of terminals. In Sample Program 2, one exit list is assumed; the address of this exit list is provided to ACF/VTAM in the EXLST operand of the ACB when the ACB is opened.

The following notes are keyed to the numbers in Figure 12-2.

1 The ACB is opened and a SETLOGON is issued. A request to read input from any logical unit is issued; the operation is to be completed asynchronously, and an RPL exit routine (RPL1) is designated for scheduling by ACF/VTAM when the operation is completed. The logical unit whose input is read into Sample Program 2 is to be put into continue-specific mode. Thus, subsequent requests to read input from any logical unit, issued in the RPL exit routine, will exclude the logical unit whose input was just read and with whom the program will now be in specific communication The RECEIVE can be coded:

```
RECEIVE     RPL=RPL1ANY,AREA=AREAANY,AREALEN=100,
            RTYPE=DFSYN,OPTCD=(ASY,ANY,CS),EXIT=RPL1
```

Issuing more than one request to receive input from any logical unit at this point can improve efficiency. If three RECEIVEs are issued using three different RPLs and data areas, when one RECEIVE is completed (thus causing the RPL1 exit routine to be scheduled and entered), there are two other RECEIVEs outstanding that allow scheduling of RPL1. The RECEIVE that is completed first can be reissued in the RPL1 exit routine.

The RPLs and input areas can be assembled in the program as fixed areas and reused each time the program issues a request to read input from any logical unit.

A branch is made to the wait routine, which waits for the first input to arrive from a connected logical unit. The initial routine is executed only once.

2 Both 3600 and 3270 logical units can be connected in the LOGON exit routine. The INQUIRE macro instruction is used to determine which type of logical unit is being connected. The particular type of SNA terminal product (3600, 3790, 3270,

etc.) need not be identified by the ACF/VTAM application program. Instead, the ACF/VTAM application program distinguishes between types of logical units on the basis of the set of session parameters associated with the logical unit. In this example, 3600 logical units have a different set of session parameters from 3270 logical units; the program can relate a logical unit that is being connected with one of these sets and use the appropriate routine to communicate with the logical unit. Storage that is to be associated with this terminal can be obtained from a pool or can be obtained dynamically from the system. (The storage can include an ECB, an RPL, and a logical unit block for additional logical-unit-related information.) The address of this storage or any other logical-unit-related information can be put in the USERFLD field of the NIB, using the MODCB macro; when the logical unit is connected, ACF/VTAM will save this address and return it to the program following completion of each subsequent input from the logical unit.

A logical unit can be connected as the result of a logical-unit-initiated logon, installation-initiated (automatic) logon, network operator-initiated logon, or application-program-initiated logon. A 3600 logical-unit-initiated logon could be the result of either some logical-unit-operator action or could be initiated solely by the 3601 application program without involving a logical unit operator (in either case, the actual request would be transmitted by the 3601 application program).

Figure C-1 in Appendix C shows part of the general sequence of events that occur prior to and during a logon. Here is a sequence of events that can occur prior to and during a 3600 logical-unit-initiated logon:

a. The 3601 and its logical units, defined to ACF/VTAM during ACF/VTAM definition, are made an active part of the ACF/VTAM network (perhaps by a network operator VARY command).

b. As a result of receiving an activation request for the 3601 controller, ACF/VTAM sends an Activate Physical Unit command to the 3601 controller. The 3601 acknowledges the command and responds that it is ready for operation. (This command is followed by an Activate Logical Unit command to one or more of the logical units [logical work stations] associated with that particular 3601.)

c. After the Activate Logical Unit command is received, the 3601 application program can either wait for a terminal operator at a 3601 work station to indicate that the logical unit (work station) is to be logged on, or can issue a logon on its own initiative. This is done by sending an Initiate command to ACF/VTAM specifying the name of the ACF/VTAM application program with which the logical unit is to be connected. The Initiate command may also specify a logon mode (set of session parameters) and a user-defined logon message.

d. ACF/VTAM, receiving the Initiate command, schedules Sample Program 2's LOGON exit routine.

e. In the LOGON exit routine, after confirming the validity of the logon, the logical unit is connected, using an OPNDST macro instruction. The OPNDST causes ACF/VTAM to send a Bind command (containing the session parameters) and to issue a Start Data Traffic (SDT) command to the logical unit (assuming SDT=SYSTEM was specified in the NIB used for connection). On receipt of a response to the SDT signal, the logical unit is connected to the ACF/VTAM application program and the OPNDST is complete. (If SDT=APPL is specified in the NIB used for connection, the ACF/VTAM application program must itself send the initial SDT, using the SESSIONC macro instruction.)

The OPNDST can be specified as a synchronous operation or an asynchronous operation. If the latter, the LOGON exit routine can either identify an ECB to be posted or an RPL exit routine to be scheduled as soon as the connection has been made.

The same RPL and NIB can be used for each logical unit being connected in the LOGON exit routine.

It may be desirable to write an initial message to the connected logical unit; this could be done from the LOGON exit routine or in an RPL exit routine following an asynchronously scheduled OPNDST.

So that the session parameters to be associated with the logical unit can be identified (which in this program determine whether the 3600 or the 3270 I/O routines are used with the logical unit), an INQUIRE macro instruction optionally specifying OPTCD=SESSPARM can be issued. This will allow the session parameters and logon data to be inspected and perhaps saved in the storage that is to be associated with the logical unit.

3 As soon as the first input is received from a connected logical unit, the operation started by the request to read input from any logical unit, issued in the initial routine, is completed. RPL1 is then scheduled and entered.

RPL1 can use the USER or the ARG field of the RPL of the request just completed to locate the logical unit storage and the identity of the logical unit. On entry to RPL1, the address of the RPL is in register 1.

A CHECK macro is required to free the RPL for reuse; it will also cause LERAD or SYNAD exit routines to be entered if any error occurred.

RPL1 posts an ECB so that, subsequently, the wait routine in the main program will determine that the input has been received and pass it to the processing routine analyzer. The RPL1 exit routine then reissues a request to read input from any logical unit. Because a logical-unit-related RPL obtained in the LOGON exit routine is used for subsequent I/O with the terminal just read, the RPL causing entry to RPL1 can be continuously reused by the RECEIVE in RPL1. The operation is to be asynchronous with relation to the program and RPL1 is to be reentered each time the request is completed. Note that the logical unit whose input caused entry to RPL1 is now in continue-specific (CS) mode. The RECEIVE can be coded identically to the RECEIVE in the initial routine.

Although not shown in Figure 12-2, the RPL exit routine sends a positive response to the message that caused it to be entered if a positive response is requested by the logical unit. If a response is to be sent for an exception condition, sending the negative response is probably performed in a SYNAD exit routine after a CHECK was issued.

An alternative to having an RPL exit routine for the RECEIVE with OPTCD=ANY and related logic is to have this logic located in the main part of the program and have an ECB posted. In Sample Program 2, one advantage to using an RPL exit routine is that input resulting from a RECEIVE with OPTCD=ANY is handled sooner in an RPL exit routine than if an ECB were to be posted (which would require waiting until the next entry to the main program's wait routine). This gives some preference to handling the first input of a new transaction or conversation over transactions or converstations already in progress.

4 The wait routine waits on a list of ECBs, with each ECB associated with a separate RPL. When an ECB is posted, the wait routine is activated, and the routine searches the ECB list to find the posted ECB and zeros it out. The routine then issues a CHECK macro instruction. This macro instruction clears the RPL for reuse in the next request involving the logical unit, and if an error occurred, the macro instruction causes the LERAD or SYNAD routine to be entered. On return from CHECK, the feedback fields of the RPL contain information provided by ACF/VTAM; in addition, the LERAD or SYNAD routine may have indicated action to be taken. If the operation was successful, the wait routine branches to the address associated with the ECB. (In the case of the first input of a transaction or conversation, that address is the one for the processing routine analyzer.) When control is returned to the wait routine, the routine again searches the ECB list to see if another ECB has been posted. If a posted ECB is found, processing continues as described above. If not, the RECEIVE with OPTCD=ANY is issued again, and the program enters the wait state.

5 The processing routine analyzer, which can consist of separate routines for different types of logical units, analyzes the input and branches to or calls the appropriate processor. This processor may have been coded in a higher level language, such as COBOL or PL/I.

6 The processing routine processes the input and prepares the output. This may require one or more disk I/O operations, which can be performed by calling a common disk I/O routine. When output is ready, or, in a conversation the next input is required, the processing routine requests ACF/VTAM I/O, causing control to pass to an appropriate ACF/VTAM I/O routine.

7 The disk I/O routine requests a disk I/O operation asynchronously and uses the wait routine to wait for completion. This allows processing for other logical units to continue while a disk I/O operation for one logical unit is under way.

8 Although not shown, a processing routine can return control to the next sequential instruction in the main program from which it was called; a branch can then be made to a common I/O routine, which in turn branches to a 3600 or a 3270 input or output routine. A special routine might be required to edit 3270 input and format 3270 output.

If the logical unit has the 3600 session parameters, an input or an output operation is requested as appropriate. The operation is specified as asynchronous; completion is determined when the ECB related to the logical unit is posted. Before issuing the request, the address to which the wait routine should branch (the return address is the processing routine) is placed in the ECB-related logical unit block.

If additional input is requested, the input, when it arrives, will not be used to satisfy the outstanding RECEIVE request in RPL1 because the logical unit is now in CS mode.

If output is requested, the data can be sent in a chain of transmissions which is useful with output that is passed from the 3601 application program to a 3610 printer. The 3601 application program can store all elements of the chain in a buffer until the entire chain is received (or print each element as it arrives). The ACF/VTAM application program would ensure arrival of the entire chain by receiving a single positive response sent by the 3601 application program when the last element of the chain is received. This notifies the ACF/VTAM application program that the data transfer was successful.

If the output completes a transaction or conversation, the logical unit is reset to continue-any (CA) mode so that input that begins the next transaction or conversation will satisfy the RECEIVE with OPTCD=ANY request that is issued in RPL1.

Details about the 3600 I/O routine are provided in Figure 12-3 and in accompanying notes.

9 If the logical unit has the 3270 session parameters, different I/O routines are required. The size of I/O areas required may be different and the range of input that may arrive may be wider. An additional requirement is the use of brackets for controlling the overlap of input and output with 3270.

Details about the 3270 I/O routine are provided in Figure 12-5 and in accompanying notes.

10 The RESP exit routine is scheduled and entered when a response arrives from a logical unit. A response will be received by ACF/VTAM because the ACF/VTAM application program requested it in the RESPOND operand of the SEND macro. When the response is received, the operation, only scheduled in the 3270 or 3600 output routine, is now completed and the RESP exit routine can now post the ECB. If the operation was successful, the response is positive; if an error occurred, a negative response is indicated in the RPL. The RESP exit routine can set up parameters and branch to the SYNAD exit routine which analyzes the error and takes corrective action. The ECB is posted and control is returned to ACF/VTAM.

Details of the RESP exit routine are in Figure 12-6 and accompanying notes.

11 In Sample Program 2, two kinds of expedited-flow commands can be received from a 3601 application program: a command to stop sending to the logical unit at the end of the chain that is currently being sent (a Quiesce at End of Chain [QEC] command) and a command to reinitiate sending after previously being requested to stop (a Release Quiesce [RELQ] command). The use of these commands can be desirable if a work station operator wants to interrupt a long series of printing so that keyboard input can be entered. After handling the operator's request, the ACF/VTAM application program can resume printing. When either of these commands is received, ACF/VTAM schedules Sample Program 2's DFASY exit routine.

This exit routine does not apply to 3270 operation. Details of this routine are shown in Figure 12-7 and accompanying notes.

12 The TPEND exit routine is scheduled and entered when the network operator enters a HALT command or when ACF/VTAM terminates itself or is abnormally terminated. In addition to other possible processing, the TPEND exit routine posts a special close ECB so that, subsequently, the main program's wait routine will branch to a CLOSE macro instruction in the main program.

13 The LERAD, SYNAD, and LOSTERM exit routines handle different categories of errors or unusual situations. The LERAD or SYNAD exit routine can be entered as the result of any RPL-based request. The LOSTERM exit routine is scheduled asynchronously when certain situations occur, such as the deactivation of a connected logical unit by the terminal operator. Like other parts of the program, the LOSTERM exit routine can branch to the LERAD or SYNAD exit routine for problem analysis. The LERAD exit routine primarily handles logical errors; it is most likely for these to occur during the debugging stages of the program. This exit

routine can gather information, format it, and save it for programmer analysis after the program ends. The SYNAD exit routine primarily handles physical errors; it determines what general action should be taken (for example, retry, disconnection of the logical unit, termination of the program, or sending a message to the network operator) and either takes the action or passes an action code to the main program where the action is taken. The SYNAD exit routine may also want to record information related to situations that it handles for later problem analysis.

A number of error situations must be perceived and analyzed as the result of receiving a response from a logical unit; the response is analyzed following a SEND with POST=RESP specified, following a RECEIVE with RTYPE=RESP specified, or after a RESP exit routine is entered. Errors or special situations that result in negative responses cause the SYNAD exit routine to be entered when a synchronous macro or a CHECK macro is issued; the SYNAD exit routine can determine the cause of the negative response by analyzing sense information in the RPL and then take appropriate action. The program, after determining that negative response has been received, can also branch directly to the SYNAD exit routine.

The Logic of the 3600 I/O Routine

This routine is entered directly or indirectly (perhaps from a common I/O branching routine in the main program) as the result of a request for input from a specific terminal with which a processor is currently engaged in a transaction or a conversation.

Figure 12-3 shows the logic of the 3600 I/O routines. The following notes are keyed to this figure.

1 If the processor's request is for input, the information that must be passed to ACF/VTAM is set up and a RECEIVE is issued. The address of the logical unit's RPL is put in a register and the address of the input area associated with the terminal and the length of the area are put in other registers. Since data is to be read, RTYPE=DFSYN is specified. The operation is to be asynchronous and input is to be read only from the specific terminal (whose CID is located in the RPL's ARG field). The ECB associated with the logical unit is specified for posting by ACF/VTAM when the operation is completed. After issuing the RECEIVE request, register 15 contains 0 if the request is accepted, or some other return code if it is not. If the request is accepted, the wait routine is returned to, after setting the next sequential instruction in this routine as the address to be branched to when the ECB is posted.

Note: *For simplicity, most checks of register 15 are not shown in Sample Program 2.*

2 When data is received from the logical unit, ACF/VTAM posts the ECB. When the wait routine discovers the posted ECB, it branches to the indicated location in the 3600 I/O routine. The RESPOND field of the RPL can be tested to determine whether the logical unit wants a definite response returned so that it will know positively that the input was received. If so, a SEND is issued, indicating that a response is to be sent to the logical unit (STYPE=RESP).

The SEND that sends the response, if requested, is scheduled synchronously. ACF/VTAM assumes POST=SCHED. Since no response can be returned to a response, once the request to send the response is accepted, the ACF/VTAM application program considers the sending of the response as complete.

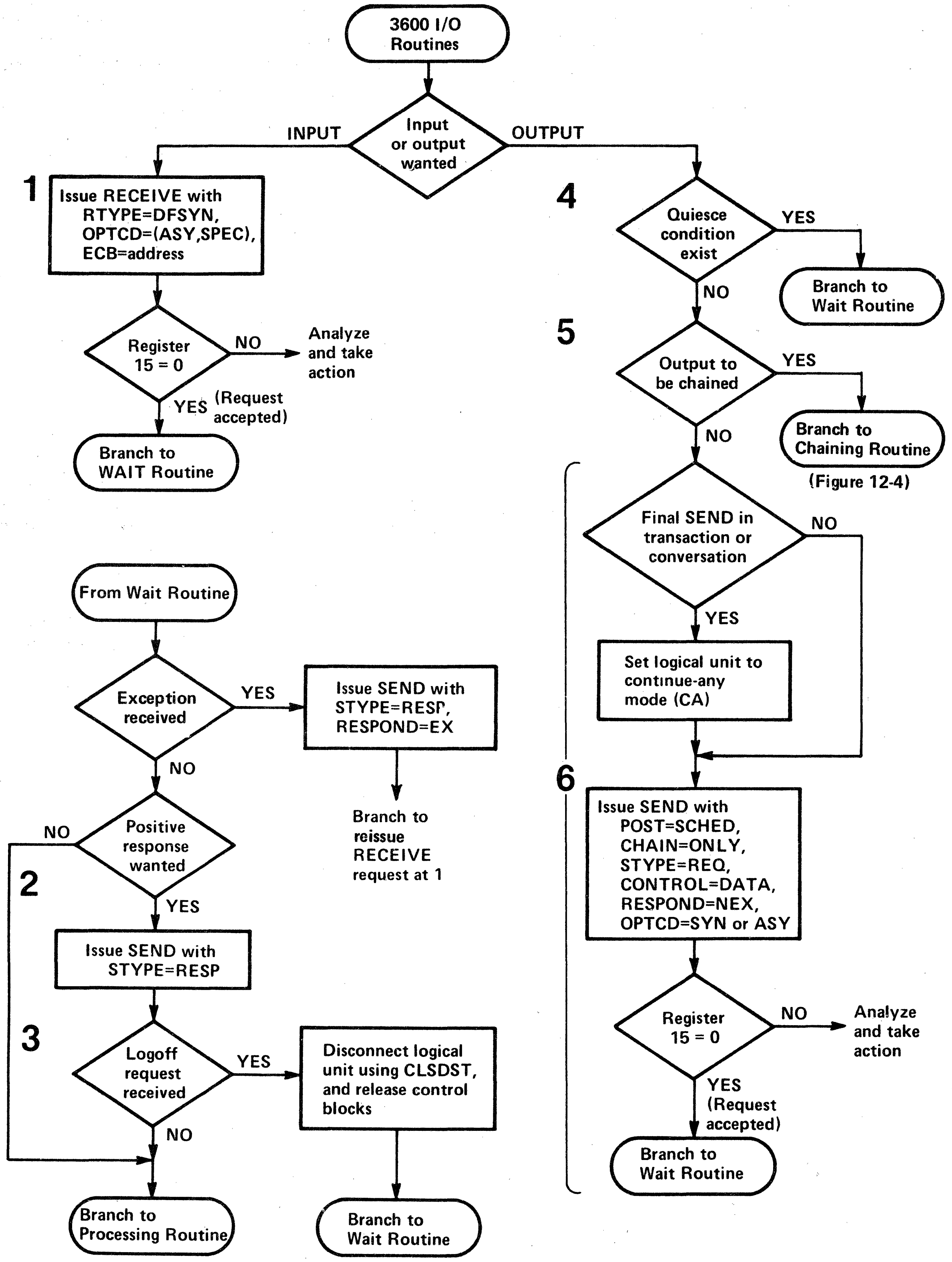

3600 I/O Routines
Input or output wanted
INPUT
OUTPUT
1
Issue RECEIVE with RTYPE=DFSYN, OPTCD=(ASY,SPEC), ECB=address
Register 15 = 0
NO
Analyze and take action
YES (Request accepted)
Branch to WAIT Routine
4
Quiesce condition exist
YES
Branch to Wait Routine
NO
5
Output to be chained
YES
Branch to Chaining Routine (Figure 12-4)
NO
6
Final SEND in transaction or conversation
NO
YES
Set logical unit to continue-any mode (CA)
Issue SEND with POST=SCHED, CHAIN=ONLY, STYPE=REQ, CONTROL=DATA, RESPOND=NEX, OPTCD=SYN or ASY
Register 15 = 0
NO
Analyze and take action
YES (Request accepted)
Branch to Wait Routine
From Wait Routine
Exception received
YES
Issue SEND with STYPE=RESP, RESPOND=EX
Branch to reissue RECEIVE request at 1
NO
2
Positive response wanted
NO
YES
Issue SEND with STYPE=RESP
3
Logoff request received
YES
Disconnect logical unit using CLSDST, and release control blocks
NO
Branch to Processing Routine
Branch to Wait Routine

Figure 12-3. The Logic of the 3600 I/O Routine

If input arrives unsuccessfully or out of sequence (indicating that some input was lost), ACF/VTAM completes the ACF/VTAM application program's input request with an indication that a negative response must be returned; no input is forwarded to the program. The application program sends the negative response. The input can be reissued.

Although not shown, the ACF/VTAM application program can also return a negative response to input that is successfully received. This might be done where such a response is understood by both the application program and the logical unit. The USENSEO field of the RPL can be used to convey exception information.

3 If the input contains a request to log off (to be disconnected from this program), the logical unit is disconnected by issuing a CLSDST macro instruction, and the control blocks associated with the logical unit are returned to the system or to a pool. Optionally, a message can be sent to the logical unit, confirming logoff, prior to issuing CLSDST.

The above description of the 3600 input routine assumes that a CHECK macro is issued in the wait routine upon completion of each requested input operation; if an error occurs, CHECK causes entry to the LERAD or SYNAD exit routine which takes appropriate action. This can include sending the negative response for 2 . It might be noted that the input routine can issue a request to receive any kind of input: a normal-flow data or an SNA command (DFSYN), an expedited-flow command, which is always an SNA command, (DFASY), or a response (RESP). In this sample program, DFASY and RESP-type input is handled by ACF/VTAM-scheduled DFASY and RESP exit routines, but the logic in these routines could have been branched to after determining in the wait routine or 3600 input routine that DFASY or RESP information had been received. DFSYN means that either data or normal-flow SNA commands can be received; although not shown in this example, normal-flow commands such as a Quiesce Complete (QC) command might be receivable in some applications, in which case, such commands have to be responded to.

4 When an output request from a processor is received by the 3600 output routine, the routine does not process the request if the logical unit has quiesced the ACF/VTAM application program; instead, the I/O routine branches to the wait routine. The processor must wait until the quiesce is released at which time the ECB for this logical unit is posted, a pending send request detected, and the routine is reentered. (This logic is discussed in "The Logic of the DFASY Exit Routine.")

5 If the output request is chained to other output requests, a branch is made to a chaining output routine (see Figure 12-4).

6 If output is not being chained, a SEND is issued that includes the operand CHAIN=ONLY. If the output completes a transaction or conversation, the logical unit is returned to continue-any mode; its next input will satisfy the RECEIVE with OPTCD=ANY request issued in RPL1. The request can specify scheduling of the operation (POST=SCHED) with completion to be determined as the result of a positive or negative response (RESPOND=NEX) that will cause scheduling of the RESP exit routine. (The RESP exit routine will post the ECB associated with the logical unit, thus notifying the ACF/VTAM application program and the processor that the output request was completed.) The output routine then branches to the wait routine.

The Logic of the 3600 Chaining Output Routine

Figure 12-4 shows the logic of the 3600 chaining output routine. The following notes are keyed to this figure.

1 The number of elements in the chain can vary or always be the same. Assuming that it varies in Sample Program 2, the number of chain elements must be determined so that the routine will know when to send the last element. It might be convenient to picture this routine being entered to send a report to an administrative line printer; this report may vary in length between 20 and 100 printer lines. Each line is sent to the 3601 logical unit as a chain element. The 3601 logical unit determines how many lines (chain elements) it collects before sending them on to the administrative printer.

   This chaining routine can be passed all of the data to be sent in a chain or only part of it. In other words, the routine is not necessarily sending an entire chain each time it is entered. The logic discussed here, however, assumes that all or, in a retry situation, the last part of a chain is being sent.

2 Because one of the advantages of chaining output is to reduce the number of required responses while still breaking output into segments (request units) that can be interspersed on the communication path, all SEND macros other than the last one specify that a response is to be returned only if an exception is noted (RESPOND=EX). When the last segment is received, a positive response is returned, and the ACF/VTAM application program recognizes that the entire chain arrived successfully. When RESPOND=EX is specified, the scheduling of output (POST=SCHED) is assumed by ACF/VTAM; it does not have to be specified.

3 In some cases, it may be necessary to save the sequence number of the first element sent in a chain. This number is available as soon as sending has been scheduled. It can be obtained from the SEQNO field of the RPL by using the SHOWCB macro. In case all or part of the chain must be resent (a negative response arrives in the RESP exit routine), the first-in-chain sequence number may be useful in determining where to start resending. It may also be necessary (not shown here) to reset the beginning sequence number for the logical unit that is receiving the chain; this number is sent to the logical unit by using a SESSIONC macro. The sequence number can be saved in the control block associated with the logical unit.

4 All elements except the first and last are middle elements (CHAIN=MIDDLE).

5 For the last element in the chain, the SEND macro must identify it as the last (CHAIN=LAST) and request the return of a response (RESPOND=NEX). Either POST=SCHED or POST=RESP can be specified.

   When the response is received, if POST=SCHED was specified, the RESP exit routine posts an ECB, causing the wait routine to return to the processor that originated the output request. If POST=RESP was specified, ACF/VTAM posts an ECB or schedules an RPL exit routine.

The Logic of the 3270 I/O Routine

Figure 12-5 shows the logic of Sample Program 2's 3270 I/O routine. With few exceptions, the ACF/VTAM application program using record-mode macro instructions need not distinguish between locally attached, BSC, and SDLC 3270s. Data received from a 3270 begins with an AID (Attention Identifier) character. Data sent to the 3270, whether local or remote, must begin with a 3270 command character, indicating Erase,

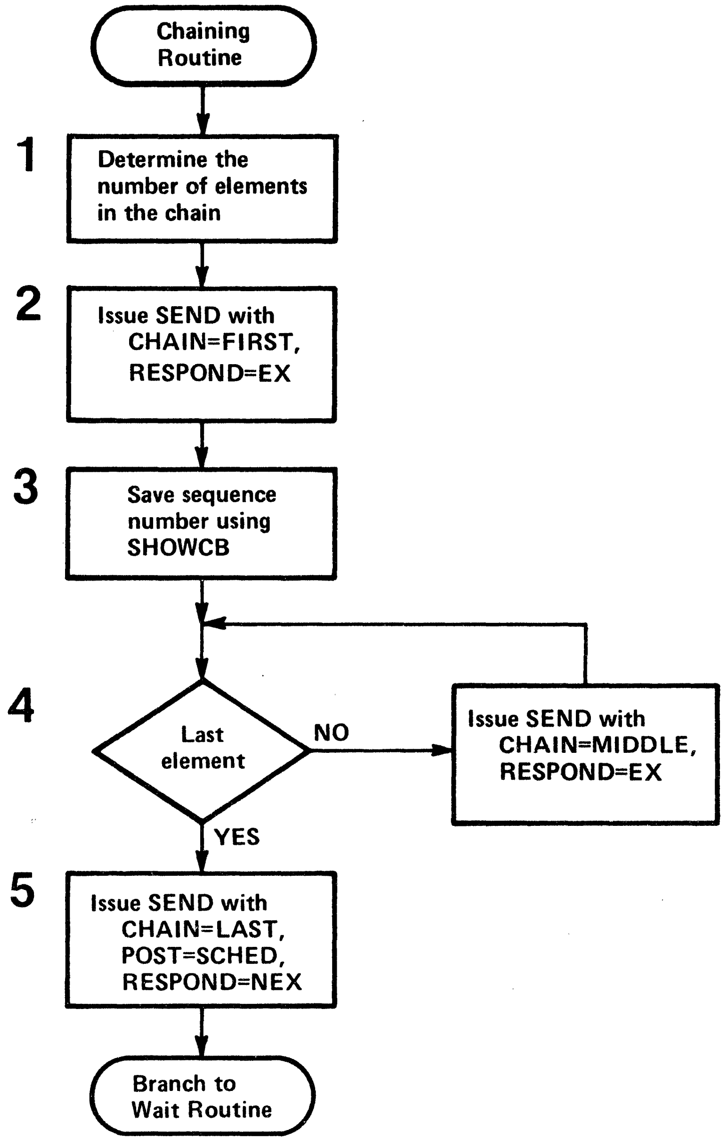

Chaining Routine
1
Determine the number of elements in the chain
2
Issue SEND with CHAIN=FIRST, RESPOND=EX
3
Save sequence number using SHOWCB
4
Last element
NO
Issue SEND with CHAIN=MIDDLE, RESPOND=EX
YES
5
Issue SEND with CHAIN=LAST, POST=SCHED, RESPOND=NEX
Branch to Wait Routine

Figure 12-4. The Logic of the Chaining Output Routine

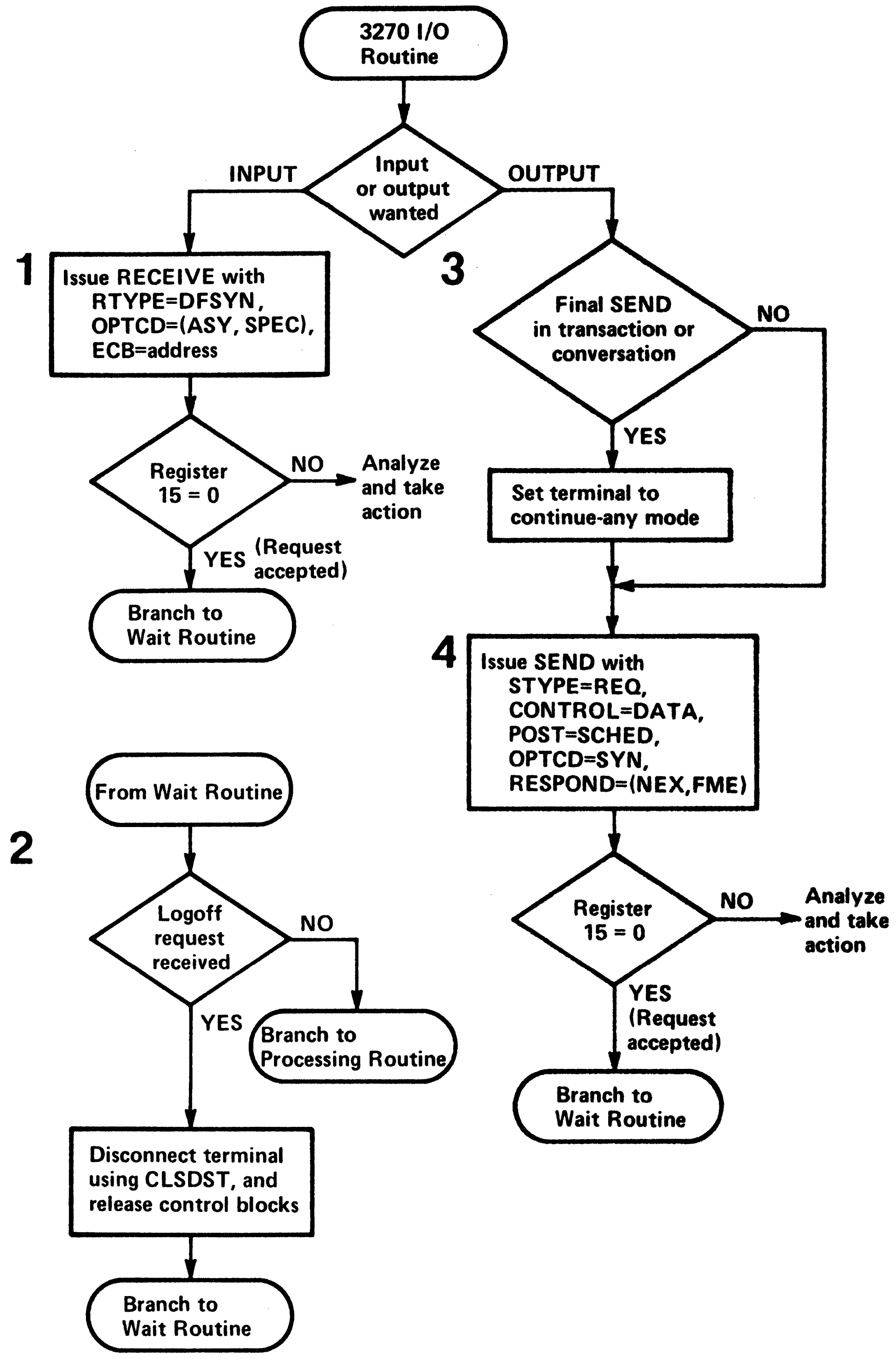

3270 I/O Routine
Input or output wanted
INPUT
OUTPUT
1
Issue RECEIVE with RTYPE=DFSYN, OPTCD=(ASY, SPEC), ECB=address
Register 15 = 0
NO
Analyze and take action
YES
(Request accepted)
Branch to Wait Routine
2
From Wait Routine
Logoff request received
NO
YES
Branch to Processing Routine
Disconnect terminal using CLSDST, and release control blocks
Branch to Wait Routine
3
Final SEND in transaction or conversation
NO
YES
Set terminal to continue-any mode
4
Issue SEND with STYPE=REQ, CONTROL=DATA, POST=SCHED, OPTCD=SYN, RESPOND=(NEX,FME)
Register 15 = 0
NO
Analyze and take action
YES
(Request accepted)
Branch to Wait Routine

Figure 12-5. The Logic of the 3270 I/O Routine

whether local or remote, must begin with a 3270 command character, indicating Erase, Erase and Write, or Erase All Unprotected; ACF/VTAM inserts an ESC character for BSC 3270. The 3270 is different from other logical units in several ways, including the following:

Since a 3270 does not contain an application program (a variable program), it cannot send commands or record-mode responses. However, in some cases, ACF/VTAM will provide responses to the ACF/VTAM application program on behalf of the 3270 as a result of receiving BSC responses to transmitted data or as a result of receiving indications that the 3270 is in a particular bracket state.

The amount of data that can be sent to or received from the 3270 is limited by the physical characteristics of the 3270, whereas the amount of data that can be sent to or received from a 3601 is more indefinite.

Chaining output to the 3270 is not possible.

Responses cannot be requested by 3270 terminals.

For other differences, see the considerations for "IBM 3270 Information Display System (Record-Mode)" in Appendix I of the *ACF/VTAM Macro Language Reference.*

The following notes are keyed to Figure 12-5.

1 Except that the type and length of data may be different for a 3270 RECEIVE, this request is similar to **1** discussed for the 3600 input routine.

2 This logic is similar to that of **2** for the 3600 input routine except that, because the 3270 cannot request a response to input it has provided, no check is made to determine whether to send a response.

3 If 3270 output is requested by a processing routine, the 3270 I/O routine determines whether this output completes a transaction or conversation. If it does, the 3270 terminal is put back into continue-any mode so that the RECEIVE (with OPTCD=ANY) specified in the RPL1 exit routine can receive input from this terminal when the terminal operator wishes to begin a new transaction or conversation.

4 A SEND macro instruction is issued to send the output. (Although not shown, this routine may also have to determine from the processing-routine request what 3270 command character—for example, Erase and Write—is to precede the output data stream that the processing routine furnishes.) The sending of the output is scheduled synchronously (POST=SCHED, OPTCD=SYN); ACF/VTAM returns control after it has scheduled the output operation. A response is requested (RESPOND=(NEX,FME)) so that the ACF/VTAM application program can determine whether or not the operation was successful. The 3270 returns information enabling ACF/VTAM to provide the appropriate response in the RPL and to schedule the RESP exit routine. The RESP exit routine (Figure 12-6) posts an ECB so that the main program's wait routine can determine that the operation completed, branching back to the processing routine that requested the output. (Note that an output request to a 3270 printer requires a definite response (RESPOND=(NEX,FME)); an output request to a display can specify either NEX or EX but cannot specify that neither a positive nor negative response is to be returned (RESPOND=(NEX,NFME)).) After successfully scheduling output to the 3270, the 3270 output routine branches to the wait routine.

The Logic of the RESP Exit Routine

Figure 12-6 shows the logic of the RESP exit routine. This routine is entered when the response is received to an output request that has POST=SCHED specified in a 3600 or 3270 output routine. The output operations have been scheduled with responses to be returned by the logical unit so that completion of each operation can be determined. (It is also possible for all output operations to be specified with POST=RESP. In this case, the response is received by ACF/VTAM and its nature determined by the ACF/VTAM application program after ECB posting or RPL exit routine scheduling. No RESP exit routine is required.)

Note that when the ACF/VTAM application program gets control in its RESP exit routine, a RECEIVE is not issued. The nature of the response is determined by examining the RESPOND and other fields of an RPL that is in ACF/VTAM's storage. The address of this RPL is in a parameter list whose address is in register 1 when the RESP exit routine is entered. The logical unit control area (the ECB, RPL, and logical unit control block) can be located by the address in the USER field of the ACF/VTAM RPL. (It contains whatever was placed in the USERFLD field of the NIB when the logical unit was connected.)

The following notes are keyed to Figure 12-6.

1 If the response is positive, the appropriate ECB is posted and a return is made to ACF/VTAM. Even if other action must be taken because the response is negative, the ECB is posted so that the wait routine will know that the operation has been completed.

2 The RESP exit routine can use the SYNAD exit routine to analyze a negative response; if so, the user could set up the appropriate registers and branch directly to the SYNAD exit routine.

3 If the situation is defined by the SYNAD exit routine to be recoverable, the operation is retried. If it is part of a chaining operation, it may be necessary to save the sequence number of the output segment to which a negative response was returned so that the chaining routine can determine the sequence number at which it starts a retry. If the logical unit's inbound sequence number (outbound from the host) must be reset, a SESSIONC using the STSN operand can be used to synchronize sequence numbers. (This logic can also be in the SYNAD exit routine.)

On completion, the RESP exit routine returns control to ACF/VTAM.

The Logic of the DFASY Exit Routine

Figure 12-7 shows the logic of the DFASY exit routine in Sample Program 2. The DFASY exit routine is entered when a request is received from the logical unit asking the program to quiesce (stop) sending to the logical unit or to resume sending, if sending was previously quiesced.

Quiescing can be done for two reasons:

- To ensure that, at a given time, only the logical unit or the ACF/VTAM application program can be sending. This use of quiescing is not demonstrated in this sample program. (Quiescing is only one means available to ensure that both sides do not send at the same time. Change-direction indicators may also be used. In many cases, receiving a response message ensures that both ends do not send at the same time.)

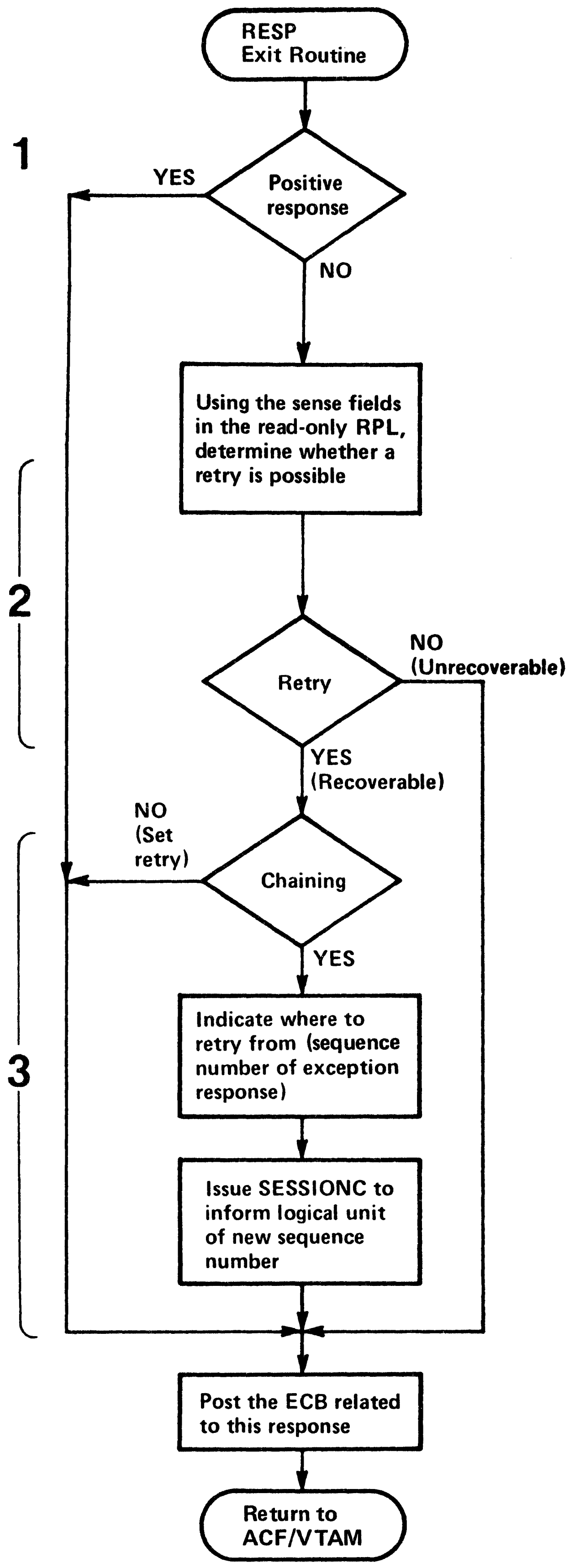
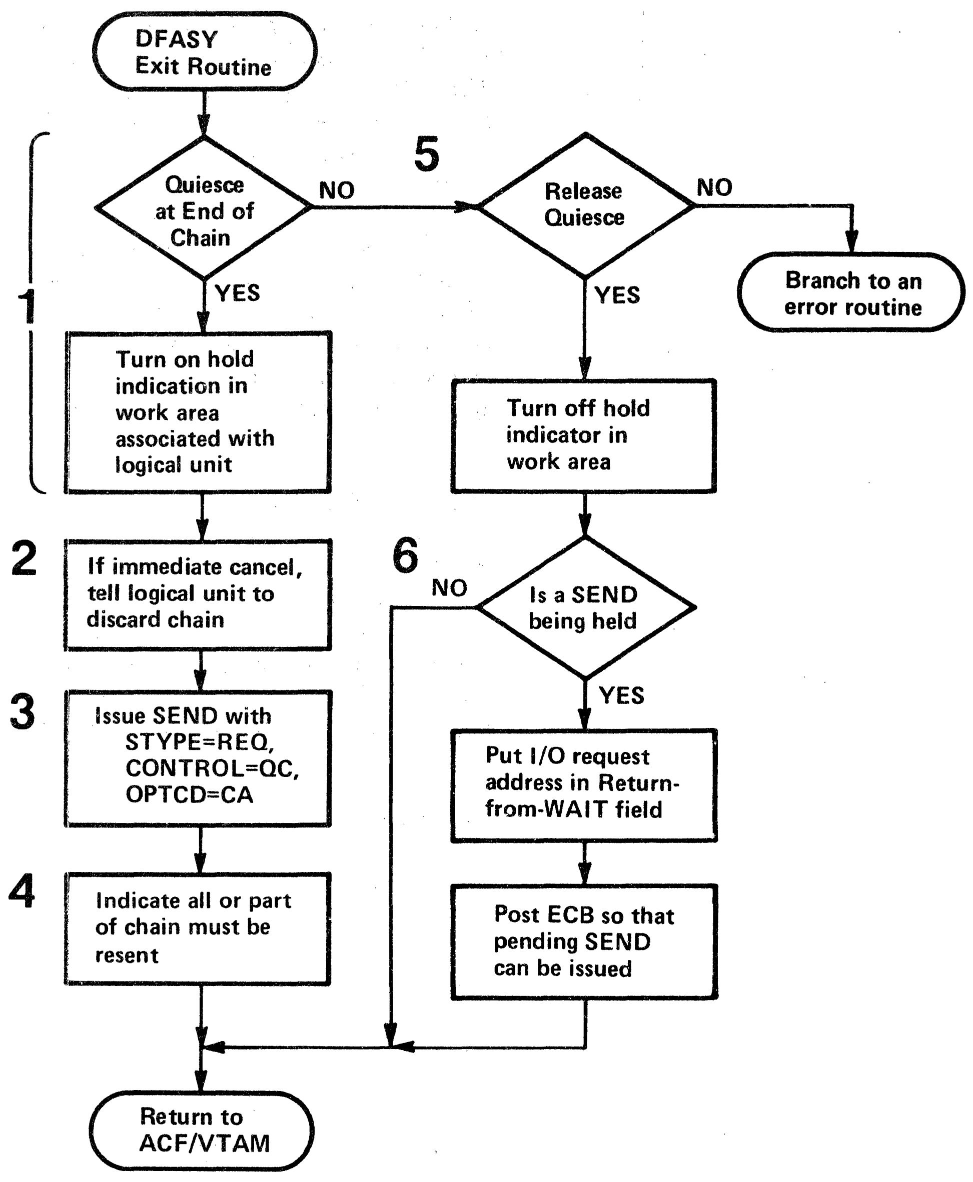

RESP Exit Routine
1
YES
Positive response
NO
Using the sense fields in the read-only RPL, determine whether a retry is possible
2
Retry
NO (Unrecoverable)
YES (Recoverable)
NO (Set retry)
Chaining
YES
3
Indicate where to retry from (sequence number of exception response)
Issue SESSIONC to inform logical unit of new sequence number
Post the ECB related to this response
Return to ACF/VTAM

Figure 12-6. The Logic of the RESP Exit Routine



DFASY Exit Routine
Quiesce at End of Chain
NO
YES
1
Turn on hold indication in work area associated with logical unit
2
If immediate cancel, tell logical unit to discard chain
3
Issue SEND with STYPE=REQ, CONTROL=QC, OPTCD=CA
4
Indicate all or part of chain must be resent
5
Release Quiesce
NO
YES
Branch to an error routine
Turn off hold indicator in work area
6
NO
Is a SEND being held
YES
Put I/O request address in Return-from-WAIT field
Post ECB so that pending SEND can be issued
Return to ACF/VTAM

Figure 12-7. The Logic of the DFASY Exit Routine

- To interrupt a steady flow of input data so that an output operation can be performed. This is the use of quiescing that is demonstrated here. For example, a teller at a 3600 terminal may wish to temporarily interrupt a long printout so that an informational message can be sent to the ACF/VTAM application program. As a result of a teller action, the 3601 logical unit for the teller's work station sends a Quiesce at End of Chain command to the ACF/VTAM application program, which can then agree to stop sending and be ready to read input from the logical unit.

The Quiesce at End of Chain and Release Quiesce commands are sent as expedited-flow commands unaccompanied by data. ACF/VTAM schedules the ACF/VTAM application program's DFASY exit routine when one of these commands is received.

The following notes are keyed to Figure 12-7.

1 The type of command that caused the DFASY exit routine to be entered is available in the CONTROL field of the read-only RPL whose address is provided by

ACF/VTAM on entry. If a Quiesce at End of Chain (QEC) command was received, this routine sets a do-not-send bit in the work area associated with the logical unit. The logical unit control block, as in the RESP exit routine, is located by the address in the USER field of the RPL. The QEC may be treated as an immediate quiesce rather than as a quiesce on the completion of sending the current chain; this is not shown here.

2 If the quiesce is to be immediate, the exit routine can instruct the logical unit to discard the chain by issuing a SEND macro that specifies CONTROL=CANCEL. Alternatively, the next SEND would be set to CHAIN=LAST; the logical unit determines whether or not to use the chain elements previously received. If in the middle of a chain and not all of the chain is to be resent, the ACF/VTAM application program can note where sending is to resume when the quiesce condition is released.

3 The QEC command is acknowledged by sending back a Quiesce Complete (QC) command. The command is specified symbolically (CONTROL=QC) by the ACF/VTAM application program. So that the logical unit's input will be able to complete the request to receive input from any terminal, being recurrently issued in the RPL1 exit routine, the logical unit is put back into continue-any mode (OPTCD=CA). In sending with CONTROL specified other than DATA, a response should not be specified; ACF/VTAM does not post the operation until it receives a response (assuming that, POST=RESP and RESPOND=(NEX,FME) were specified).

4 This flag may be required in addition to the do-not-send flag to determine where to resume sending. See 2 above.

5 If the command is a Release Quiesce (RELQ) command, the do-not-send flag is turned off.

6 If further output is being held, the output routine is rescheduled for this logical unit, and an ECB is posted so that the wait routine will branch to it. Control is returned to ACF/VTAM.

Appendix A. Communicating with BSC and Start-Stop Terminals

The ACF/VTAM user communicating with logical units can also communicate with BSC terminals, start-stop terminals, and (optionally) local 3270 terminals, using either BTAM or ACF/VTAM basic-mode macro instructions. In this appendix, the term *terminal* is used for a BSC device, a start-stop device, or a BSC or local 3270 (used in basic mode), while the term *logical unit* is used for the entity (program or device) that is communicated with in ACF/VTAM record mode.

Figure A-1 shows the communication mode (record or basic) that can or must be used for the various types of devices that can be attached to an ACF/VTAM network.

Using BTAM

Figure A-2 shows that communication through BTAM can be combined with communication through ACF/VTAM in a number of ways:

An application program that uses ACF/VTAM record-mode macro instructions to communicate with SNA logical units can also include BTAM macro instructions to communicate with BSC, start-stop, and local 3270 terminals.

An application program that uses ACF/VTAM record-mode macro instructions to communicate with SNA logical units can use ACF/VTAM basic-mode macro instructions to communicate with some BSC terminals, start-stop terminals, and local 3270 terminals, and can also use BTAM macro instructions to communicate with other BSC, start-stop, and local 3270 terminals.

ACF/VTAM and BTAM application programs can be used separately.

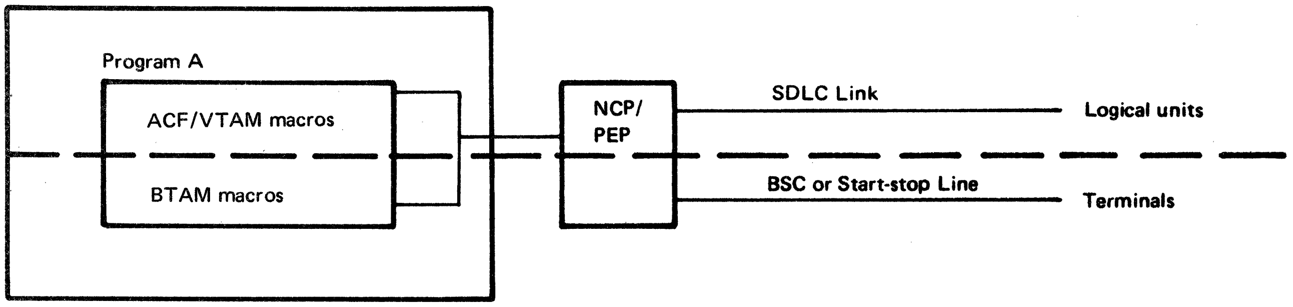
A user who wanted to continue to use terminals not supported by ACF/VTAM must use one of these combinations. However, even if all BSC and start-stop terminals in a network are supported by ACF/VTAM, the user can use BTAM (instead of ACF/VTAM) to communicate with them.

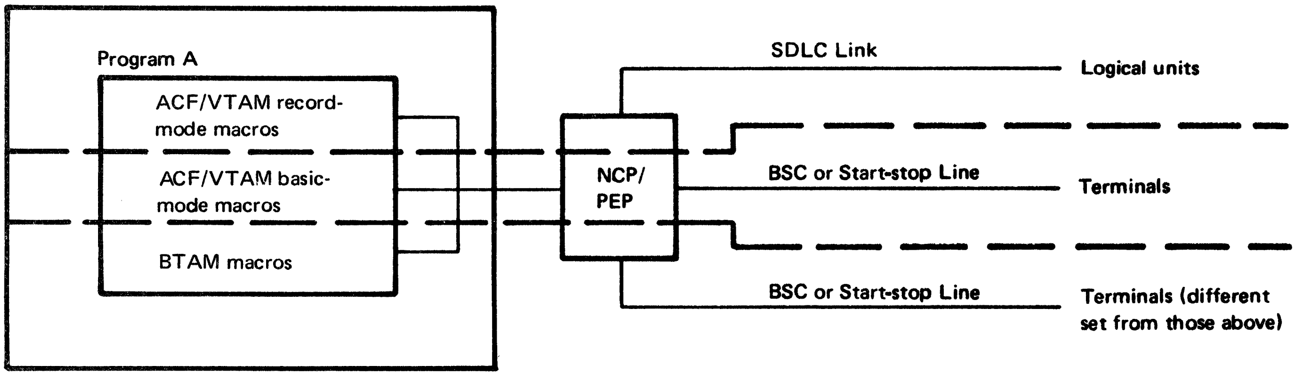
Using ACF/VTAM

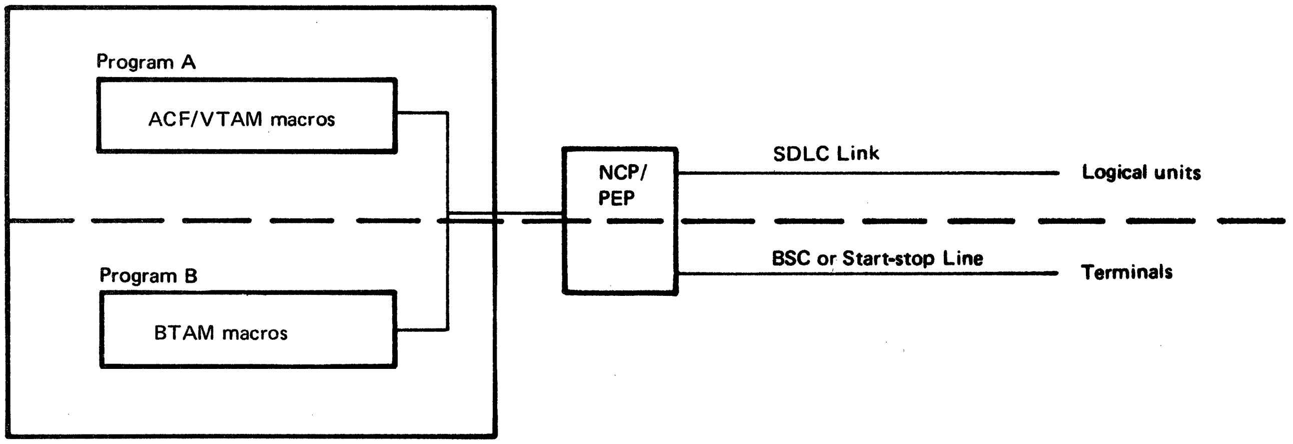
As an alternative to using BTAM macro instructions, a set of ACF/VTAM macro instructions is provided for communicating with certain BSC, start-stop, and local 3270 terminals. (Supported terminals are listed in *ACF/VTAM Concepts and Planning.*) This set consists of the basic-mode macro instructions—READ, WRITE, SOLICIT, and others—described in this appendix. These basic-mode macro instructions contrast with the record-mode macro instructions described in Chapter 6.

Must use record mode	Application program in session with another application program Local (channel-attachment) SNA devices (for example, 3790) Remote SNA devices (on SDLC link) BSC 3270s defined with PU=YES
Can use either record mode or basic mode	BSC 3270s defined with PU=NO Local (channel-attached) 3270s
Must use basic mode	Start-stop terminals BSC terminals (except the BSC 3270 as shown above)

Figure A-1. Types of Devices and Modes (Record or Basic) Used for Their Sessions







Notes: Local and remote 3270 terminals can be communicated with using ACF/VTAM record-mode macro instructions, ACF/VTAM basic-mode macro instructions, or BTAM macro instructions. Using ACF/VTAM record mode allows logical units and 3270s to be communicated with using the same set of macro instructions.

Figure A-2. Using BTAM and ACF/VTAM to Communicate with BSC and Start-Stop Terminals

Distinguishing between Logical Units and BSC/Start-Stop Devices

When connecting a terminal whose identity and device type are unknown, ACF/VTAM, as the result of an OPNDST macro instruction, identifies the type of terminal (logical unit or BSC/start-stop terminal) by setting a value in the DEVCHAR field of the NIB furnished by the application program. ACF/VTAM also indicates in the DEVCHAR field whether record mode macro instructions or basic mode macro instructions (or either) can be used to communicate with the logical unit or terminal. By testing the values in the DEVCHAR field, the program can determine the appropriate macro instructions and related logic with which to communicate with the terminal.

The Basic-Mode Macro Instructions

Here are brief descriptions of the basic-mode macro instructions:

SOLICIT: Requests ACF/VTAM to solicit input from a specific BSC, start-stop, or local 3270 terminal or from a group of terminals. Input is read into ACF/VTAM buffers, not into the application program; a READ macro instruction is used to read the input from the ACF/VTAM buffers into the application program's data area. The solicitation for a group of terminals continues until input has been received from every terminal in the group. Once input has been received from a terminal, the terminal must be resolicited unless continuous solicitation was specified when the terminal was connected.

READ: Requests ACF/VTAM to transfer data from a specific BSC, start-stop, or local 3270 terminal or from any one of a group of terminals into an area in the application program. A request to read from any one of a group requires a SOLICIT prior to the READ; a request to read from a specific terminal causes solicitation of input to take place if a SOLICIT was not previously issued for that terminal. If data is already in an ACF/VTAM buffer as the result of a previous solicit or read operation, the transfer of data takes place immediately.

WRITE: Requests ACF/VTAM to transfer data from an application program to a specific BSC, start-stop, or local 3270 terminal. The application program can also request that ACF/VTAM send certain control information to a terminal (such as an Erase All Unprotected Fields command to a 3270 terminal) or that ACF/VTAM write conversationally (write and then read from a terminal).

DO: Requests ACF/VTAM to perform the I/O defined by an LDO control block in the application program.

RESET: Requests ACF/VTAM to cancel all outstanding I/O requests to a BSC, start-stop, or local 3270 terminal and, if necessary, to reset any error lock that may have been set for the terminal to prevent output.

LDO: Defines a particular kind of I/O operation that is not ordinarily performed, such as writing a positive response with leading graphics to a System/3 or System/370 CPU. The operation is requested by issuing a DO macro instruction that specifies the LDO.

CHANGE: Requests that the information furnished to ACF/VTAM as part of a connection request be changed. The macro instruction assumes that the information in the NIB, used to pass information when requesting connection, has been modified.

Basic-Mode Concepts and Facilities

Most of the concepts and facilities that have been discussed previously in this book apply to communication with BSC, start-stop, and local 3270 terminals as well as to communication with logical units. This appendix supplements the information provided previously in this book (most of Chapter 6, however, does not apply to BSC terminals, start-stop terminals or local 3270 terminals (used in basic mode)). The following concepts and facilities apply to both the basic mode and the record mode:

Connection

Overlapping ACF/VTAM requests with other processing

Application program exit routines (the DFASY, SCIP, and RESP exit routines apply only to communication in record mode)

Error notification

Specific-mode and any-mode

Continue-any and continue-specific modes

Terminal identification (see "Identifying Logical Units" in Chapter 6)

Handling overlength input data

The concepts and facilities described in the following sections are those that are used only for basic-mode communication.

Data Blocks

The unit of data exchanged in record-mode operations is different from that exchanged in basic-mode operations. In record mode, the units exchanged are messages (which includes data) and responses. In basic mode, the unit of data is a *block.*

Blocks are delimited differently for different types of terminals. For start-stop terminals, a block ends with an EOB character; for BSC terminals, a block ends with an ETB character.

Although the application program can solicit more than one block from a terminal, a READ macro instruction can move only one block into the application program's input area (or less, if the input area is smaller than a block). An output operation (a WRITE macro instruction) always sends one block to the terminal.

Solicitation

When the application program solicits data from a terminal, ACF/VTAM initiates whatever actions (such as polling or line preparation) are required to obtain data from the terminal and put it into ACF/VTAM buffers.

READ requests issued in the specific-mode cause solicitation to take place if no previously solicited data is in ACF/VTAM's buffers, and then cause the data to be moved into the application program's I/O storage area. In contrast, READ requests issued in the any-mode can only move solicited data from ACF/VTAM's buffers into the application program's input area. The user of READ requests in the any-mode must therefore explicitly request solicitation. Figure A-3 illustrates both implicit and explicit soliciting of data.

Specific-mode and any-mode are also used when data is solicited. In the specific-mode, data is solicited only from a single terminal. In the any-mode, data is solicited from all connected terminals.

An application program might use these forms of solicitation in the following manner:

1. The application program initially uses the SOLICIT macro instruction to solicit data from all of the terminals to which it has become connected.
2. The application program then issues a READ in the any-mode, which is completed when one of the terminals responds to the solicitation.

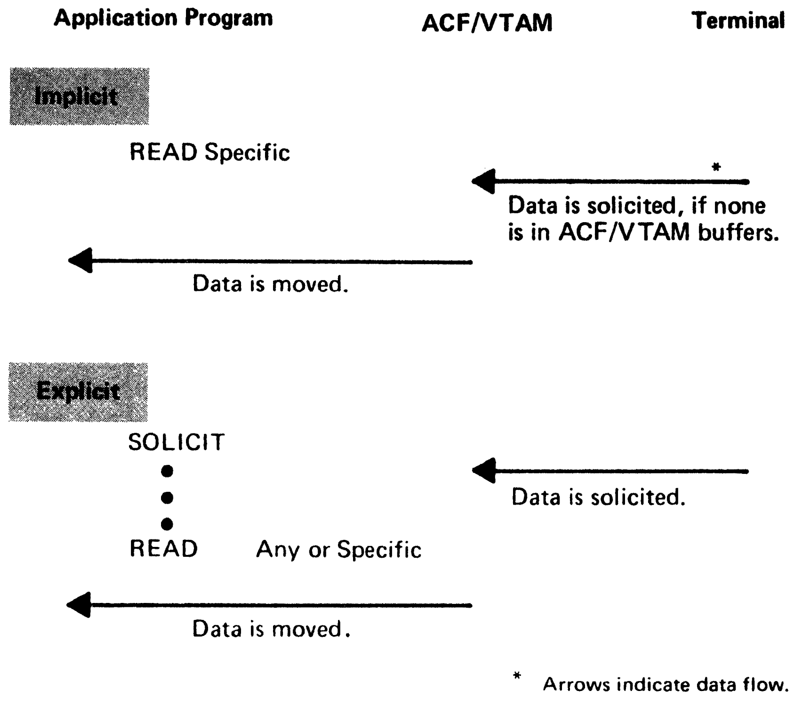


Figure A-3. Implicit and Explicit Solicitation Using Basic Mode

3. The application program communicates with the terminal using WRITE and READ macro instructions issued in the specific-mode. The READ macro instructions cause implicit solicitation to occur.
4. When the transaction is completed, the application program issues a new SOLICIT macro instruction directed specifically at the terminal, so that a new READ issued in the any-mode will be satisfied when the next transaction begins.

When connection is established with a terminal in the basic mode, the application program indicates the amount of data that each solicit request (implicit or explicit) is to obtain from that terminal. It is the application program's responsibility to determine when a new solicit request should be issued.

The application program can designate that, for each solicit request, ACF/VTAM is to:

- Solicit only a *block* of data from the terminal. For start-stop terminals, a block ends with an EOB character; for BSC terminals, a block ends with an ETB character.
- Solicit a *message* from the terminal. Messages do not apply to start-stop terminals; for BSC terminals, a message ends with an ETX character. Messages consist of one or more blocks. (A BSC "message" is not to be confused with the more general "message" that is exchanged with logical units.)
- Solicit a *transmission* from the terminal. For both start-stop and BSC terminals, a transmission ends with an EOT character. A transmission consists of one or more messages (for start-stop terminals, one or more blocks).
- Solicit the terminal *continuously* until the application program cancels the solicitation.

Soliciting Blocks

When data is solicited one block at a time and an error occurs during transmission, only a limited amount of data need be recovered by the terminal. However, since the application program must frequently reissue a solicit request (to acknowledge the previous block and obtain a new one), data throughput over the communications line is reduced. Block solicitation is appropriate when an unusually high number of line errors is expected and when the length of retransmitted data must be kept to a minimum, even if at the expense of slower response times and poorer line utilization. The installation must authorize, in the application's APPL definition statement, the solicitation of blocks.

Soliciting Messages and Transmissions

The lengths of messages and transmissions are not as closely dependent on the type of terminal as are block lengths. Message and transmission lengths are usually established by the terminal's operator and the nature of the application. The lengths of messages and transmissions from a remote job-entry station, for example, are determined by the number of cards in each job deck and the number of job decks available for transmission at one time.

Since messages and transmissions tend to be much longer than blocks, message and transmission solicitation means more data has to be recovered when an I/O error is detected. However, with these forms of solicitation, data transmission is much more efficient, because the acknowledgments and resolicitations needed to obtain the blocks are performed by the communications controller, not the application program.

Message and transmission solicitation is appropriate for applications that require short response times but can tolerate lengthy transmissions when required.

The choice between message solicitation and transmission solicitation (which can be made only for BSC terminals) depends on how undesirable delays between messages would be. With transmission, delays between messages are minimized, although more data must be recovered if errors occur.

Continuous Solicitation

The advantages and disadvantages of continuous solicitations are the opposite of those of block solicitation. By soliciting continuously, the application program can obtain data with a minimum of programming. However, the application program must determine when solicitation should cease, and must explicitly tell ACF/VTAM when to do so. If the solicitation must be interrupted frequently, the efficiency is lost.

Continuous solicitation is appropriate for batch input applications, where transmissions are relatively frequent and delays between blocks, messages, and transmissions must be minimized.

Special I/O Operations

The application program can initiate the following I/O operations with one request:

- Copy a remotely attached 3277 Display Station's buffer into the buffer of any printer or display station attached to the same cluster control unit (COPYLBM or COPYLBT operation)
- Read the entire contents of any 3270 display station buffer (READBUF operation)
- Send a positive response with leading graphic characters to a System/3 or System/370 CPU and then read the terminal's next block of data (WRTPRLG and READ operations); or send a negative response with leading graphic characters to one of these terminals and then reread the block of data (WRTNRLG and READ operations)
- Write data beginning with a block of heading characters to a System/3 or System/370 CPU (WRTHDR and WRITE operations)
- Write data to a terminal from separate output data areas (gather-write) or read from a terminal into separate input data areas (scatter-read)

To use these facilities, the application program builds a set of *logical device orders* (LDOs). Each LDO indicates the specific type of I/O operation (such as COPYLBM or READBUF), the data area to be used, and an optional indicator that links the LDO to a following one. In both form and manner of use, LDOs resemble channel command word (CCW) programs. A set of LDOs is executed with a DO macro instruction.

By using LDOs, the application program can request I/O operations that are not available via the conventional macro instructions like READ and WRITE.

Special Processing Options

When connection is established with a terminal, the application program can designate a set of ground rules that ACF/VTAM is to follow during subsequent communication with that terminal. The extent of solicitation described above–block, message, transmission, or continuous–is one example. Other options, most of which relate to NCP processing, can be selected by the application program (some options are not available for all types of terminals):

- ACF/VTAM can treat the receipt of leading graphic characters as either a normal condition or as an error condition. These options are called the LGIN and LGOUT options. (The names of these options, like those which follow, are the names coded as part of the PROC operand of the terminal's NIB.)
- The application program can allow the communications controller to insert idle device-control characters into output data, or it can prevent the insertion of these charactars (TMFLL option).

  If the communications controller is prepared to receive intermediate transmission blocks (ITBs) from a terminal, the application program can allow the communications controller to insert an error information byte (EIB) into each block, or it can prevent the insertion of EIBs (EIB option). The application program can use the EIBs to perform error recovery (retries) on a subblock basis, rather than on a block basis.
- The application program can override any text time-out limitation that the communications controller might otherwise use with the terminal (TIMEOUT option).
- The application program can prevent the communications controller from employing error recovery procedures if an error is detected during output to the terminal, during input from the terminal, or during either input or output (ERPIN and ERPOUT options).
- For some start-stop terminals, the application program can determine whether the communications controller is to monitor the terminal for attention interruptions and whether it is to notify the application program when the attention interruption is detected (MONITOR option). ACF/VTAM notifies the application program by scheduling its ATTN exit routine. (These are attention interruptions detected when the application program is not communicating with the terminal; attention interruptions that occur *during* an I/O operation are always brought to the attention of the application program by an RPL return code.)
- The application program can insert its own line-control characters into output data, or it can allow ACF/VTAM to do so (ELC option).
- The application program can send all data to the terminal in transparent text mode (BINARY option).

Unless the application program issues a request to change the ground rules, they remain in effect as long as the terminal is connected.

Using the Basic-Mode Macro Instructions

This section describes ways in which the basic-mode macro instructions can be used to connect and communicate with BSC and start-stop terminals.

Connecting BSC and Start-Stop Terminals

See Chapter 5, "Connecting and Disconnecting Logical Units." In addition, these options must be specified in the NIB used when a terminal is connected:

Options Set by PROC: The processing options in a NIB determine the attributes to be applied to the terminal represented by that NIB. These options are dependent on each terminal's specific characteristics. These options are defined with the PROC operand of the NIB macro instruction before a terminal is connected.

These options can be changed after connection by using the MODCB macro instruction to modify the PROC operand of the NIB and then issuing the CHANGE macro instruction to make the change permanent.

TRUNC or KEEP specifies how ACF/VTAM will handle an incoming message that is too large for a specified input area. TRUNC specifies that the input will be truncated to fit the area. KEEP specifies that any input that does not fit into the area will be held for a subsequent read request. KEEP is assumed if neither is chosen.

BLOCK, MSG, TRANS, or CONT specifies the amount of read-ahead to be used with the terminal. If BLOCK, MSG, or TRANS is coded, the SOLICIT macro instruction solicits a block, message, or transmission. If CONT is coded, SOLICIT continues to solicit data from the terminal until a RESET macro instruction is issued to stop the read-ahead operation. TRANS is assumed if no option is chosen. A program must be authorized to specify PROC=BLOCK.

MONITOR or NMONITOR specifies whether ACF/VTAM is to detect attentions generated by the terminal. If an ATTN exit routine is coded, it is invoked each time the terminal generates an attention. NMONITOR is assumed if neither is chosen.

Modifying Terminal Characteristics

The CHANGE macro instruction modifies the characteristics of a terminal that has already been connected by a program. CHANGE is used to change the processing options, mode, and user data associated with the terminal.

Before a terminal is connected, it is defined in a NIB. The mode (MODE=BASIC), any processing options (PROC), and any user data (USERFLD) are specified. When the OPNDST is issued, these characteristics are put into ACF/VTAM's internal tables, and the NIB is not used again for that terminal.

To change these characteristics, CHANGE can be used. A new NIB is built or the same NIB used for connection is reused. The symbolic name of the terminal must be in the NIB. Then, using MODCB, the PROC, USERFLD, or MODE field of the NIB is set. The RPL referred to by the CHANGE macro instruction must point to the modified NIB. When the CHANGE macro is issued, ACF/VTAM's internal tables are modified.

For example, if a terminal is connected to ACB1, using NIB1 as shown here:

```
NIB1      NIB       NAME=TERM1,MODE=BASIC,PROC=MSG
```

and you want to change MSG to BLOCK, use MODCB:

```
          MODCB     AM=VTAM,NIB=NIB1,PROC=BLOCK
```

The RPL refers to NIB1:

```
RPL1      RPL       AM=VTAM,ACB=ACB1,NIB=NIB1
```

Issue the CHANGE macro to effect the change:

```
          CHANGE    RPL=RPL1
```

Reading Data

There are three ways in which data can be read from a terminal:

A READ can be issued to a specific terminal, which causes ACF/VTAM to solicit and read data from the terminal, and pass it to your program.

A SOLICIT can be issued to initiate input from a specific terminal connected to your program; a READ can be issued to read data from any solicited terminal that has sent data in response to the SOLICIT.

A conversational WRITE can be issued as discussed later in "Writing Data."

READ SPEC

Using this method, the program issues READ macros directed to specific terminals. Before the READ is issued, the ARG field of the accompanying RPL must contain the CID of the terminal. Because ACF/VTAM puts the CID into the ARG field of the RPL when the terminal is connected, the connection RPL can be used for issuing the READ. For example, if a terminal has just been connected using the RPL whose address is in register 5, the READ macro can also use that RPL:

```
           READ     RPL=(5),OPTCD=SPEC
             .
             .
             .
RPL1       RPL      ACB=ACB1,AM=VTAM,AREA=AREA1,AREALEN=100
AREA1      DS       CL100
```

This example reads one block of input from the terminal whose CID is currently in the RPL. The data is read into AREA1. When the READ is completed, the RECLEN field of the RPL will contain the actual length of the data read in. Note that each READ reads only one block of data.

To use ACF/VTAM's read-ahead facility, PROC=TRANS or PROC=MSG should be set in the NIB when the terminal is connected. This means that the first time a READ SPEC is issued to a terminal, ACF/VTAM reads blocks until a message or transmission has been read. A block is received each time READ is issued to that terminal. After each READ, TESTCB can be used to inspect the RPL's feedback information to see if the last block of the message or transmission was read. (See Chapter 8, "Manipulating Control Blocks," for how to use TESTCB.)

```
LOOP       READ     RPL=(5),OPTCD=SPEC
             .
             .
             .
  (Process input block.)
             .
             .
             .
           TESTCB   AM=VTAM,RPL=(5),DATAFLG=EOT
           BNE      LOOP
             .
             .
             .
*SAMPLE RPL AND I/O AREA
RPL1       RPL      ACB=ACB1,AM=VTAM,AREA=AREA1,AREALEN=100
AREA1      DS       CL100
```

In this example, each input block is processed as it is read so that a single input area can be reused. To process a complete message or transmission at one time, multiple data areas are required.

One input area can be defined to be large enough to accommodate a message or transmission. This example assumes that input will be no longer than 500 bytes. After each READ, a pointer can be updated to point to the next position in the input area.

```
          LA      5,AREA1
LOOP      READ    RPL=(6),AREA=(5)
          TESTCB  AM=VTAM,RPL=(6),DATAFLG=EOT
          BE      OUT
          SHOWCB  AM=VTAM,RPL=(6),
                  FIELDS=RECLEN,AREA=INCR,
                  LENGTH=4
          L       7,INCR
          AR      5,7
          B       LOOP
OUT       (Process input data.)
          .
          .
          .
*SAMPLE RPL AND I/O AREA
RPL1      RPL     ACB=ACB1,AM=VTAM,OPTCD=SPEC,AREA=AREA1,
                  AREALEN=100
AREA1     DS      CL500
INCR      DS      F
```

The total input area (AREA1) should be large enough to handle the longest message or transmission, or the area can be filled, the data processed, and the same area reused to finish the message or transmission.

Here is another example. In this example, messages are read asynchronously from three terminals. First, here are some assumptions:

- When each terminal is connected, the USERFLD of the NIB is set to contain the address of a data area to be used only with that terminal.

  ```
  NIB1      NIB ...USERFLD=A(AREA1)
  NIB2      NIB ...USERFLD=A(AREA2)
  NIB3      NIB ...USERFLD=A(AREA3)
  ```

- A system WAIT macro is used to await completion of all READ requests. Since the WAIT macro is implemented differently for DOS/VS and OS/VS, it is shown only as WAIT in this example. And, while the ECBs are shown, the system-dependent means of constructing an ECB list are not shown here.
- TESTCB is used to inspect the FDBK field of the RPL to see if a message has been completely received.
- The program communicates with three terminals in message mode; the maximum length of a message is 500 bytes.

```
          READ    RPL=RPL1
          READ    RPL=RPL2
          READ    RPL=RPL3
WAIT1     WAIT
* GET ADDRESS OF POSTED ECB INTO REGISTER 1
* ZERO ECB FOR NEXT WAIT
```

```
         LA      2,4(1)
         CHECK   RPL=(2)
         TESTCB  AM=VTAM,RPL=(2),DATAFLG=EOM
         BE      LAST
         SHOWCB  AM=VTAM,RPL=(2),FIELDS=(AREA,RECLEN),
                 AREA=AREA0,LENGTH=8
         L       4,AREA0
         L       5,AREA00
         AR      4,5
         SHOWCB  AM=VTAM,RPL=(2),FIELDS=USER,AREA=AREA0,
                 LENGTH=4
         L       6,AREA0
         A       6,=A(400)
         CR      6,4
         BL      OVERFLOW
         READ    RPL=(2),AREA=(4)
         B       WAIT1
LAST     SHOWCB  AM=VTAM,RPL=(2),FIELDS=USER,AREA=AREA0,
                 LENGTH=4
         L       6,AREA0
  (The data area address is now in register 6. Process the input message.)
         READ    RPL=(2),AREA=(6)
         .
         .
         .
AREA0    DS      CL4  FOR SHOWCB
AREA00   DS      CL4  FOR SHOWCB
ECB1     DC      A(0)
RPL1     RPL     ACB=ACB1,AM=VTAM,OPTCD=(SPEC,ASY),ECB=ECB1,
                 AREA=AREA1,AREALEN=500
AREA1    DS      CL500
ECB2     DC      A(0)
RPL2     RPL     ACB=ACB1,AM=VTAM,OPTCD=(SPEC,ASY),ECB=ECB2,
                 AREA=AREA2,AREALEN=500
AREA2    DS      CL500
ECB3     DC      A(0)
RPL3     RPL     ACB=ACB1,AM=VTAM,OPTCD=(SPEC,ASY),ECB=ECB3,
                 AREA=AREA2,AREALEN=500
AREA3    DS      CL500
         END
```

Here is a brief explanation of this example:

- The program issues three READs, one to each terminal. Each terminal has been connected with PROC=MSG, and with its USERFLD pointing to its data area. Whenever a READ is completed, its accompanying RPL will contain the address of that data area in the RPL's USER field; this will always be the address of the *beginning* of the data area.
- The program then waits for any READ to be completed. The coding for the WAIT macro and its accompanying ECB list depends on the system being used; it is not shown here. When any ECB is posted, the WAIT is completed. The address of the posted ECB is obtained and, because each ECB immediately precedes its RPL, 4 is added to it to get the address of the RPL.
- After issuing a CHECK to test for errors, TESTCB is used to see if the last block of a message was read. If so, a branch is made to process the message and reinitiate read-ahead to the terminal. If the message is not completed, the next block is read from the terminal.

- Using SHOWCB, the current data area address is obtained from the RPL, as well as the length of the last input block. They are added and the AREA operand is reset to the next available location in the data area, ensuring that there is enough room to read the next block. (The example does not show the processing if there is no more room.)
- To test for overflow, SHOWCB is used to get the address of the beginning of the data area from the USER field of the RPL. If 400 is added to this, there should be 100 bytes left in the area; enough for another block. If the current data area pointer (the AREA field of the RPL) shows less than 100 bytes left, there may not be enough room to read another block.
- If there are no overflow problems, another READ is issued to get the next block and a branch is made back to WAIT to await the next block from any terminal.
- When beginning to process the input data (after the message has been read), the address in the RPL's USER field is used to determine the beginning of the data area for that terminal. (The AREA address in the RPL was changed as each block was read. It no longer points to the beginning of the data area.)

READ ANY

The READ SPEC method in the previous example is useful for a fixed number of terminals. The READ ANY method provides a way of communicating with a varying number of terminals.

There are two steps in using READ ANY. First, SOLICIT macros are issued to have ACF/VTAM poll the terminals connected to the program. Second, a READ ANY macro is issued to read one block of data from any terminal that sent data in response to the SOLICIT. Here is a simplified example:

```
              SOLICIT  RPL=RPL1,OPTCD=ANY
              READ     RPL=RPL1,OPTCD=ANY
              .
              .
              .
RPL1          RPL      ACB=ACB1,AM=VTAM,AREA=AREA1,AREALEN=100
```

A loop can be used to reexecute the READ ANY.

READ ANY can be used in a program that accepts logons. A LOGON exit routine is used to process each logon. After connecting a terminal, a SOLICIT SPEC is issued for that terminal. The main program issues READ ANY macros to read data obtained by any of the SOLICIT requests. A READ ANY may be issued before terminals are connected; it becomes effective when input is available. Here is a simplified example:

```
        Main Program                       LOGON Exit

        OPEN       ACB1
        SETLOGON   OPTCD=START
        .                                  OPNDST   RPL=RPL0,NIB=NIB0,
        .                                           OPTCD=(ACCEPT,ANY)
        .                                  .
LOOP    READ       RPL=RPL1,               .
                   OPTCD=ANY               SOLICIT  RPL=RPL0,OPTCD=SPEC
        (Branch to process input           BR       14
        and respond.)
        SOLICIT    RPL=RPL1,               .
                   OPTCD=SPEC              .
        B          LOOP                    .
        .                                  RPL0   RPL  ACB=ACB1,AM=VTAM
        .                                  NIB0   NIB  MODE=BASIC
        .
```

```
RPL1     RPL      ACB=ACB1,
                  AM=VTAM,
                  AREA=AREA1,
                  AREALEN=100
AREA1    DS       CL100
         END
```

If PROC=MSG or PROC=TRANS is set for the terminal, the READ ANY can be followed with enough specific READs to get the entire message or transmission. Or the READ ANY loop can continue, but the data is processed only when a complete message or transmission has been received from a terminal. The example under READ SPEC illustrates this procedure.

With some further modification, this sample program can service three terminals concurrently, and many terminals over a period of time. A LOGON exit routine is used (not shown below, but it would be the same as that in the previous example) to connect the terminals and solicit input. This example adds a new exit routine, an RPL exit routine. This routine, whose address is coded in an RPL, is scheduled when any READ ANY is completed. It processes the input data and initiates another input request. (This routine can also write replies; for this logon, see "Writing Data" below.) Here is the example:

```
MAINP    OPEN     ACB1
         .
         .
         READ     RPL=RPL1,OPTCD=ANY
         READ     RPL=RPL2,OPTCD=ANY
         READ     RPL=RPL3,OPTCD=ANY
         .
         .
         .
EX1      BALR     3,0
         USING    *,3
         ST       14,SAVE1
         LA       13,SAVE2
         CHECK    RPL=(1)
         (Branch to process the input.)
         TESTCB   AM=VTAM,RPL=(1),DATAFLG=EOT
         BE       REPLY
READA    READ     REPL=(1)
         L        14,SAVE1
         BR       14
REPLY    (Write the reply; then resolicit and reissue a READ ANY.)
         .
         .
         .
RPL1     RPL      ACB=ACB1,AM=VTAM,OPTCD=ASY,EXIT=EX1,
                  AREA=AREA1,AREALEN=100
RPL2     RPL      ACB=ACB1,AM=VTAM,OPTCD=ASY,EXIT=EX1,
                  AREA=AREA2,AREALEN=100
RPL3     RPL      ACB=ACB1,AM=VTAM,OPTCD=ASY,EXIT=EX1,
                  AREA=AREA3,AREALEN=100
SAVE1    DS       F
SAVE2    DS       18F
```

After each terminal is connected in the LOGON exit routine, it is solicited. Then as each of the first three terminals responds with data, one of the READ ANYs in the main program is completed. When any READ ANY is completed, the RPL exit routine (EX1) is scheduled. EX1 checks and processes the input block and, using TESTCB, determines whether this is the last block of a transmission. If not, EX1 issues another READ ANY that, when completed, causes EX1 to be scheduled again. If the input block is the last of a transmission, EX1 prepares a response to the terminal. An example shown later under "Writing Data" uses another RPL exit routine to resolicit the terminal when the WRITE is completed.

Multiple-Block Processing: When using READ ANY macros to read the first block of a message or transmission, the remaining blocks can be read with READ SPEC macros. Since data from a terminal will normally satisfy an outstanding READ ANY as well as a READ SPEC, it must be specified that subsequent data from a terminal is to satisfy only a READ SPEC macro directed to that terminal. The CA and CS option codes in an RPL control this:

CA specifies that data from a terminal will satisfy either a READ ANY or a READ SPEC macro.

CS specifies that data from a terminal will satisfy *only* a READ SPEC macro.

When CA or CS is specified in a macro, it applies to the subsequent READ request.

OPTCD=CA (the assumed value) can be specified in the RPL when a terminal is connected. Any data from that terminal will satisfy a READ ANY macro. When the READ ANY macro is issued, OPTCD=CS is specified:

```
READ      RPL=(2),OPTCD=(ANY,CS)
```

Any subsequent blocks from the terminal can now be read only with a READ SPEC macro. As each block is read, it is tested for end-of-message or end-of-transmission:

```
TESTCB    AM=VTAM,RPL=(2),DATAFLG=EOT
```

If a full message or transmission has not been read, READ SPEC macros are issued–OPTCD=(SPEC,CS)–to read the remainder. When all the data has been read, OPTCD=CA is respecified and the terminal is resolicited.

Writing Data

This section discusses how to use ACF/VTAM to transmit the output data stream to a terminal. It shows how to write simple one-block messages, how to send multiple-block messages, and how to write conversationally.

Simple Writes

Many output requests will probably be simple, one-block answers to input data received from terminals. For example, an input block might be read and processed, and then a short answer prepared using the same data area. Using the RPL that was used for input, the reply is sent.

```
          READ      RPL=(5)
          (Branch to process input data and build a 50-byte data stream.)
          WRITE     RPL=(5),RECLEN=50
          .
          .
          .
*SAMPLE RPL
RPL1      RPL       ACB=ACB1,AM=VTAM,AREA=AREA1,AREALEN=100
```

When the terminal was connected (again, using RPL1), ACF/VTAM put the CID into the ARG field of the RPL. Since the same RPL is then used for both the READ and the WRITE, the CID is maintained to indicate the terminal's address. And, since block mode is used, the output mode OPTCD=LBT is assumed: after each output block is sent, ACF/VTAM provides an EOB or ETX. When the terminal acknowledges receipt of the data, ACF/VTAM sends the terminal an EOT.

Still using block mode, the WRITE can be made asynchronous. An RPL exit routine is scheduled whenever the WRITE is completed. In the exit routine, the terminal is resolicited or another READ is issued.

```
         READ     RPL=(5)
         (Branch to process input and prepare a response.)
         WRITE    RPL=(5),EXIT=EX2,RECLEN=50,OPTCD=ASY
         .
         .
         .
RPL1     RPL      ACB=ACB1,AM=VTAM,AREA=AREA1,AREALEN=100
         .
         .
         .
EX2      BALR     3,0   ENTRY PROCEDURE
         USING    *,3
         ST       14,SAVE1
         LA       13,SAVE2
         LR       2,1
         CHECK    RPL=(2)
         READ     RPL=(2),EXIT=...
         .
         .
         .
         L        14,SAVE1
         BR       14
SAVE1    DS       F
SAVE2    DS       18F
```

Here is an example with the READ ANY processing (see "Reading Data") that uses a LOGON exit routine (not shown) and two RPL exit routines, one for READs and one for WRITEs.

```
MAINP    OPEN     ACB1
         .
         .
         .
         READ     RPL=RPL1,OPTCD=(ANY,ASY),EXIT=EX1
         READ     RPL=RPL2,OPTCD=(ANY,ASY),EXIT=EX1
         READ     RPL=RPL3,OPTCD=(ANY,ASY),EXIT=EX1
         .
         .
         .
EX1      BALR     3,0
         USING    *,3
         ST       14,SAVE1
         LA       13,SAVE2
         LR       2,1
         CHECK    RPL=(2)
         (Branch to process input.)
         TESTCB   AM=VTAM,RPL=(2),DATAFLG=EOT
         BE       WRITE1
READ1    READ     RPL=(2),EXIT=EX1,OPTCD=ANY
         L        14,SAVE1
         BR       14
         .
         .
         .
WRITE1   (Branch to prepare reply.)
         .
         .
         .
         WRITE    RPL=(2),RECLEN=50,EXIT=EX2
         L        14,SAVE1
         BR       14
EX2      BALR     3,0
         USING    *,3
         ST       14,SAVE1
         LA       13,SAVE2
         LR       2,1
         CHECK    RPL=(2)
         SOLICIT  RPL=(2),OPTCD=(SYN,SPEC)
         READ     RPL=(2),OPTCD=(ANY,ASY),EXIT=EX1
         L        14,SAVE1
         BR       14
RPL1     RPL      ACB=ACB1,AM=VTAM,AREA=AREA1,AREALEN=100
RPL2     RPL      ACB=ACB1,AM=VTAM,AREA=AREA2,AREALEN=100
RPL3     RPL      ACB=ACB1,AM=VTAM,AREA=AREA3,AREALEN=100
AREA1    DS       CL100
AREA2    DS       CL100
AREA3    DS       CL100
SAVE1    DS       F
SAVE2    DS       18F
```

Multiple Writes

A program may have to send multiple-block messages or transmissions in response to inquiries received from terminals or, for example, a request for a large report from a data base. The entire message can be put into an output area and written one block at a time, or each block could be transmitted immediately after it was built. In either case, the last block is written with OPTCD=LBT in the RPL; the preceding blocks are written with OPTCD=BLK or LBM in the RPL. Here is a simplified example:

```
FIRST    (Build one block of output in AREA1.)
         .
         .
         .
         (Last block in message or transmission ?)
NO       WRITE    RPL=(5),OPTCD=BLK
         CHECK    RPL=(5)
         B        FIRST
YES      WRITE    RPL=(5),OPTCD=LBT
         CHECK    RPL=(5)
```

Conversational Write

Requests for input and output can be combined in a single macro. The conversational option of the WRITE macro is used to write a block of data to a terminal and then read a block from the same terminal. The conversational WRITE uses the AREA field of the RPL to contain the address of the output data and the AAREA field to contain the address of the input data. Here is an example:

```
         WRITE   RPL=RPL1
         .
         .
         .
RPL1     RPL     ACB=ACB1,AM=VTAM,OPTCD=(CONV,LBT),
                 AREA=AREA1,RECLEN=12,
                 AAREA=AREA2,AAREALN=100
AREA1    DC      CL100'GOOD MORNING'
AREA2    DS      CL100
```

The program writes a GOOD MORNING message from AREA1; ACF/VTAM then reads one block from the same terminal into AREA2.

This method works well with an acquired list of terminals. After connection, a conversational WRITE is issued to each terminal. If block mode and short responses are used, this method can be used throughout the program.

The following expands this example. Assume that three terminals have been connected using three different RPLs. Each terminal uses the RPL with which connection was made, and a system WAIT macro is used to await completion of any outstanding WRITE macro. When any conversational WRITE is completed, the input is processed and another conversational WRITE issued. If message or transmission mode is used, the conversational WRITE is followed with READ SPECs to read in a complete message or transmission. Here is the example:

```
         WRITE   RPL=RPL1
         WRITE   RPL=RPL2
         WRITE   RPL=RPL3
WAIT1    WAIT
         (Put the address of the posted ECB into register 1.)
         LA      2,4(1)
         CHECK   RPL=(2)
         (Branch to process input block.
         Issue READ SPECs if using message or transmission mode.
         Prepare reply in area field.
         Zero ECB for next WAIT.)
         WRITE   RPL=(2),RECLEN=100
         B       WAIT1
         .
         .
         .
ECB1     DC      A(0)
RPL1     RPL     ACB=ACB1,AM=VTAM,AREA=AREA1,RECLEN=12,
                 AAREA=AAREA1,AAREALN=100,ECB=ECB1,
                 OPTCD=(CONV,ASY)
AREA1    DC      CL100'GOOD MORNING'
AAREA1   DS      CL100
ECB2     DC      A(0)
RPL2     RPL     ACB=ACB1,AM=VTAM,AREA=AREA2,RECLEN=12,
                 AAREA=AAREA2,AAREALN=100,ECB=ECB2,
                 OPTCD=(CONV,ASY)
```

```
AREA2    DC     CL100'GOOD MORNING'
AAREA2   DS     CL100
ECB3     DC     A(0)
RPL3     RPL    ACB=ACB1,AM=VTAM,AREA=AREA3,RECLEN=12,
                AAREA=AAREA3,AAREALN=100,ECB=ECB3,
                OPTCD=(CONV,ASY)
AREA3    DC     CL100'GOOD MORNING'
AAREA3   DS     CL100
         END
```

Canceling Data-Transfer Requests

Active or pending data-transfer requests are canceled using the RESET macro. RESET can also be used to reset an error lock for a terminal.

If, for example, a series of WRITE macros is issued to a terminal, and, when the first WRITE is completed, the RPL FDBK2 field indicates that the terminal has been disconnected, RESET can be issued to cancel the remaining WRITEs. Or, if a SYNAD or LERAD exit is taken after an I/O operation, RESET can be used to turn off the error lock that ACF/VTAM sets when an I/O error occurs.

To use RESET, an RPL that is not currently active is used. This means that a request cannot be canceled by using the RPL that was used for the request being canceled. In some exit routines, an RPL might be set aside (or obtained from a pool) to be used only for RESET.

Handling Attentions

Terminal users operating with 1050s or 2741s can generate attentions with the attention key or break feature of their terminals as soon as an NCP session is in progress. An attention usually means that the user wants to "get the attention of" the program to enter some special comment or request. If ACF/VTAM detects an attention while the program is reading from or writing to a terminal, the FDBK field of the data-transfer RPL will indicate that an attention was detected. If working with terminals that can generate attentions, the FDBK field should be checked after each data-transfer request. If an attention is detected while not communicating with the terminal, there is no RPL in which to set a flag. For these instances, ACF/VTAM schedules the program's ATTN exit routine. (If there isn't one, ACF/VTAM ignores the attention.)

When an attention is detected and there is an ATTN exit routine, ACF/VTAM sets a lock so that further communication with that terminal is not possible until the lock is reset. Data-transfer requests issued to a "locked" terminal are queued until the lock is reset. The ATTN exit routine decides whether to reset the lock. At the same time, any data-transfer requests that have been queued since the lock was set can be canceled. The RESET macro can be used to do both. RESET OPTCD=LOCK resets the lock; RESET OPTCD=UNCOND both resets the lock and cancels queued requests for the terminal.

The ATTN exit routine is entered with register 1 pointing to a 3-word parameter list:

Word 1 contains the address of the ACB to which the terminal is connected.

Word 2 contains the terminal's CID.

Word 3 contains the terminal's USERFLD data.

In the ATTN exit routine, the CID can be loaded into a register, the lock reset and a READ SPEC issued to see what the terminal wants. Here is an example:

```
ATTN1   BALR    3,0
        USING   *,3
        ST      14,SAVE1
        LA      13,SAVE2
        L       2,4(1)      GET CID
        RESET   RPL=RPL1,OPTCD=(UNCOND,SYN),ARG=(2)
        READ    RPL=RPL1,OPTCD=(SPEC,SYN),ARG=(2)
        CHECK   RPL=RPL1
        .
        .
        L       14,SAVE1
        BR      14
RPL1    RPL     ACB=ACB1,AM=VTAM,AREA=AREA1,AREALEN=100
AREA1   DS      CL100
SAVE1   DS      F
SAVE2   DS      18F
```

Handling Release Requests

If a program is connected to a terminal and another program wants to use that terminal, the other program is notified that the terminal is unavailable. If there is a RELREQ exit routine in your program, it is scheduled when another program requests connection to one of your program's terminals. In the RELREQ exit routine, a CLSDST macro is issued to disconnect the terminal and make it available for the requesting program.

When a RELREQ exit routine is entered, register 1 points to a 2-word parameter list:

Word 1 contains the address of the ACB to which the terminal is connected.

Word 2 contains the address of the terminal's symbolic name.

The RELREQ exit routine may release the terminal. First it is necessary to be sure that there is no pending data-transfer request for the terminal. In the main program, a terminal-status flag can be set in a terminal's status save area (whose address is in the terminal's user data) indicating whether the terminal is currently being solicited. This information is used in the RELREQ exit routine.

The RELREQ exit routine:

- Converts the terminal's symbolic name to a CID for use with the RESET macro.
- Issues a conditional RESET to cancel any pending SOLICIT request for the terminal. On return from RESET, the USER field of the RPL points to the terminal's status save area. Now, the status save area is tested to see if the terminal is being solicited.
- If the terminal was being solicited, the terminal is disconnected and control returned to the main program.
- If the terminal is not being solicited, the RESET may have canceled a pending WRITE macro. The main program will handle this condition when the WRITE completion code indicates that it was canceled by a RESET. The main program can reissue the WRITE, and then disconnect the terminal.
- If the terminal is not being solicited, or if the RESET failed, a switch is set indicating to the main program that another program has requested use of the terminal, but has not yet been disconnected.

```
RREQ1    BALR     3,0
         USING    *,3
         ST       14,SAVE1
         LA       13,SAVE2
         L        4,4(1)
         MODCB    AM=VTAM,NIB=NIB1,                 GET THE NAME
                  NAME=(*,4(4))                     INTO THE NIB
         INQUIRE  RPL=RPL1,OPTCD=CIDXLATE,          CONVERT NAME
                  NIB=NIB1,AREA=CIDAREA             INTO CID
         L        4,CIDAREA
         RESET    RPL=RPL1,OPTCD=COND,ARG=(4)
         LTR      15,15
         BNZ      SSW
         (Examine the terminal status save area where address is in the USER field
         to see if the terminal is being solicited.
         Is the terminal being solicited?
         Yes, issue CLSDST.
         No, go to SSW.)
         CLSDST   RPL=RPL1,ARG=(4)
         B        OUT
         .
         .
         .
SSW      (Set a switch to indicate that the terminal is requested; the main
         program should release it.)
         .
         .
         .
OUT      L        14,SAVE1
         BR       14
SAVE1    DS       F
SAVE2    DS       18F
CIDAREA  DS       F
NIB1     NIB
RPL1     RPL      AM=VTAM,OPTCD=SYN
```

A situation to guard against is when one program tries to acquire a list of connected terminals and has OPTCD=CONANY in his connection request. He wants connection to only one of the terminals. Suppose all terminals on the list are already connected to other programs having RELREQ exit routines. Further, suppose that each program chooses to give up a requested terminal. In this case, only one terminal is connected to the requesting program; the rest are not longer connected to any program; they have been released, but not reconnected. The program can attempt to reconnect the terminal after releasing it by coding OPTCD=NQ in the RPL used for connection. If the terminal was not connected to the requesting program, the releasing program will get it back. If it was connected, the OPNDST will be completed with a terminal-unavailable return code. The OPTCD=NQ operand indicates that no RELREQ exit routine is to be scheduled to get this terminal; if it is busy, it is not to be taken away.

Basic-Mode Sample Programs

Here are three basic-mode sample programs. These sample programs are designed to illustrate the principles of basic-mode communication rather than to be complete programs that a particular installation would want to use.

Basic-Mode Sample Program 1: LOGON Exit, RPL Exit, and READ ANY

This sample program illustrates the use of a LOGON exit routine to connect terminals as they log on. The program does not know which terminals will log on, so it accepts logons from an infinite number of terminals. As terminals begin to log on, the program communicates with three terminals in parallel. As each terminal is satisfied, another is serviced on a rotating basis. Although only three terminals are handled at a time, all get their chance to communicate with the main program. Each time a terminal logs on, the LOGON exit routine connects and solicits data from the terminal; it then reads from and writes to each terminal. The overall program flow is shown in Figure A-4.

The main program opens an ACB to link itself to ACF/VTAM and to indicate that logons are to be queued. The SETLOGON macro is used to initiate queuing of logons. Then the program issues three read requests (with OPTCD=ANY): each read request reads one block of data from any one terminal that has been connected and that has responded to a solicit request. These three read requests cannot be completed until the LOGON exit routine has connected and solicited data from terminals. The main program then waits for a specific ECB to be posted. The user's program, in the READ exit routine, posts this ECB when it *wants* to terminate. It then closes the ACB (disconnecting the terminals) and returns control.

The main program uses five exit routines:

LOGON1 receives control when a terminal logs onto the program. This routine connects the terminal and solicits data from the connected terminal.

READ1 receives control when a READ is completed. This routine processes input data and writes an appropriate response.

WRITE1 receives control when a WRITE (in READ1) is completed. This routine resolicits the terminal to which the response has just been written.

LERAD1 abnormally terminates on logical errors.

SYNAD1 disconnects a failing terminal and returns control to the program on physical errors.

```
* SIMPLE INQUIRY PROGRAM
* WITH LOGON (REAL OR SIMULATED)
* WITH MULTIPLE READ ANYS AND RPL EXITS
* INITIALIZATION
SAMPLE1  CSECT
         SAVE     (14,12)        SAVE REGISTERS
         BALR     3,0            ESTABLISH BASE
BASE     EQU      *              GLOBAL
                                 ADDRESSABILITY
                                 POINT
         USING    *,3            ESTABLISH ADDRESSING
         ST       3,BASESAV      SAVE
                                 GLOBAL
                                 ADDRESSABILITY
                                 POINT
         ST       13,SAVE0+4     SAVE SAVE AREA PTR
         LA       13,SAVE0       OUR SAVE AREA
         SR       15,15          CLEAR REGISTER 15
         OPEN     ACB0           CONNECT WITH ACF/VTAM
         LTR      15,15          TEST FOR ERRORS
         BZ       OPENOK
         ABEND                   IF BAD OPEN
```

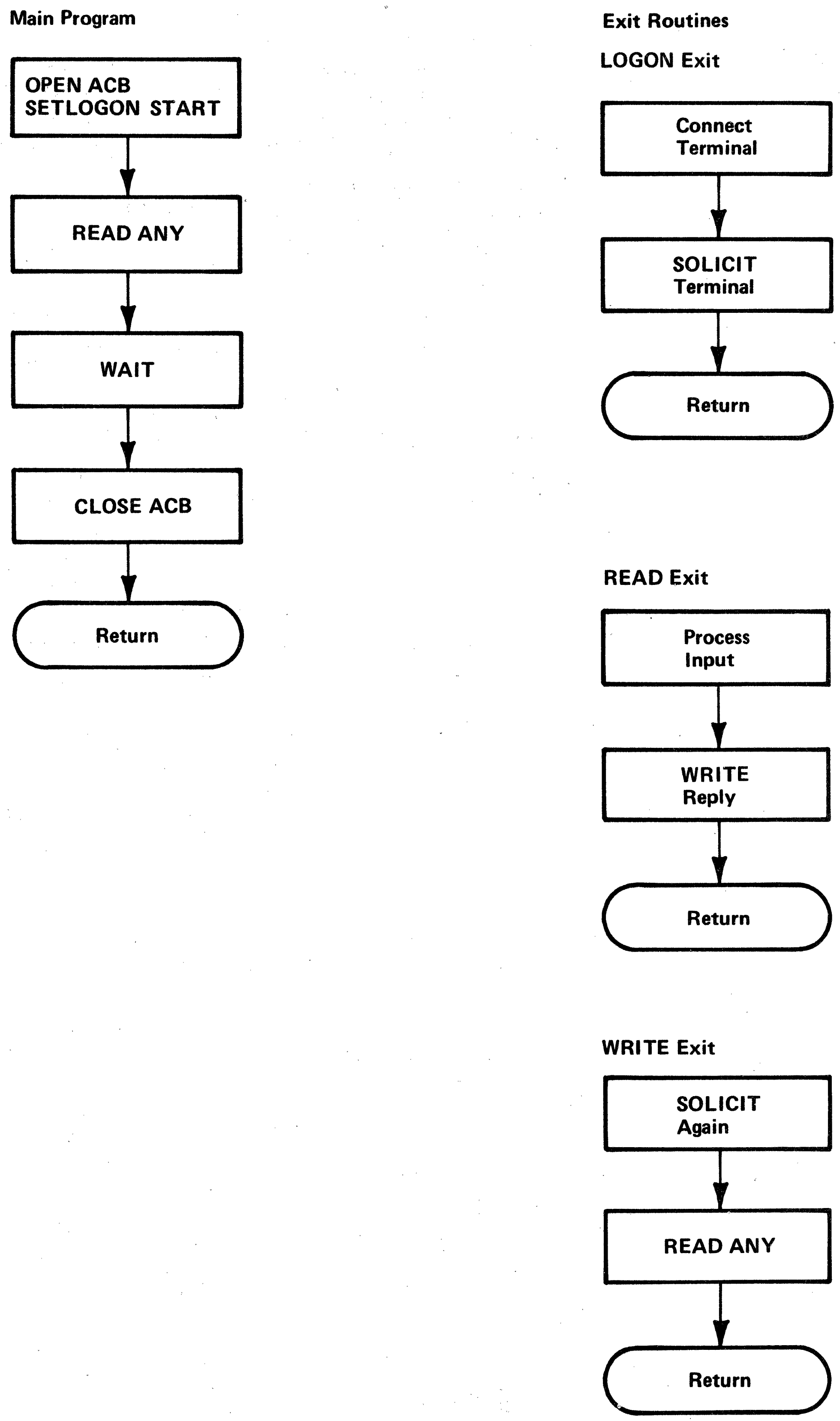


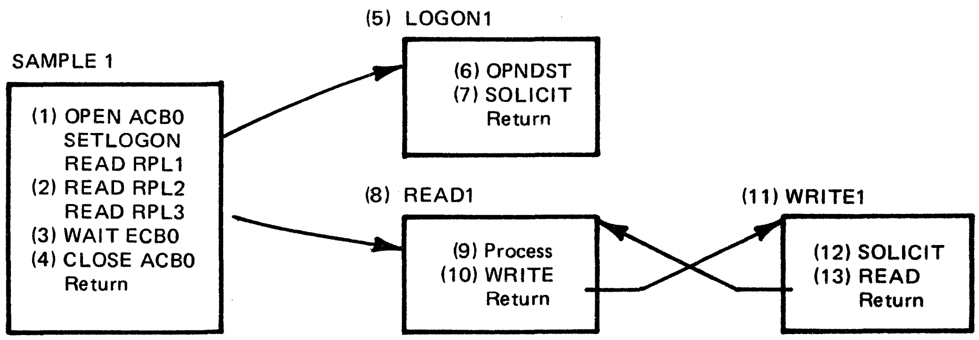
Figure A-4. The Logic of Basic-Mode Sample Program 1

```
OPENOK   EQU        *
         SETLOGON   RPL=RPL0,OPTCD=START
         READ       RPL=RPL1          READ FROM ANY TERMINAL
         READ       RPL=RPL2          READ FROM ANY TERMINAL
         READ       RPL=RPL3          READ FROM ANY TERMINAL
* WAIT FOR CLOSE TIME
* ECB0  SHOULD BE POSTED BY PROCESSING RTN WHEN APPROPRIATE
         WAIT       ECB=ECB0
* CLOSE ROUTINE
         CLOSE      ACB0              ACF/VTAM CLOSE
         L          13,SAVE0+4        GET OS SAVE AREA PTR
         RETURN     (14,12)
* EXLST/LOGON EXIT
* WILL BE ENTERED WHEN TERMINAL LOGS ON
LOGON1   USING      *,15              TEMPORARY ADDRESSABLE SPACE
         L          3,BASESAV         ESTABLISH GLOBAL
                                      ADDRESSABILITY BASE
         USING      BASE,3            ESTABLISH ADDRESSABILITY
         ST         14,SAVE1          SAVE RETURN ADDRESS
         LA         13,SAVE2          OUR SAVE AREA
* HERE COULD LOOK AT TERM NAME, TERM TYPE, AND LOGON MSG
         OPNDST     RPL=RPL0,         ALLOCATE TERMINAL
                    OPTCD=(ACCEPT,
                    ANY),             WHICHEVER LOGGED ON
                    NIB=NIB0          NIB ADDRESS
         SOLICIT    RPL=RPL0,         START INPUT FROM TERMINAL
                    OPTCD=SPEC        THE ONE JUST ALLOCATED
         L          14,SAVE1          RETURN ADDRESS
         BR         14                TO ACF/VTAM
* ERROR ANALYSIS ROUTINES
* WHEN THE PROGRAM IS DEBUGGED
* ENOUGH TO DO PRODUCTION WORK
* REPLACE THE FOLLOWING ABEND
* WITH A DUMP AND CONTINUE
LERAD1   ABEND                        ABEND ON LOGICAL ERRORS
SYNAD1   USING      *,15              TEMPORARY ADDRESSABLE SPACE
         L          3,BASESAV         ESTABLISH GLOBAL
                                      ADDRESSABILITY BASE
         USING      BASE,3            ESTABLISH ADDRESSABILITY
* GET STORAGE FOR SAVE AREAS
         ST         13,SAVE214        SAVE OLD SAVE AREA ADDR
         ST         14,SAVE1
         LA         13,SAVE2          LOAD NEW SAVE AREA ADDR
         CLSDST     RPL=(1)           DISCONNECT TERMINAL
         L          13,4(13)          RESTORE OLD SAVE AREA ADDR
         L          14,SAVE1          RETURN ADDRESS
* RELEASE STORAGE FOR SAVE AREAS
         BR         14                TO ACF/VTAM
* READ RPL/EXIT
* WILL BE ENTERED WHEN ANY READ COMPLETES
* REGISTER 1 POINTS TO RPL
```

```
READ1   USING     *,15            TEMPORARY ADDRESSABLE SPACE
        L         3,BASESAV       ESTABLISH GLOBAL
                                  ADDRESSABILITY BASE
        USING     BASE,3          ESTABLISH ADDRESSABILITY
        ST        14,SAVE1        SAVE RETURN ADDRESS
        LA        13,SAVE2        OUR SAVE AREA
        LR        2,1
        CHECK     RPL=(2)         FOR ERRORS ON READ
* RPL HAS CID OF TERM AND INPUT DATA ADDR AND LENGTH
* CAN DO PROCESSING OF INPUT MESSAGE HERE
* AND PREPARE AN OUTPUT MSG AND PUT LENGTH OF MSG IN RPL
        WRITE     RPL=(2),        REUSE SAME RPL
                  EXIT=WRITE1     WRITE COMPLETE EXIT
        L         14,SAVE1        RETURN ADDRESS
        BR        14              TO ACF/VTAM
* REGISTER 1 POINTS TO RPL
* WRITE RPL/EXIT
* WILL BE ENTERED WHEN ANY WRITE IS COMPLETE
WRITE1  USING     *,15            TEMPORARY ADDRESSABLE SPACE
        L         3,BASESAV       ESTABLISH GLOBAL
                                  ADDRESSABILITY BASE
        USING     BASE,3          ESTABLISH ADDRESSABILITY
        ST        14,SAVE1        SAVE RETURN ADDRESS
        LA        13,SAVE2        OUR SAVE AREA
        LR        2,1
        CHECK     RPL=(2)         FOR ERRORS ON WRITE
        SOLICIT   RPL=(2),        REUSE SAME RPL
                  OPTCD=(SYN,     WAIT FOR ACCEPTANCE
                  SPEC)           START INPUT FROM SAME TERMINAL
        READ      RPL=(2),        REUSE SAME RPL
                  EXIT=READ1,     READ COMPLETE EXIT
                  OPTCD=(ASY,     DON'T WAIT HERE
                  ANY)            READ FROM ANY SOLICITED TERM
        L         14,SAVE1        RETURN ADDRESS
        BR        14              TO ACF/VTAM
* CONSTANTS AND WORK AREAS
SAVE0   DS        18F             FOR MAINLINE MACROS
* ESTABLISH DSECT FOR LABELS SAVE1 AND SAVE 2
SAVE1   DS        F               FOR EXIT RETURN ADDRESS
SAVE2   DS        18F             FOR EXIT RTN MACROS
* CONSTANTS RESUME HERE
ACB0    ACB       AM=VTAM,        ACB TYPE
                  APPLID=APID,    PGM IDENTIFIER ADDRESS
                  MACRF=LOGON,    TO ACCEPT LOGONS
                  EXLST=EXLST0    EXIT LIST PTR
APID    DC        X'04'           LENGTH BYTE
        DC        CL4'PGM2'       APPLICATION ID
EXLST0  EXLST     AM=VTAM,
                  LOGON=LOGON1,   LOGON EXIT
                  SYNAD=SYNAD1,   I/O ERROR RTN
                  LERAD=LERAD1    LOGICAL ERROR RTN
NIB0    NIB       PROC=BLOCK,     READ TO BLOCK
                  MODE=BASIC
```

```
RPL0     RPL   AM=VTAM,
               ACB=ACB0          OPNDST AND SOLICIT
RPL1     RPL   AM=VTAM,          ACCESS METHOD ID
               ACB=ACB0,         RPL FOR READ ANY TERM
               AREA=AREA1,       INPUT/OUTPUT AREA
               AREALEN=100,      AREA SIZE
               EXIT=READ1,       READ COMPLETE EXIT
               OPTCD=(ASY,       NO WAIT INCLUDED
               ANY)              READ FROM ANY TERMINAL
RPL2     RPL   ACB=ACB0,         RPL FOR READ FROM ANY TERM
               AM=VTAM,
               AREA=AREA2,       INPUT/OUTPUT AREA
               AREALEN=100,      AREA SIZE
               EXIT=READ1,       READ COMPLETE EXIT
               OPTCD=(ASY,       NO WAIT INCLUDED
               ANY)              READ FROM ANY TERMINAL
RPL3     RPL   AM=VTAM,          ACCESS METHOD ID
               ACB=ACB0,         RPL FOR READ ANY TERM
               AREA=AREA3,       INPUT/OUTPUT AREA
               AREALEN=100,      AREA SIZE
               EXIT=READ1,       READ COMPLETE EXIT
               OPTCD=(ASY,       NO WAIT INCLUDED
               ANY)              READ FROM ANY TERMINAL
AREA1    DS    100C
AREA2    DS    100C
AREA3    DS    100C
BASESAV  DS    F
ECB0     DC    A(0)
         END
```

The following diagram shows the key operations in the main program and in the exit routines. Use the numbers in parentheses to find descriptive text following the diagram.



1. The program opens ACB0 to initiate queuing of logons, which will be handled by the LOGON1 routine. ACB0 specifies that an exit list, labeled EXLST0, will be used.

   EXLST0 specifies three exit routines:

LOGON1	Processes logon requests.
SYNAD1	Handles synchronous errors.
LERAD1	Handles logical errors.

   When ACB0 is opened and SETLOGON is issued, ACF/VTAM begins queuing logons to be handled by LOGON1 (Step 5).

2. The program issues three asynchronous READ macros with OPTCD=ANY. Each READ uses a different RPL (RPL1, RPL2, RPL3).

   Each RPL specifies an I/O area and an exit (READ1) to be taken whenever one of the read operations is complete.

   None of the read operations will be completed until a terminal is connected and solicited by LOGON1.
3. The program waits until ECB0 is posted. This ECB will be posted at some time during the execution of the program.
4. When this wait is over, the program closes ACB0 which disconnects all terminals. The program gives up control.
5. LOGON1 is the LOGON exit routine, which is entered to process each logon.
6. LOGON1 uses RPL0 to issue an OPNDST to connect whichever terminal logged on to the program. RPL0 points to NIB0, which is blank. When the OPNDST is completed, ACF/VTAM puts the symbolic name and CID of the connected terminal into NIB0, and puts the CID into the ARG field of RPL0. The NIB is only used for connection and is therefore free for another connection request each time the OPNDST is completed.
7. LOGON1 uses RPL0 to solicit data from the terminal just connected. Once a terminal responds to the SOLICIT, a pending READ request will be completed.
8. READ1 is invoked each time a READ operation is completed. READ1 first issues a CHECK to see if an error occurred on the READ. If so, LERAD1 processes logical errors, and SYNAD1 processes synchronous errors.
9. READ1 next processes the data brought in by the completed READ. When entered, READ1 receives the address of the appropriate RPL in register 1. The RPL contains the CID of the terminal from which data was read. READ1 then prepares a response to the terminal.
10. READ1 writes a response to the same terminal, using the RPL whose address was passed in register 1. When this WRITE is completed, WRITE1 is invoked. Meanwhile, control is returned to the part of the program that was interrupted when the READ was completed.
11. WRITE1 is invoked when a WRITE operation (requested in READ1) is completed. On entry, register 1 contains the address of the RPL associated with the completed WRITE. WRITE1 first issues a CHECK to test for errors.
12. WRITE1 then uses SOLICIT to resolicit the terminal for more data. It uses the RPL whose address was passed in register 1. In this way, the terminal to which a message is written can immediately enter data.
13. WRITE1 then issues a READ with OPTCD=ANY to get another block of data into the program. Again, when this READ is completed, the READ1 routine is invoked.

Using the three READ macros in the main program and the one in the WRITE1 routine, this program continuously reads and responds to data entered from each of the connected terminals.

Basic-Mode Sample Program 2: ACQUIRE, SOLICIT, and RPL EXIT

Figure A-5 shows the general logic of basic-mode sample program 2. In this example, a series of synchronous OPNDST macros connect the terminals by acquiring them. There is one OPNDST and one NIB for each terminal. The USERFLD field of each NIB is set to contain the address of the RPL. This RPL address will be used later for output requests to the terminal. When data is received, the RPL for the READ ANY contains the address of

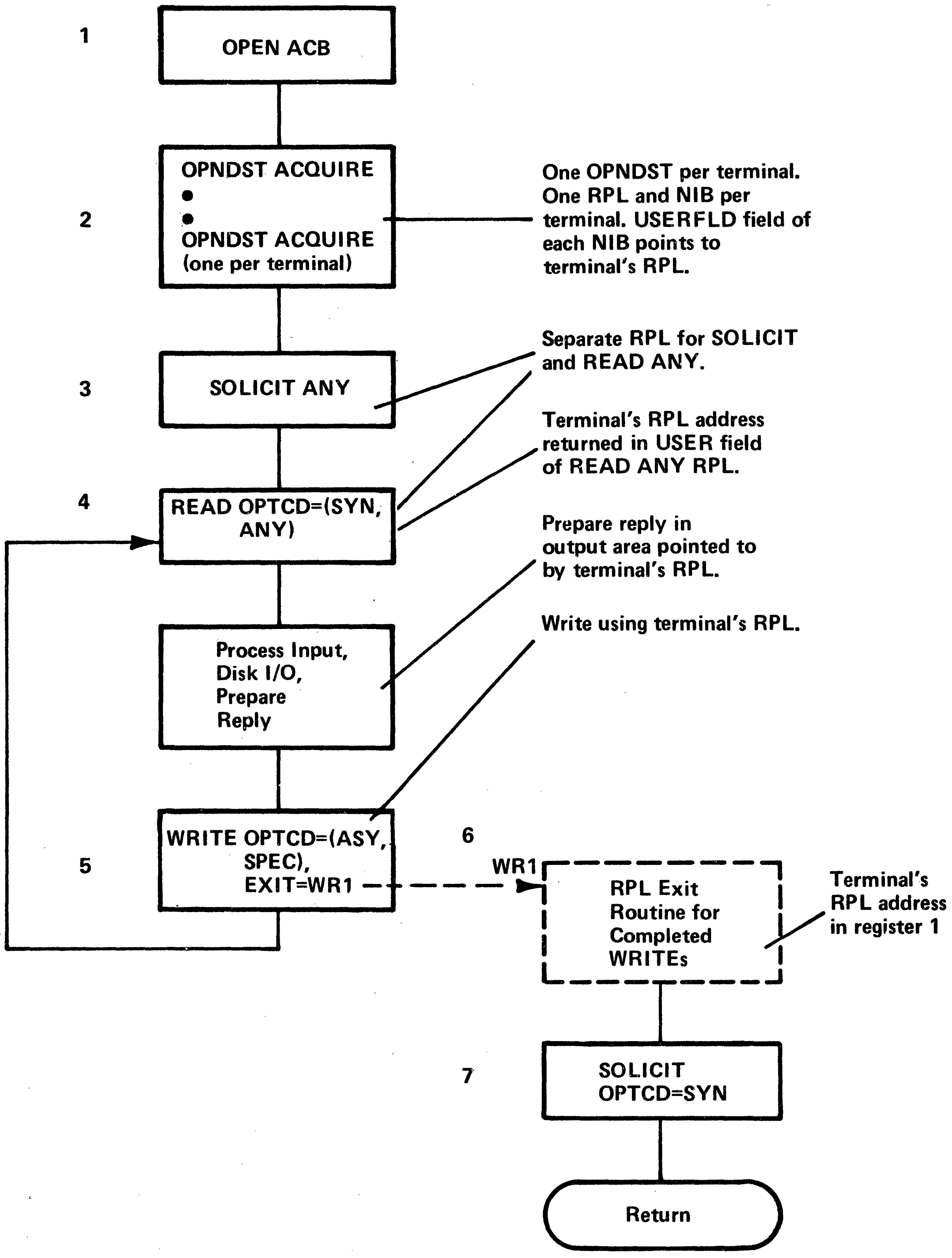
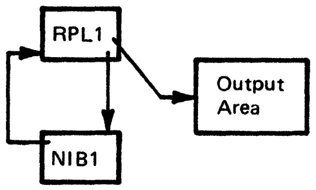

1
OPEN ACB
2
OPNDST ACQUIRE
●
●
OPNDST ACQUIRE
(one per terminal)
One OPNDST per terminal. One RPL and NIB per terminal. USERFLD field of each NIB points to terminal's RPL.
3
SOLICIT ANY
Separate RPL for SOLICIT and READ ANY.
Terminal's RPL address returned in USER field of READ ANY RPL.
4
READ OPTCD=(SYN, ANY)
Prepare reply in output area pointed to by terminal's RPL.
Process Input, Disk I/O, Prepare Reply
Write using terminal's RPL.
5
WRITE OPTCD=(ASY, SPEC), EXIT=WR1
6
WR1
RPL Exit Routine for Completed WRITEs
Terminal's RPL address in register 1
7
SOLICIT OPTCD=SYN
Return

Figure A-5. The Logic of Basic-Mode Sample Program 2

the terminal-dedicated RPL in its USER field. The input is processed, and a response is prepared in the output area whose address is contained in the terminal's RPL (the one used for connection).

The WRITE request is asynchronous. When it is completed, an RPL exit routine is scheduled. Meanwhile, a branch is taken back to the READ ANY to get the next block of input. When the WRITE is completed, the RPL exit routine is entered. Register 1 contains the address of the completed RPL. The same terminal is then solicited again.

In this example, a separate RPL is used to connect each terminal. The USERFLD field of the NIB for a specific terminal is set to contain the address of the RPL used to connect that terminal. The RPL is set to contain the address of an output area used only for that terminal.



The SOLICIT ANY and READ ANY macros use the same RPL to get input from all terminals. This RPL is used only for input requests. For output and connection requests, a different RPL is assigned to each terminal. When the READ ANY is completed, its RPL contains, in the USER field, the address of the terminal's RPL for the responding terminal. This happens because the terminal's RPL address was put in the USERFLD field of the NIB at connection. A response is prepared in the output area pointed to by the terminal's RPL. When the asynchronous WRITE is completed, the RPL exit routine is entered with the address of a terminal's RPL in register 1. This RPL address is used to resolicit the terminal.

Notes to Sample Program 2:

1. The ACB is opened to begin processing. MACRF=NLOGON specifies that logons are not to be accepted. A LOGON exit routine is not required.
2. An OPNDST is coded for each terminal to be connected (or a loop to reexecute a single OPNDST). A separate NIB and RPL are used for each terminal. In the USERFLD field of each NIB, the address of the corresponding RPL is coded.

```
NIB1     NIB      NAME=TERM1,LISTEND=YES,MODE=BASIC,
                  PROC=BLOCK,USERFLD=A(RPL1)
RPL1     RPL      ACB=ACB1,AM=VTAM,NIB=NIB1,AREA=AREA1,
                  AREALEN=100,OPTCD=(ACQUIRE,SPEC,SYN)
AREA1    DS       CL100
```

   Notice that the USERFLD field of NIB1 points to RPL1. The contents of this USERFLD field are returned whenever data from this terminal is received by a READ ANY macro. When this terminal responds to a READ ANY, the USERFLD of the READ ANY's RPL will contain the address of RPL1.

3. Using a separate RPL, this SOLICIT solicits data from all connected terminals.
4. Using the same RPL as did the SOLICIT, this READ ANY reads one block of data from any responding terminal. The RPL for SOLICIT and READ ANY can be coded as:

```
RPL99    RPL      ACB=ACB1,AM=VTAM,AREA=INAREA,
                  AREALEN=100,OPTCD=(SYN,ANY)
INAREA   DS       CL100
```

Whenever a READ ANY is completed, RPL99 will contain the CID and user data for the responding terminal. After processing the input data, contained in INAREA, a reply is prepared in the output area pointed to by the RPL whose address is in the USER field of RPL99.

5. After preparing the reply, the contents of RPL99's USER field is put into a register, and a reply written using that register for the RPL address:

```
WRITE    RPL=(3),OPTCD=ASY,EXIT=WR1
```

   Then, a branch is made back to the READ ANY. When the WRITE is completed, WR1 is scheduled.

6. WR1, an RPL exit routine, is invoked when a WRITE is completed. On entry, register 1 contains the address of the terminal's RPL used for the completed WRITE.
7. Using this RPL address, the terminal is resolicited:

```
SOLICIT  RPL=(1),OPTCD=SPEC
```

Basic-Mode Sample Program 3: ACCEPT, LOGON Exit, SOLICIT, and RPL Exits

Figure A-6 shows the general logic of basic-mode Sample Program 3. This is a more complex program to handle dynamic connection. The program uses a LOGON exit routine to connect terminals that have issued logons for the program. This LOGON routine will also handle simulated logons. The program uses exit-dedicated RPLs for only those terminals that have been connected. This is similar to the previous example in that terminal-dedicated RPLs are used for connection and output data transfer; a fixed RPL is used for input requests. But, the terminal-dedicated RPLs are obtained dynamically, from a pool, as they are needed. (Chapter 3, "Organizing Program," suggests how to handle and construct storage pools.)

To connect terminals, a LOGON exit routine is used that is invoked each time a terminal logs on. The LOGON exit routine gets an RPL and an output area from a pool. Using one fixed NIB, the program issues an asynchronous OPNDST to connect the terminal. As in the previous example, the USERFLD field of the NIB is set to contain the address of the terminal-dedicated RPL that was gotten from a pool, and the terminal-dedicated RPL is set to contain the address of the output storage area.

The asynchronous OPNDST uses and RPL exit routine, which is scheduled whenever the request is completed. The RPL exit routine uses the RPL passed in register 1 to issue a SOLICIT macro to the terminal just connected.

Notes to Sample Program 3:

1. The OPEN macro opens an ACB that specifies that logon requests will be accepted. The ACB points to an exit list, which specifies a LOGON exit routine:

```
          OPEN   ACB
          .
          .
          .
ACB       ACB    AM=VTAM,APPLID=DAVE,
                 EXLST=EXLST1,MACRF=LOGON
DAVE      DC     X'08'
          DC     CL8'HELPS'
EXLST1    EXLST  AM=VTAM,LOGON=LOGON1
```

   The SETLOGON macro initiates queuing of logons for the main program.

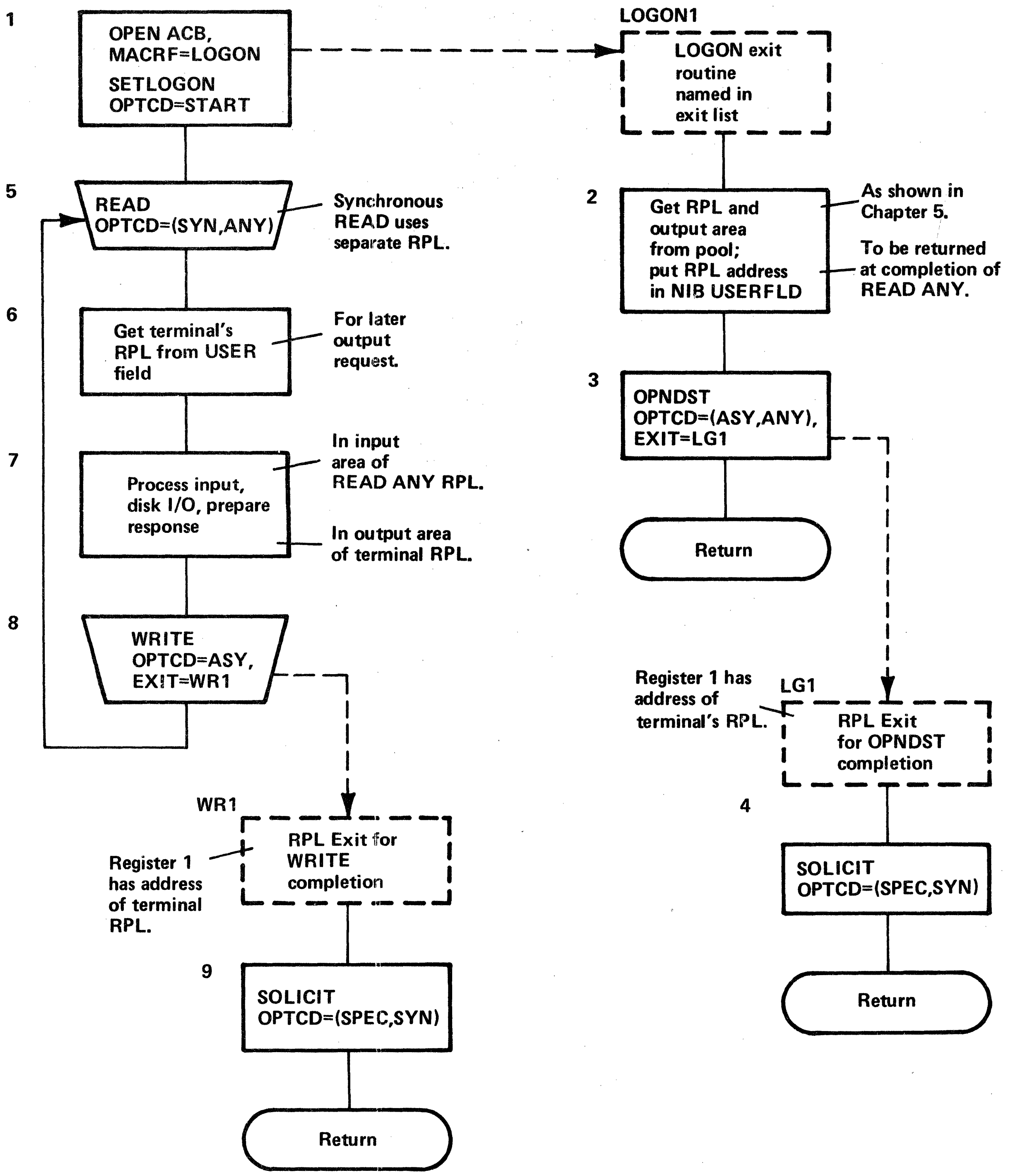

1
OPEN ACB, MACRF=LOGON
SETLOGON OPTCD=START
LOGON1
LOGON exit routine named in exit list
5
READ OPTCD=(SYN,ANY)
Synchronous READ uses separate RPL.
2
Get RPL and output area from pool; put RPL address in NIB USERFLD
As shown in Chapter 5.
To be returned at completion of READ ANY.
6
Get terminal's RPL from USER field
For later output request.
3
OPNDST OPTCD=(ASY,ANY), EXIT=LG1
7
Process input, disk I/O, prepare response
In input area of READ ANY RPL.
In output area of terminal RPL.
Return
8
WRITE OPTCD=ASY, EXIT=WR1
Register 1 has address of terminal's RPL.
LG1
RPL Exit for OPNDST completion
4
SOLICIT OPTCD=(SPEC,SYN)
Return
WR1
RPL Exit for WRITE completion
Register 1 has address of terminal RPL.
9
SOLICIT OPTCD=(SPEC,SYN)
Return

Figure A-6. The Logic of Basic-Mode Sample Program 3

2. When a logon is queued, LOGON1 is entered; the terminal's symbolic name is passed in the parameter list whose address is in register 1. LOGON1 uses a fixed NIB for all connection requests. An RPL and output area are obtained from a storage pool. The address of that RPL is put in the USERFLD field of the NIB.
3. LOGON1 issues an asynchronous OPNDST macro and returns control to the point of interruption.

   Instructions can be added for handling the situation where no more elements are left in the storage pool. For this, a fixed RPL and output area are reserved. Then, the terminal is connected and a "no resources available" message is written, and then, the terminal is disconnected. The reserved RPL and output area are then free in case this situation should again arise.
4. Whenever an OPNDST is completed, the LG1 RPL exit routine is entered. Register 1 contains the address of the RPL used for connection. LG1 uses this RPL to solicit the terminal:

   ```
   SOLICIT    RPL=(1),OPTCD=SPEC
   ```
5. Back in the main program, the synchronous READ ANY macro waits until some data comes in from a solicited terminal. The READ ANY uses the same RPL in all its iterations. When it is completed, the USER field of that RPL contains the the address of the RPL that is dedicated to the responding terminal. This RPL contains the address of the terminal's output area.
6. The main program gets the address of the terminal's RPL for the subsequent WRITE macro. This RPL address is put into a register.
7. After processing the input, which is in the input area named in the RPL for the READ ANY, the program prepares a response in the output area named in the terminal's RPL. The program can also test the input to see if it is a logoff message. When one is received, the program can disconnect the terminal and free any storage elements.
8. The asynchronous WRITE uses the terminal's RPL, and names an RPL exit routine to be invoked when the WRITE is completed. Meanwhile, control is returned to the READ ANY to begin processing more input.
9. The WR1 RPL exit routine is invoked at the completion of a WRITE macro. Using the terminal's RPL, whose address is passed in register 1, WR1 resolicits the terminal.

Appendix B. Summary of Commands and Indicators

This appendix contains tables (Figure B-1 through B-7) that summarize the commands and indicators that can be sent and received by ACF/VTAM application programs. The table summarize these commands and indicators:

Normal-flcw commands

Expedited-flow commands

SESSIONC commands

Change-direction indicators

Bracket indicators

For normal-flow commands, expedited-flow commands, and SESSIONC commands, the information for each command is provided on facing pages. On the left-hand page is information about sending the command, including the purpose of each command, who can send it, the macro instruction used by an application program to send it, the type of data flow on which the command travels, and the next action to be taken by the sender. On the right-hand page is information about receiving the command, including who can receive it, how it is received by an application program, who sends the response to the command, and the next action to be taken by the receiver.

For the change-direction and bracket indicators, the information is summarized on a single page, with the entry for each indicator providing information on both sending and receiving the indicator.

The ability to send and receive the indicators and commands described in this appendix is determined by the session parameters agreed on by the application program and the logical unit when connection is established. For detailed information on session parameters, see Appendix J of *ACF/VTAM Macro Language Reference.*

Command Sent	Function	Who Can Send	Macro Used by Application Program to Send	Data-Flow Type	Next Action by Sender
Bid	Asks receiver for permission to begin a bracket.	Primary application program Secondary application program Logical unit	SEND with STYPE=REQ and CONTROL=BID	DFSYN	Expects response from receiver.[1] Response indicates whether or not the sender can begin a bracket.
Bracket Initiation Stopped (BIS)	Tells the receiver that the sender will not begin any new brackets.	Primary application program Secondary application program Logical unit	SEND with STYPE=REQ and CONTROL=BIS	DFSYN	Expects response from receiver.[1] Refrains from beginning any new brackets.
Cancel	Tells receiver to purge elements of incomplete chain it is receiving.	Primary application program Secondary application program Logical unit	SEND with STYPE=REQ and CONTROL=CANCEL	DFSYN	Expects response from receiver.[1] Positive response indicates that chain elements have been purged.
Chase	Tells receiver to send responses to any data message or command it has not yet responded to.	Primary application program Secondary application program Logical unit	SEND with STYPE=REQ and CONTROL=CHASE	DFSYN	Expects response from receiver.[1] When response to Chase command is received, the sender of the command knows that all responses are accounted for.
Logical Unit Status (LUS)	Informs receiver of an unexpected condition encountered at the sender's end of the session. Codes indicating reason for sending the command are placed in the SSENSEO, SSENSMO and USENSEO fields of the RPL.	Primary application program Secondary application program Logical unit	SEND with STYPE=REQ and CONTROL=LUS	DFSYN	Expects response from receiver.[1]
Quiesce Complete (QC)	Tells receiver that the sender has quiesced itself (as the result of receipt of a Quiesce at End of Chain command) and will not send any normal-flow messages until released.	Primary application program Secondary application program Logical unit	SEND with STYPE=REQ and CONTROL=QC	DFSYN	Expects response from receiver.[1] Refrains from sending any normal-flow messages until a Release Quiesce command is received.
Ready to Receive (RTR)	Tells the receiver that the sender has finished a bracket and that the receiver can now send a request to begin a bracket.	Primary application program Secondary application program Logical unit	SEND with STYPE=REQ and CONTROL=RTR	DFSYN	Expects response from receiver.[1] After receiving the response, an application program issues RECEIVE with RTYPE=DFSYN to receive Bid command or normal-flow message with BB indicator.

[1] An application program can receive the response in one of the following ways, depending on how the program has been coded:

1. By specifying POST=RESP in the SEND macro (macro is not completed until response is received)
2. By issuing a RECEIVE with RTYPE=RESP (a RESP response)
3. In a RESP exit routine
4. By issuing a RECEIVE with RTYPE=DFSYN (a DFSYN response)

Figure B-1. Summary of Sending Normal-Flow Commands

Command Received	Who Can Receive	How Received by Application Program	Who Sends Response	Next Action by Receiver
Bid	Primary application program Secondary application program Logical unit	RECEIVE with RTYPE=DFSYN *CONTROL field in RPL will contain BID.*	Application program or logical unit	Sends positive response to indicate bidder can start a bracket. Sends negative response to deny permission to start a bracket. Application program sends response with SEND...,STYPE=RESP,CONTROL=BID, RESPOND=(*response operands*).
Bracket Initiation Stopped (BIS)	Primary application program Secondary application program Logical unit	RECEIVE with RTYPE=DFSYN *CONTROL field in RPL will contain BIS.*	Application program or logical unit	Sends response to Bracket Initiation Stopped command. Application program sends response by using SEND..., STYPE=RESP, CONTROL=BIS, RESPOND=(*response operands).*
Cancel	Primary application program Secondary application program Logical unit	RECEIVE with RTYPE=DFSYN *CONTROL field in RPL will contain CANCEL.*	Application program or logical unit	Purges any elements of incomplete chain that have been received. Then sends positive response. Application program sends response with SEND...,STYPE=RESP, CONTROL=CANCEL,RESPOND=(*response operands*).
Chase	Primary application program Secondary application program Logical unit	RECEIVE with RTYPE=DFSYN *CONTROL field in RPL will contain CHASE.*	Application program or logical unit	If any responses to previously received messages or commands have not been sent, sends those responses. Then sends response to Chase command. Application program sends response to Chase command with SEND...,STYPE=RESP,CONTROL=CHASE, RESPOND=(*response operands*).
Logical Unit (LUS)	Primary application program Secondary application program Logical unit	RECEIVE with RTYPE=DFSYN *CONTROL field in RPL will contain LUS.* Codes indicating reason for the command are in the SSENSEI, SSENSMI, and USENSEI fields of the RPL.	Application program or logical unit	Examines codes in SSENSEI, SSENSMI, and USENSEI fields of RPL and takes action based on those codes. Then sends response to LUS command. Application program sends response with SEND...,STYPE=RESP, CONTROL=LUS,RESPOND=(*response operands*).
Quiesce Complete (QC)	Primary application program Secondary application program Logical unit	RECEIVE with RTYPE=DFSYN *CONTROL field in RPL will contain QC.*	Application program or logical unit	Sends response to Quiesce Complete command. Application program sends response with SEND...,STYPE=RESP, CONTROL=QC,RESPOND=(*response operands*). Then starts sending to receiver.
Ready to Receive (RTR)	Primary application program Secondary application program Logical unit	RECEIVE with RTYPE=DFSYN *CONTROL field in RPL will contain RTR.*	Application program	Sends response to Ready to Receive command by using SEND...,STYPE=RESP, CONTROL=RTR,RESPOND=(*response operands*). Then, according to session parameters, sends either a Bid command or sends a message that includes BRACKET=BB.

Figure B-2. Summary of Receiving Normal-Flow Commands

Command Sent	Function	Who Can Send	Macro Used by Application Program to Send	Data-Flow Type	Next Action by Sender
Quiesce at End of Chain (QEC)	Tells the receiver to quit sending normal-flow messages now, or if chaining, at the end of the chain being sent.	Primary application program Secondary application program Logical unit	SEND with STYPE=REQ and CONTROL=QEC	DFASY	Expects response from receiver.[1] Response indicates that command has been properly received. After receiving positive response, sends normal-flow input.
Release Quiesce (RELQ)	Tells the receiver that it can resume sending normal-flow messages.	Primary application program Secondary application program Logical unit	SEND with STYPE=REQ and CONTROL=RELQ	DFASY	Expects response from receiver.[1] After receiving positive response, prepares to receive normal-flow input. An application program issues a RECEIVE with RTYPE= DFSYN.
Request Shutdown (RSHUTD)	Asks the primary application program to disconnect the secondary application program or logical unit.	Secondary application program or logical unit only	SEND with STYPE=REQ and CONTROL=RSHUTD	DFASY	Expects response from receiver.[1] Response indicates that command has been properly received.
Shutdown Complete (SHUTC)	Tells the primary application program that shutdown operations (requested previously in a Shutdown command from the primary application program) have been completed.	Secondary application program or logical unit only	SEND with STYPE=REQ and CONTROL=SHUTC	DFASY	Expects response from receiver.[1] Response indicates that command has been properly received.
Shutdown (SHUTD)	Tells the secondary application program or logical unit to quiesce itself and to perform all preparations for Shutdown.	Primary application program only	SEND with STYPE=REQ and CONTROL=SHUTD	DFASY	Expects response from receiver.[1] Response indicates that command has been properly received. Then, expects to receive Shutdown Complete command from receiver.
Signal	Passes a 4-byte message with an agreed-upon meaning. Message is placed in the SIGDATA field of of the RPL.	Primary application program Secondary application program Logical unit	SEND with STYPE=REQ and CONTROL=SIGNAL	DFASY	Expects response from receiver.[1] Response indicates that command has been properly received.
Stop Bracket Initiation (SBI)	Tells receiver not to begin any new brackets.	Primary application program Secondary application program Logical unit	SEND with STYPE=REQ and CONTROL=SBI	DFASY	Expects response from receiver.[1] Response indicates that the command has been properly received.

[1] An application program can receive the response in one of the following ways, according to how the program has been coded:

1. By specifying POST=RESP in the SEND macro (macro is not completed until response is received)
2. By issuing a RECEIVE with RTYPE=RESP
3. In a RESP exit routine

Figure B-3. Summary of Sending Expedited-Flow Commands

Command Received	Who Can Receive	How Received by Application Program	Who Sends Response	Next Action by Receiver
Quiesce at End of Chain (QEC)	Primary application program Secondary application program Logical unit	Either: RECEIVE with RTYPE=DFASY or in DFASY exit routine *CONTROL field in RPL will contain QEC.*	When command is directed to application program, either ACF/VTAM or the program sends the response.[1] Otherwise, logical unit.	Halts sending of normal-flow messages immediately or at end of chain. Then sends Quiesce Complete (QC) command to sender of QEC command.
Release Quiesce (RELQ)	Primary application program Secondary application program Logical unit	Either: RECEIVE with RTYPE=DFASY or in DFASY exit routine *CONTROL field in RPL will contain RELQ.*	When command is directed to application program, either ACF/VTAM or the program sends the response.[1] Otherwise, logical unit.	Sends a normal-flow message to sender of RELQ command, if desired.
Request Shutdown (RSHUTD)	Primary application program only	Either: RECEIVE with RTYPE=DFASY or in DFASY exit routine *CONTROL field in RPL will contain RSHUTD.*	Either ACF/VTAM or the application program.[1]	Disconnects the secondary application program or logical unit by issuing the CLSDST macro. Cleanup operations, including normal-flow communications, can be performed before the CLSDST is issued.
Shutdown Complete (SHUTC)	Primary application program only	Either: RECEIVE with RTYPE=DFASY or in DFASY exit routine *CONTROL field in RPL will contain SHUTC.*	Either ACF/VTAM or the application program.[1]	Issues a Chase command to ensure that all responses have been received. Then disconnects the secondary application program or logical unit by issuing the CLSDST macro.
Shutdown (SHUTD)	Secondary application program or logical unit only	Either: RECEIVE with RTYPE=DFASY or in DFASY exit routine *CONTROL field in RPL will contain SHUTD.*	When command is directed to application program, either ACF/VTAM or the program sends the response.[1] Otherwise, logical unit.	If necessary, stops normal-flow transmission to primary application program. Performs all preparations for shutdown. Then sends the Shutdown Complete command to primary application program.
Signal	Primary application program Secondary application program Logical unit	Either: RECEIVE with RTYPE=DFASY or in DFASY exit routine *CONTROL field in RPL will contain SIGNAL.* Four-byte message is in the SIGDATA field ot the RPL.	When command is directed to application program, either ACF/VTAM or the program sends the response.[1] Otherwise, logical unit.	Depends on the content of the 4-byte message. The Signal messages are defined by Systems Network Architecture (SNA).
Stop Bracket Initiation (SBI)	Primary application program Secondary application program Logical Unit	Either: RECEIVE with RTYPE=DFASY or in DFASY exit routine *CONTROL field in RPL will contain SBI.*	When command is directed to application program, either ACF/VTAM or the program sends the response .[1] Otherwise, logical unit.	Sends a Bracket Initiation Stopped (BIS) command to sender and then refrains from initiating any new brackets.

[1] The responder to an expedited-flow command is determined by the setting of a PROC option in the NIB when the connection is made:

If PROC=APPLRESP was specified in the NIB at connection, the application program sends the response, using SEND...,STYPE=RESP,CONTROL=*command code of received command*,RESPOND=(*response operands*).

If PROC=SYSRESP was specified in the NIB at connection, ACF/VTAM automatically sends the response before presenting the command to the application program.

Figure B-4. Summary of Receiving Expedited-Flow Commands

Command[1] Sent	Function	Who Can Send	Macro Used by Application Program to Send	Next Action by Sender
Bind	Informs the receiver that the sender wants to go into session with the receiver. Session parameters are sent as part of the Bind command.	Primary application program (Command is sent by ACF/VTAM when primary program issues an OPNDST macro.)	Indirectly, by issuing the OPNDST macro	ACF/VTAM handles response and does not complete the OPNDST until response is received. Positive response causes ACF/VTAM to complete setting up the session. Negative response negates the session. After positive response, either ACF/VTAM or the primary application program sends the Start Data Traffic command.
Clear	Tells ACF/VTAM and the receiver to stop sending normal-flow and expedited-flow messages and responses. Causes ACF/VTAM to discard any messages and responses still in the network and not yet delivered. Resets outbound and inbound sequence numbers at both ends of the session to 0.	Primary application program will send command as first step in sequence number recovery. ACF/VTAM may send command as part of disconnection process.	Application program uses SESSIONC with STYPE=REQ and CONTROL=CLEAR ACF/VTAM sends command as a result of issuance of the CLSDST macro	Response reflected in RPL on completion of SESSIONC macro. ACF/VTAM handles response.
Request Recovery (RQR)	Informs primary end of session that sequence number recovery action or message recovery action is needed.	Secondary application program or logical unit	SESSIONC with STYPE=REQ and CONTROL=RQR	In a secondary application program, response reflected in RPL on completion of SESSIONC macro. After receiving positive response, awaits next command from the primary end of the session (usually the Clear command).
Set and Test Sequence Numbers (STSN)	Exchanges information with secondary application program or logical unit so that sequence numbers can be determined and/or set.	Primary application program	SESSIONC with STYPE=REQ and CONTROL=STSN and settings in IBSQAC and/or OBSQAC fields and IBSQVAL and/or OBSQVAL fields	Response reflected in RPL on completion of SESSIONC macro. Tests IBSQAC and OBSQAC fields and IBSQVAL and/or OBSQVAL fields to determine answers to action codes and values sent in the command.
Start Data Traffic (SDT)	Informs secondary application program or logical unit that session setup is complete and flow of messages and responses can begin.	Primary application program	Application program uses SESSIONC with STYPE=REQ and CONTROL=SDT ACF/VTAM sends command at beginning of session if SDT=SYSTEM was set in NIB at connection.	Depending on session parameters, may send first message or wait for secondary end to send message. Either the secondary application program or ACF/VTAM may respond.[2]
Unbind	Informs ACF/VTAM and the receiver that the primary end of the session is terminating the session.	Primary application program (Command is sent by ACF/VTAM when primary application program issues a CLSDST macro.)	Indirectly, by issuing the CLSDST macro	Continues communications with other secondary application programs or logical units, or closes program.

[1] These commands control session-related functions and are sent separately from normal- and expedited-flow messages and their responses. Bracket and change-direction indicators cannot be sent with SESSIONC commands or responses.

[2] If the secondary application program specified SDT=APPL for the NIB used in OPNSEC processing, the secondary application will respond. Otherwise (SDT=SYSTEM), ACF/VTAM will respond.

Figure B-5. Summary of Sending Session-Control Commands

Command Received	Who Can Receive	How Received by Application Program	Who Sends Response	Next Action by Receiver
Bind	Secondary application program Logical unit	For a secondary application program, receipt of command causes scheduling of the SCIP exit routine.	Application program or logical unit	Examines session parameters in Bind command and determines whether or not the complete set of parameters is acceptable. If acceptable, sends positive response. (For secondary application program, positive response results from issuance of the OPNSEC macro.) If not acceptable, sends negative response. (Secondary application program sends negative response with SESSIONC..., STYPE=RESP,CONTROL=BIND, RESPOND=(*response operands*).)
Clear	Secondary application program Logical unit	For a secondary application program, receipt of command causes scheduling of the SCIP exit routine.	When command is directed to a secondary application program, ACF/VTAM sends response. Otherwise, logical unit.	Stops sending messages and responses, and awaits next command from the primary end of the session.
Request Recovery (RQR)	Primary application program	In SCIP exit routine.	ACF/VTAM	Initiates recovery action, usually by sending the Clear command followed by a Set and Test Sequence Numbers command.
Set and Test Sequence Numbers (STSN)	Secondary application program Logical unit	For a secondary application program, receipt of command causes scheduling of the SCIP exit routine.	Application program or logical unit	Examines action codes and sequence number values provided with the command. Prepares answering action codes and values and puts them in IBSQAC and/or OBSQAC fields and IBSQVAL and/or OBSQVAL fields. Then, sends response with SESSIONC..., STYPE=RESP,CONTROL=STSN.
Start Data Traffic (SDT)	Secondary application program Logical unit	For a secondary application program, receipt of command causes scheduling of the SCIP exit routine.	Depending on the SDT field in the NIB used during OPNSEC processing, either the secondary application program or ACF/VTAM may respond.	Depending on session parameters, may send first message or wait for primary end to send send message.
Unbind	Secondary application program Logical unit	For secondary application program, receipt of command causes scheduling of the SCIP exit routine.	ACF/VTAM	Continues communication with other primary application programs or logical units, or closes program.

Figure B-6. Summary of Receiving Session-Control Commands

Indicator	Function	Primary or Secondary Application Program Can Send/Receive	Macro Used or RPL Field Set	Data-Flow Type	Next Action Expected

Change-Direction Indicators

The Change Direction Command indicator can be sent in a normal-flow message or with a Cancel, Chase, Quiesce Complete, or LU Status command. The element containing the Change Direction Command indicator must be a single-element message (only-in-chain) or the last element in a chain.

The Change Direction Request indicator is a *non-SNA* indicator supported by ACF/VTAM and certain logical units. It may be sent in a normal-flow data message and with a Cancel, Chase, Quiesce Complete, or LU Status command. It is recommended that this indicator *not* be used, because its use is incompatible with SNA. SNA uses the Signal command to request a Change Direction Command indicator.

Indicator	Function	Primary or Secondary Application Program Can Send/Receive	Macro Used or RPL Field Set	Data-Flow Type	Next Action Expected
Change Direction Command (CHD)	Tells the receiver that it may now send.	Send	SEND with CHNGDIR=CMD	DFSYN	Start receiving from the opposite end of the session.
		Receive	CHNGDIR field in RPL contains CMD	DFSYN	Start sending to the opposite end of the session.
Change Direction Request (REQ) (This is a non-SNA indicator)	Asks receiver to send a Change Direction Command indicator.	Send	SEND with CHNGDIR=REQ	DFASY or RESP	Checks incoming messages and responses for CMD in CHNGDIR field of RPL.
		Receive	CHNGDIR field in RPL contains REQ	DFASY or RESP	Sends the Change Direction Command indicator when it can.

Bracket Indicators

The normal-flow commands Bid and Ready to Receive are used by the ACF/VTAM application program to determine whether it can send a Begin Bracket indicator.

The bracket indicators can be sent in a message that contains data. In addition, the End Bracket indicator can be sent with a Cancel, Chase, Quiesce Complete, or LU Status command. A change-direction indicator can also be sent in the same message. The element containing the Begin Bracket or End Bracket indicator must be a single-element message (only-in-chain) or the first element in a chain.

Indicator	Function	Primary or Secondary Application Program Can Send/Receive	Macro Used or RPL Field Set	Data-Flow Type	Next Action Expected
Begin Bracket (BB)	Indicates first message in a bracket.	Send	SEND with BRACKET=BB	DFSYN	Continues to send or waits to receive, according to user conventions.
		Receive	BRACKET field in RPL contains BB	DFSYN	None.
End Bracket (EB)	Indicates last message in a bracket.	Send	SEND with BRACKET=EB	DFSYN	Attempts to start a new bracket, or waits for other end to start a bracket, according to user conventions.
		Receive	BRACKET field in RPL contains EB	DFSYN	Can mean "end of transaction."

Figure B-7. Summary of Indicators

Appendix C. Examples of Message, Response, and Command Exchanges for Typical Communication Operations

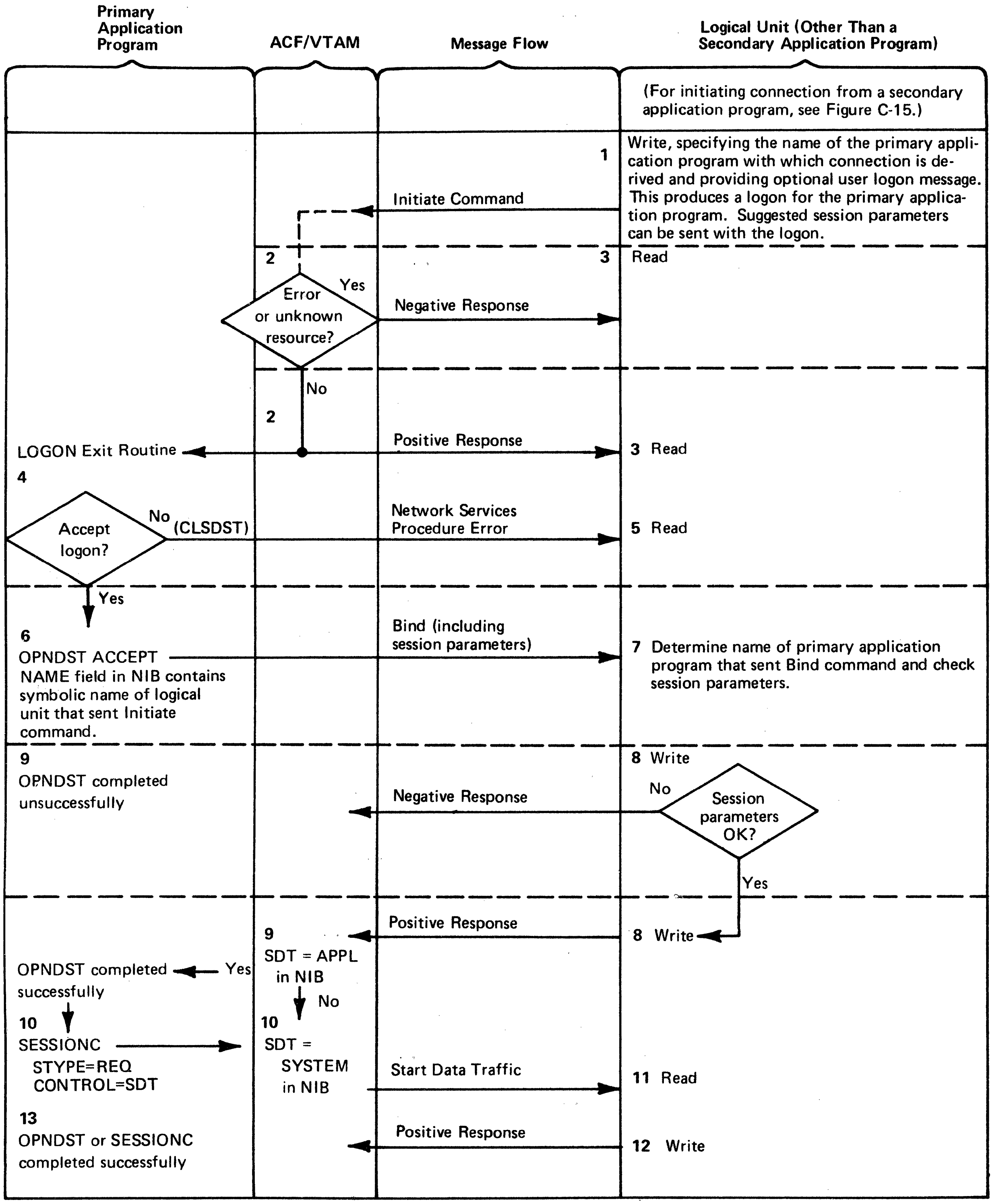
This appendix contains diagrams that show the sequence in which messages, responses, and commands are exchanged to perform typical data communication operations using ACF/VTAM. The diagrams can be useful in coding application programs that perform the operations.

Figures C-1 through C-14 are oriented primarily toward communication between a primary application program and a logical unit other than a secondary application program, although some of these diagrams apply also when the logical unit is a secondary application program. In Figures C-1 through C-14, the "reads" and "writes" shown in the "Logical Unit" column represent logic that may be performed by a control program in the logical unit, a user-written program that operates in the logical unit, or both. It is a general representation of the input and output from the logical unit. The primary program's side of the exchange is shown in more detail. The language elements in the "Primary Application Program" column show all of the alternatives that are possible (for example, when a SEND with STYPE=REQ and RESPOND=EX is specified, the ways of receiving a negative response are with a RECEIVE with RTYPE=REP or with a RESP exit routine, and both ways are mentioned in the diagrams).

Figures C-15 through C-24 are oriented toward operations between a primary application program and a secondary application program.

In any diagram showing a negative response being sent to an application program, a SYNAD exit routine would be scheduled with an exception condition return code. This is not shown.

To help you find the diagram you want, here is a list of the figures that appear in this appendix:



Primary Application Program
ACF/VTAM
Message Flow
Logical Unit (Other Than a Secondary Application Program)
(For initiating connection from a secondary application program, see Figure C-15.)
1
Write, specifying the name of the primary application program with which connection is derived and providing optional user logon message. This produces a logon for the primary application program. Suggested session parameters can be sent with the logon.
Initiate Command
2
3
Read
Error or unknown resource?
Yes
Negative Response
No
2
Positive Response
LOGON Exit Routine
3 Read
4
Accept logon?
No
(CLSDST)
Network Services Procedure Error
5 Read
Yes
6
OPNDST ACCEPT
NAME field in NIB contains symbolic name of logical unit that sent Initiate command.
Bind (including session parameters)
7 Determine name of primary application program that sent Bind command and check session parameters.
9
OPNDST completed unsuccessfully
8 Write
No
Session parameters OK?
Negative Response
Yes
Positive Response
9
8 Write
SDT = APPL in NIB
OPNDST completed successfully
Yes
No
10
10
SESSIONC
STYPE=REQ
CONTROL=SDT
SDT = SYSTEM in NIB
Start Data Traffic
11 Read
13
OPNDST or SESSIONC completed successfully
Positive Response
12 Write

Figure C-1. A Logical Unit (Other Than a Secondary Application Program) Initiates Connection with a Primary Application Program

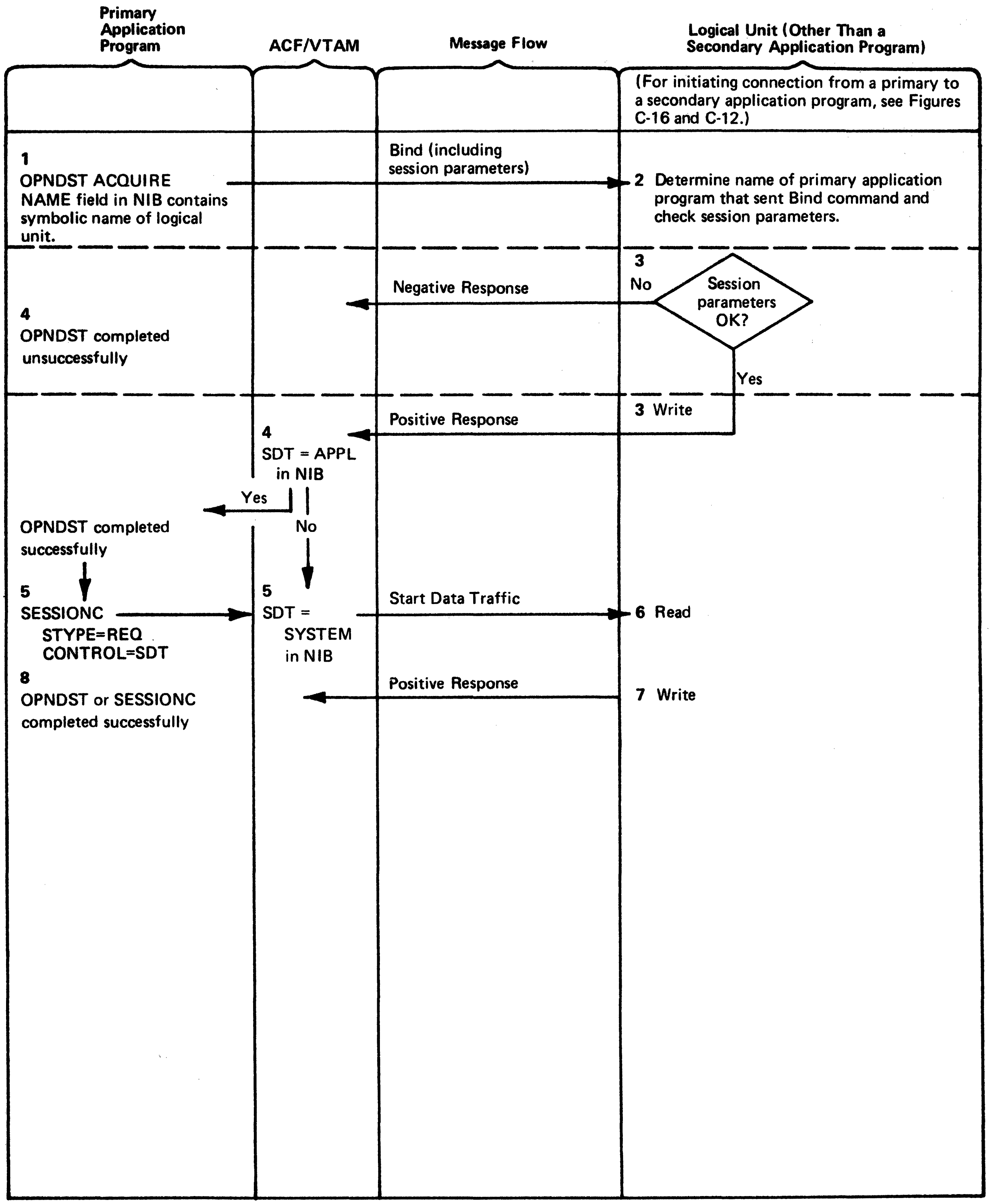
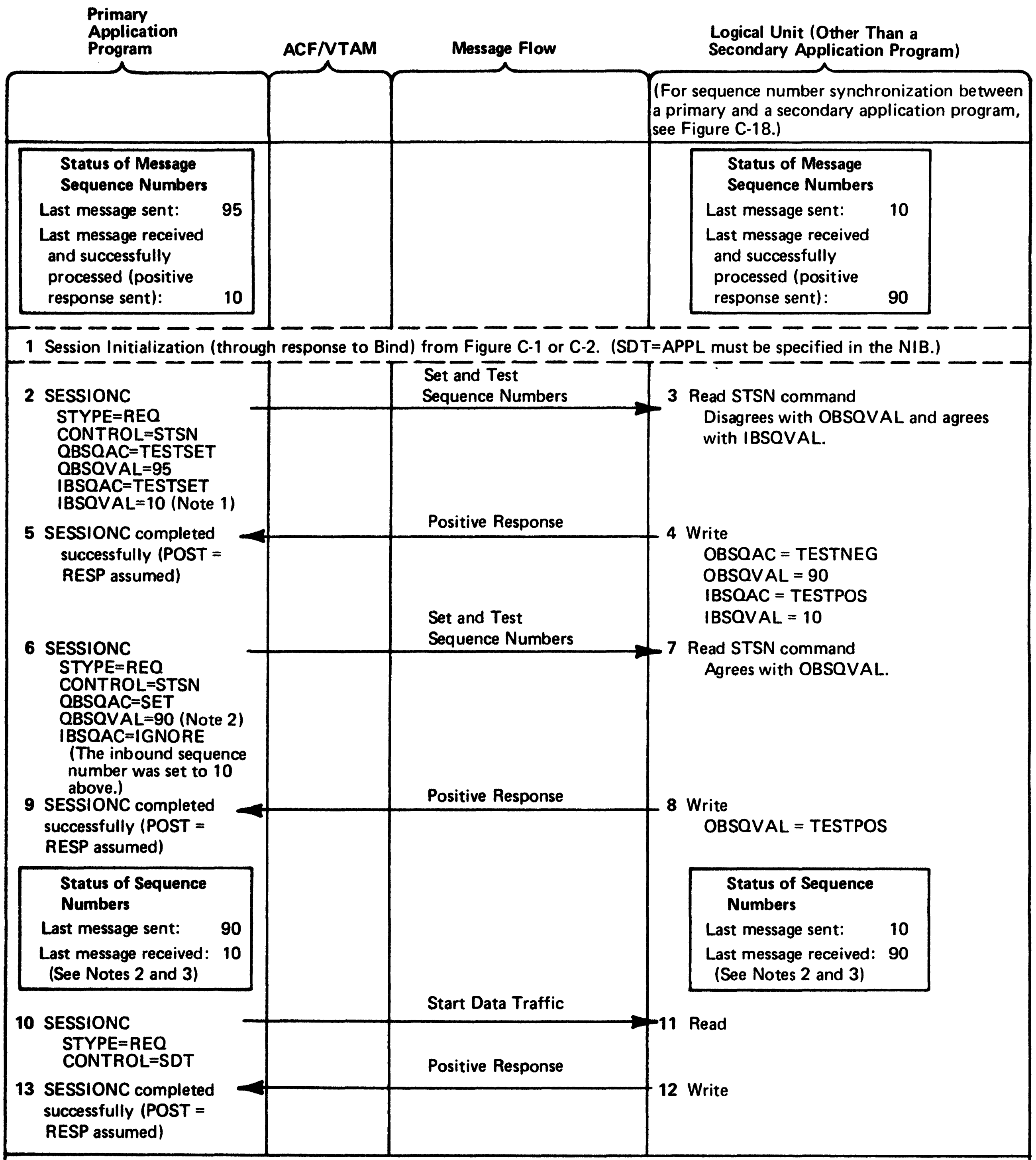

Primary Application Program
ACF/VTAM
Message Flow
Logical Unit (Other Than a Secondary Application Program)
(For initiating connection from a primary to a secondary application program, see Figures C-16 and C-12.)
1 OPNDST ACQUIRE NAME field in NIB contains symbolic name of logical unit.
Bind (including session parameters)
2 Determine name of primary application program that sent Bind command and check session parameters.
3
Negative Response
No
Session parameters OK?
4 OPNDST completed unsuccessfully
Yes
3 Write
Positive Response
4 SDT = APPL in NIB
Yes
No
OPNDST completed successfully
5 SESSIONC STYPE=REQ CONTROL=SDT
5 SDT = SYSTEM in NIB
Start Data Traffic
6 Read
8 OPNDST or SESSIONC completed successfully
Positive Response
7 Write

Figure C-2. A Primary Application Program Acquires a Logical Unit


Primary Application Program

ACF/VTAM

Message Flow

Logical Unit (Other Than a Secondary Application Program)

(For sequence number synchronization between a primary and a secondary application program, see Figure C-18.)

Status of Message Sequence Numbers
Last message sent: 95
Last message received and successfully processed (positive response sent): 10

Status of Message Sequence Numbers
Last message sent: 10
Last message received and successfully processed (positive response sent): 90

1 Session Initialization (through response to Bind) from Figure C-1 or C-2. (SDT=APPL must be specified in the NIB.)

2 SESSIONC
STYPE=REQ
CONTROL=STSN
OBSQAC=TESTSET
OBSQVAL=95
IBSQAC=TESTSET
IBSQVAL=10 (Note 1)

Set and Test Sequence Numbers

3 Read STSN command
Disagrees with OBSQVAL and agrees with IBSQVAL.

Positive Response

4 Write
OBSQAC = TESTNEG
OBSQVAL = 90
IBSQAC = TESTPOS
IBSQVAL = 10

5 SESSIONC completed successfully (POST = RESP assumed)

6 SESSIONC
STYPE=REQ
CONTROL=STSN
OBSQAC=SET
OBSQVAL=90 (Note 2)
IBSQAC=IGNORE
(The inbound sequence number was set to 10 above.)

Set and Test Sequence Numbers

7 Read STSN command
Agrees with OBSQVAL.

Positive Response

8 Write
OBSQVAL = TESTPOS

9 SESSIONC completed successfully (POST = RESP assumed)

Status of Sequence Numbers
Last message sent: 90
Last message received: 10
(See Notes 2 and 3)

Status of Sequence Numbers
Last message sent: 10
Last message received: 90
(See Notes 2 and 3)

10 SESSIONC
STYPE=REQ
CONTROL=SDT

Start Data Traffic

11 Read

Positive Response

12 Write

13 SESSIONC completed successfully (POST = RESP assumed)

Notes:

1. Notice that, in this figure, the mnemonic OB stands for *outbound from ACF/VTAM* (therefore, inbound to the logical unit) and the mnemonic IB stands for *inbound to ACF/VTAM* (therefore, outbound from the logical unit).
2. Outbound messages 91-95 from the application program were lost and will have to be resent.
3. The positive response sent by the application program for inbound message 10 may never have reached the logical unit, but it can be inferred from the first Set and Test Sequence Numbers command.

Figure C-3. After a Warm Start, a Primary Application Program Reestablishes Connection and Resynchronizes Sequence Numbers

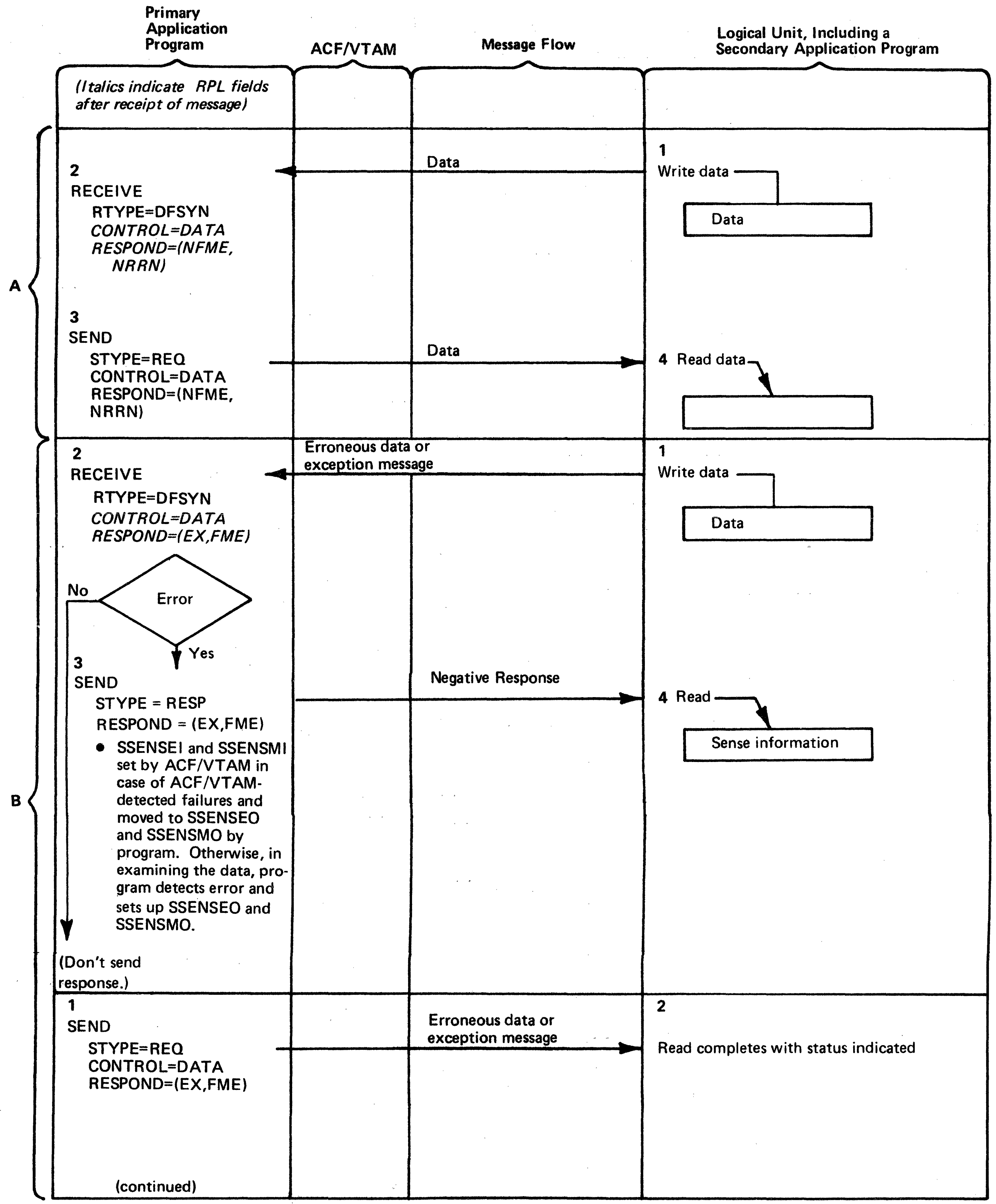

Primary Application Program
ACF/VTAM
Message Flow
Logical Unit, Including a Secondary Application Program
(Italics indicate RPL fields after receipt of message)
A
2
RECEIVE
RTYPE=DFSYN
CONTROL=DATA
RESPOND=(NFME, NRRN)
Data
1
Write data
Data
3
SEND
STYPE=REQ
CONTROL=DATA
RESPOND=(NFME, NRRN)
Data
4 Read data
B
2
RECEIVE
RTYPE=DFSYN
CONTROL=DATA
RESPOND=(EX,FME)
Erroneous data or exception message
1
Write data
Data
No
Error
Yes
3
SEND
STYPE = RESP
RESPOND = (EX,FME)
• SSENSEI and SSENSMI set by ACF/VTAM in case of ACF/VTAM-detected failures and moved to SSENSEO and SSENSMO by program. Otherwise, in examining the data, program detects error and sets up SSENSEO and SSENSMO.
Negative Response
4 Read
Sense information
(Don't send response.)
1
SEND
STYPE=REQ
CONTROL=DATA
RESPOND=(EX,FME)
Erroneous data or exception message
2
Read completes with status indicated
(continued)

Figure C-4 (Part 1 of 3). A Primary Application Program and a Logical Unit Exchange Messages: (A) with No Responses, (B) with Negative Responses Only If an Exception Occurs, (C) with Definite Response 1 (Positive or Negative), and (D) with Definite Responses 1 and 2 Sent at the Same Time

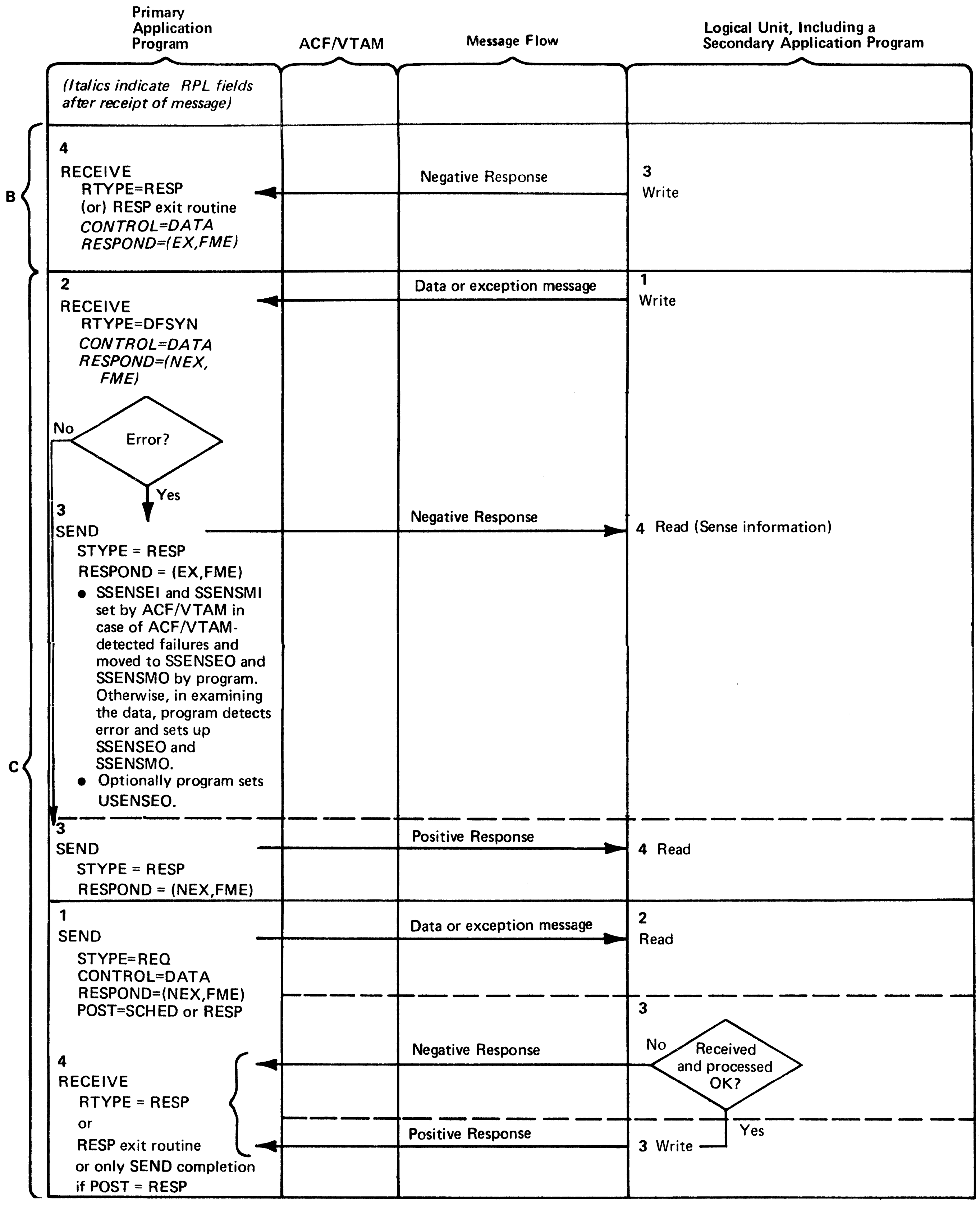

Primary Application Program
ACF/VTAM
Message Flow
Logical Unit, Including a Secondary Application Program
(Italics indicate RPL fields after receipt of message)
B
4
RECEIVE
RTYPE=RESP
(or) RESP exit routine
CONTROL=DATA
RESPOND=(EX,FME)
Negative Response
3
Write
C
2
RECEIVE
RTYPE=DFSYN
CONTROL=DATA
RESPOND=(NEX, FME)
Data or exception message
1
Write
Error?
No
Yes
3
SEND
STYPE = RESP
RESPOND = (EX,FME)
• SSENSEI and SSENSMI set by ACF/VTAM in case of ACF/VTAM-detected failures and moved to SSENSEO and SSENSMO by program. Otherwise, in examining the data, program detects error and sets up SSENSEO and SSENSMO.
• Optionally program sets USENSEO.
Negative Response
4 Read (Sense information)
3
SEND
STYPE = RESP
RESPOND = (NEX,FME)
Positive Response
4 Read
1
SEND
STYPE=REQ
CONTROL=DATA
RESPOND=(NEX,FME)
POST=SCHED or RESP
Data or exception message
2
Read
3
Received and processed OK?
No
Yes
4
RECEIVE
RTYPE = RESP
or
RESP exit routine
or only SEND completion if POST = RESP
Negative Response
Positive Response
3 Write

Figure C-4 (Part 2 of 3). A Primary Application Program and a Logical Unit Exchange Messages: (A) with No Responses, (B) with Negative Responses Only If an Exception Occurs, (C) with Definite Response 1 (Positive or Negative), and (D) with Definite Responses 1 and 2 Sent at the Same Time

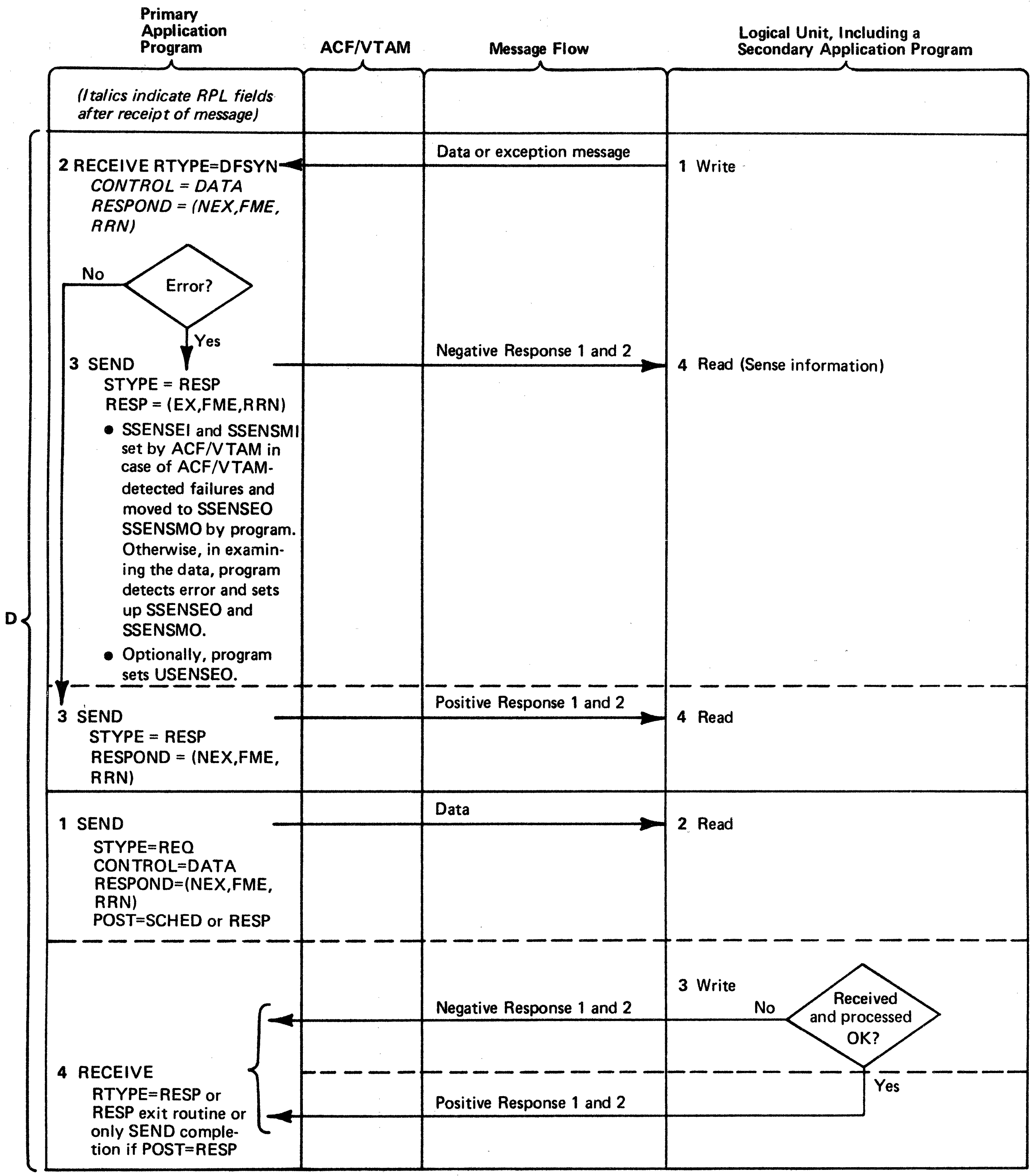
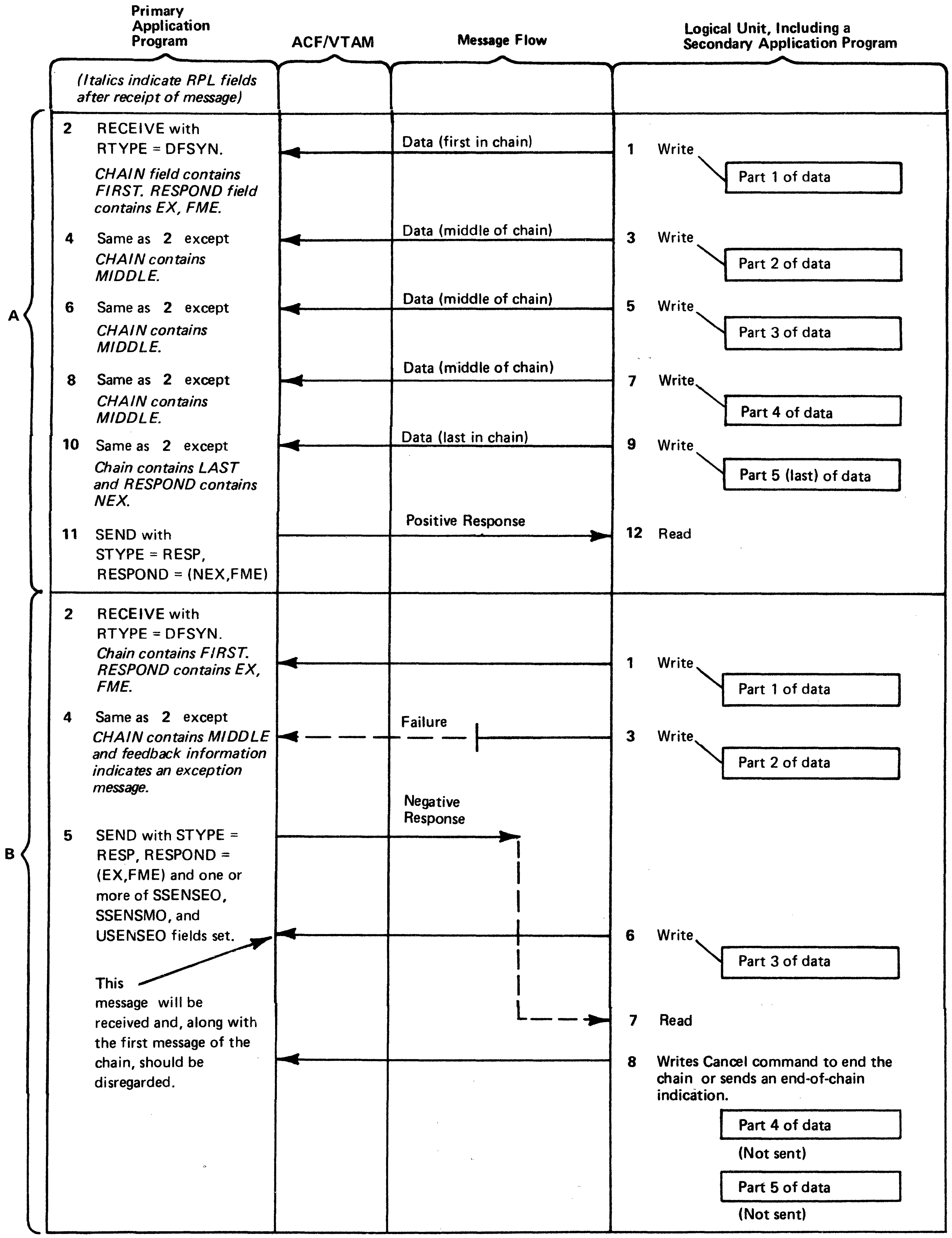
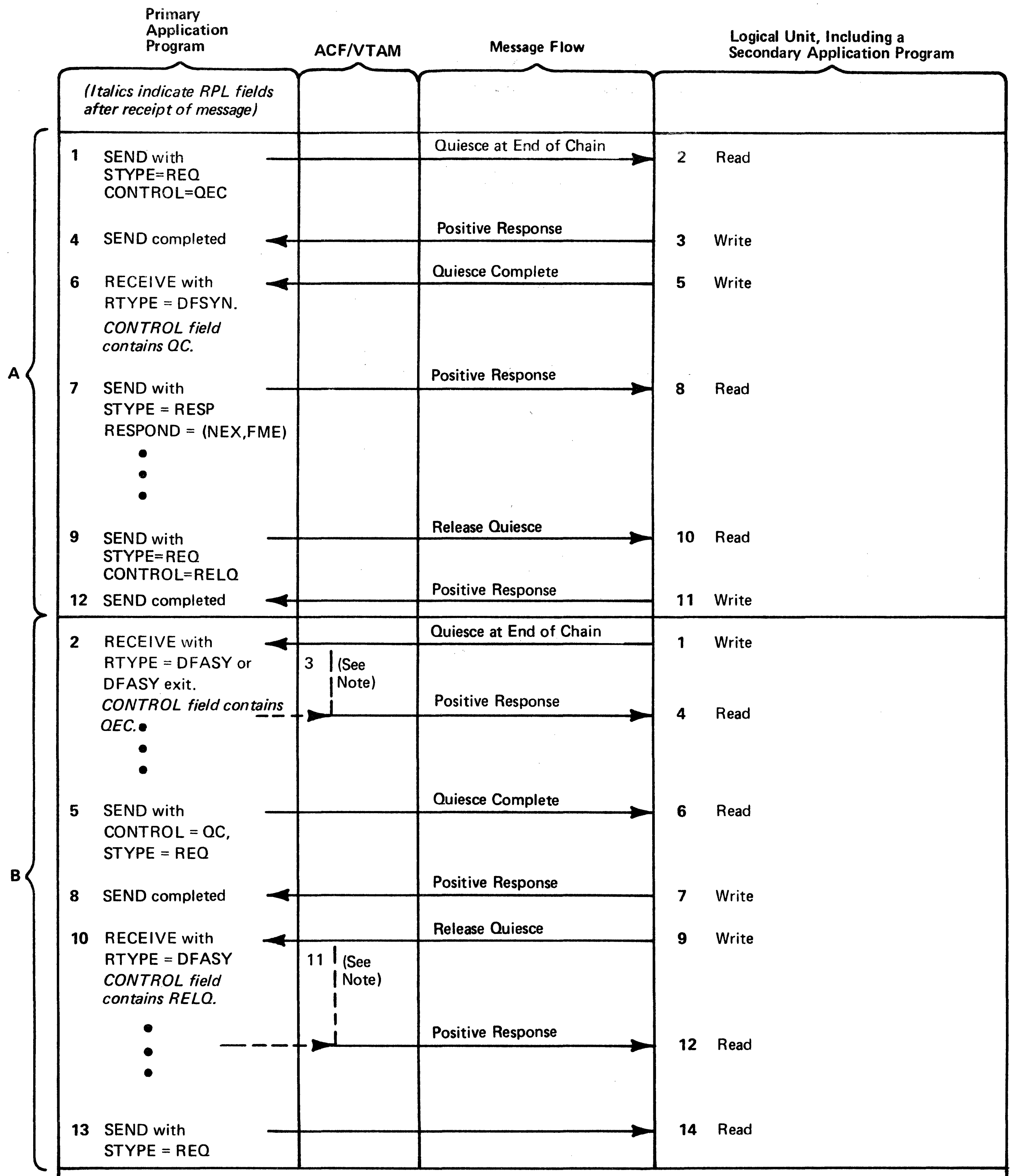

Primary Application Program
ACF/VTAM
Message Flow
Logical Unit, Including a Secondary Application Program
(Italics indicate RPL fields after receipt of message)
D
2 RECEIVE RTYPE=DFSYN
CONTROL = DATA
RESPOND = (NEX,FME, RRN)
Data or exception message
1 Write
No
Error?
Yes
3 SEND
STYPE = RESP
RESP = (EX,FME,RRN)
• SSENSEI and SSENSMI set by ACF/VTAM in case of ACF/VTAM-detected failures and moved to SSENSEO SSENSMO by program. Otherwise, in examining the data, program detects error and sets up SSENSEO and SSENSMO.
• Optionally, program sets USENSEO.
Negative Response 1 and 2
4 Read (Sense information)
3 SEND
STYPE = RESP
RESPOND = (NEX,FME, RRN)
Positive Response 1 and 2
4 Read
1 SEND
STYPE=REQ
CONTROL=DATA
RESPOND=(NEX,FME, RRN)
POST=SCHED or RESP
Data
2 Read
3 Write
No
Received and processed OK?
Yes
Negative Response 1 and 2
4 RECEIVE
RTYPE=RESP or RESP exit routine or only SEND completion if POST=RESP
Positive Response 1 and 2

Figure C-4 (Part 3 of 3). A Primary Application Program and a Logical Unit Exchange Messages: (A) with No Responses, (B) with Negative Responses Only If an Exception Occurs, (C) with Definite Response 1 (Positive or Negative), and (D) with Definite Responses 1 and 2 Sent at the Same Time



Primary Application Program
ACF/VTAM
Message Flow
Logical Unit, Including a Secondary Application Program
(Italics indicate RPL fields after receipt of message)
A
2 RECEIVE with RTYPE = DFSYN.
CHAIN field contains FIRST. RESPOND field contains EX, FME.
Data (first in chain)
1 Write
Part 1 of data
4 Same as 2 except CHAIN contains MIDDLE.
Data (middle of chain)
3 Write
Part 2 of data
6 Same as 2 except CHAIN contains MIDDLE.
Data (middle of chain)
5 Write
Part 3 of data
8 Same as 2 except CHAIN contains MIDDLE.
Data (middle of chain)
7 Write
Part 4 of data
10 Same as 2 except Chain contains LAST and RESPOND contains NEX.
Data (last in chain)
9 Write
Part 5 (last) of data
11 SEND with STYPE = RESP, RESPOND = (NEX,FME)
Positive Response
12 Read
B
2 RECEIVE with RTYPE = DFSYN. Chain contains FIRST. RESPOND contains EX, FME.
1 Write
Part 1 of data
4 Same as 2 except CHAIN contains MIDDLE and feedback information indicates an exception message.
Failure
3 Write
Part 2 of data
5 SEND with STYPE = RESP, RESPOND = (EX,FME) and one or more of SSENSEO, SSENSMO, and USENSEO fields set.
Negative Response
6 Write
Part 3 of data
This message will be received and, along with the first message of the chain, should be disregarded.
7 Read
8 Writes Cancel command to end the chain or sends an end-of-chain indication.
Part 4 of data
(Not sent)
Part 5 of data
(Not sent)

Figure C-5. The Logical Unit Sends a Chain of Messages to the Primary Application Program: (A) without a Negative Response, and (B) with a Negative Response


Primary Application Program
ACF/VTAM
Message Flow
Logical Unit, Including a Secondary Application Program
(Italics indicate RPL fields after receipt of message)
A
1 SEND with STYPE=REQ CONTROL=QEC
Quiesce at End of Chain
2 Read
4 SEND completed
Positive Response
3 Write
6 RECEIVE with RTYPE = DFSYN. CONTROL field contains QC.
Quiesce Complete
5 Write
7 SEND with STYPE = RESP RESPOND = (NEX,FME)
Positive Response
8 Read
9 SEND with STYPE=REQ CONTROL=RELQ
Release Quiesce
10 Read
12 SEND completed
Positive Response
11 Write
B
2 RECEIVE with RTYPE = DFASY or DFASY exit. CONTROL field contains QEC.
Quiesce at End of Chain
1 Write
3 (See Note)
Positive Response
4 Read
5 SEND with CONTROL = QC, STYPE = REQ
Quiesce Complete
6 Read
8 SEND completed
Positive Response
7 Write
10 RECEIVE with RTYPE = DFASY CONTROL field contains RELQ.
Release Quiesce
9 Write
11 (See Note)
Positive Response
12 Read
13 SEND with STYPE = REQ
14 Read

Note: The response to the Quiesce at End of Chain command and the Release Quiesce command (both expedited-flow commands) is sent either by ACF/VTAM or the application program, depending on the setting of a PROC option in the NIB when the connection was made:

- If PROC=APPLRESP was specified in the NIB at connection, the application program sends the response, using SEND . . . ,STYPE=RESP,CONTROL=*command code of received command*,RESPOND=(*response operands*).
- If PROC=SYSRESP was specified in the NIB at connection, ACF/VTAM automatically sends the response before presenting the command to the application program.

Figure C-6. The Application Program and Logical Unit Use Quiesce Protocol: (A) the Application Program Quiesces the Logical Unit, and (B) the Logical Unit Quiesces the Application Program

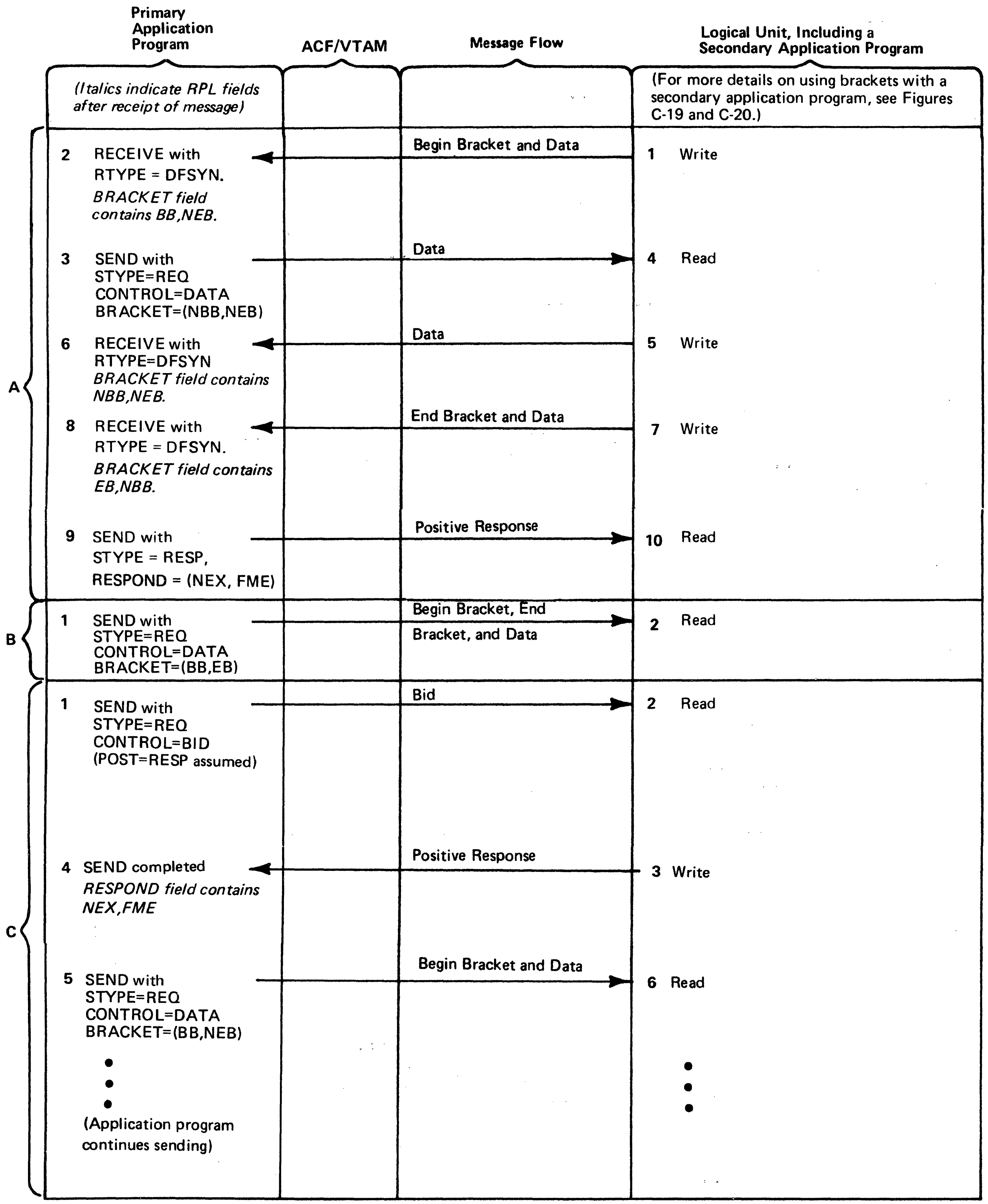
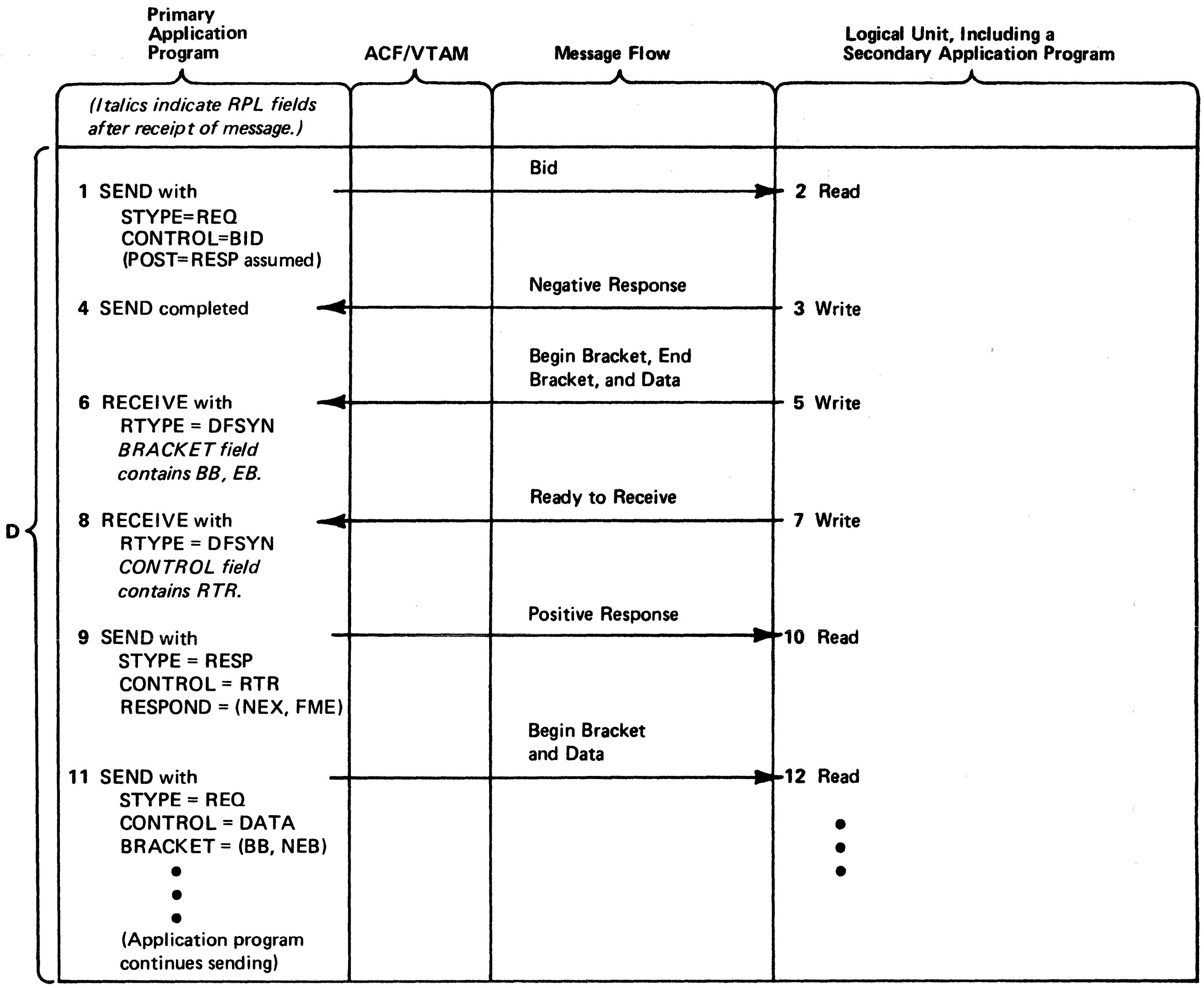

Primary Application Program
ACF/VTAM
Message Flow
Logical Unit, Including a Secondary Application Program
(Italics indicate RPL fields after receipt of message)
(For more details on using brackets with a secondary application program, see Figures C-19 and C-20.)
A
2 RECEIVE with RTYPE = DFSYN. *BRACKET field contains BB,NEB.*
Begin Bracket and Data
1 Write
3 SEND with STYPE=REQ CONTROL=DATA BRACKET=(NBB,NEB)
Data
4 Read
6 RECEIVE with RTYPE=DFSYN *BRACKET field contains NBB,NEB.*
Data
5 Write
8 RECEIVE with RTYPE = DFSYN. *BRACKET field contains EB,NBB.*
End Bracket and Data
7 Write
9 SEND with STYPE = RESP, RESPOND = (NEX, FME)
Positive Response
10 Read
B
1 SEND with STYPE=REQ CONTROL=DATA BRACKET=(BB,EB)
Begin Bracket, End Bracket, and Data
2 Read
C
1 SEND with STYPE=REQ CONTROL=BID (POST=RESP assumed)
Bid
2 Read
4 SEND completed *RESPOND field contains NEX,FME*
Positive Response
3 Write
5 SEND with STYPE=REQ CONTROL=DATA BRACKET=(BB,NEB)
Begin Bracket and Data
6 Read
(Application program continues sending)

Figure C-7 (Part 1 of 2). The Application Program and Logical Unit Use Bracket Protocol: (A) Where the Logical Unit Begins the Bracket, (B) Where the Primary Application Program Begins the Bracket, (C) Where the Primary Application Program Gets a Positive Response to Its Bid and Begins the Bracket, and (D) Where Bid Produces a Later Ready to Receive Command



Primary Application Program
ACF/VTAM
Message Flow
Logical Unit, Including a Secondary Application Program
(Italics indicate RPL fields after receipt of message.)
D
1 SEND with STYPE=REQ CONTROL=BID (POST=RESP assumed)
Bid
2 Read
Negative Response
3 Write
4 SEND completed
Begin Bracket, End Bracket, and Data
5 Write
6 RECEIVE with RTYPE = DFSYN BRACKET field contains BB, EB.
Ready to Receive
7 Write
8 RECEIVE with RTYPE = DFSYN CONTROL field contains RTR.
Positive Response
9 SEND with STYPE = RESP CONTROL = RTR RESPOND = (NEX, FME)
10 Read
Begin Bracket and Data
11 SEND with STYPE = REQ CONTROL = DATA BRACKET = (BB, NEB)
12 Read
(Application program continues sending)

Figure C-7 (Part 2 of 2). The Application Program and Logical Unit Use Bracket Protocol: (A) Where the Logical Unit Begins the Bracket, (B) Where the Primary Application Program Begins the Bracket, (C) Where the Primary Application Program Gets a Positive Response to Its Bid and Begins the Bracket, and (D) Where Bid Produces a Later Ready to Receive Command

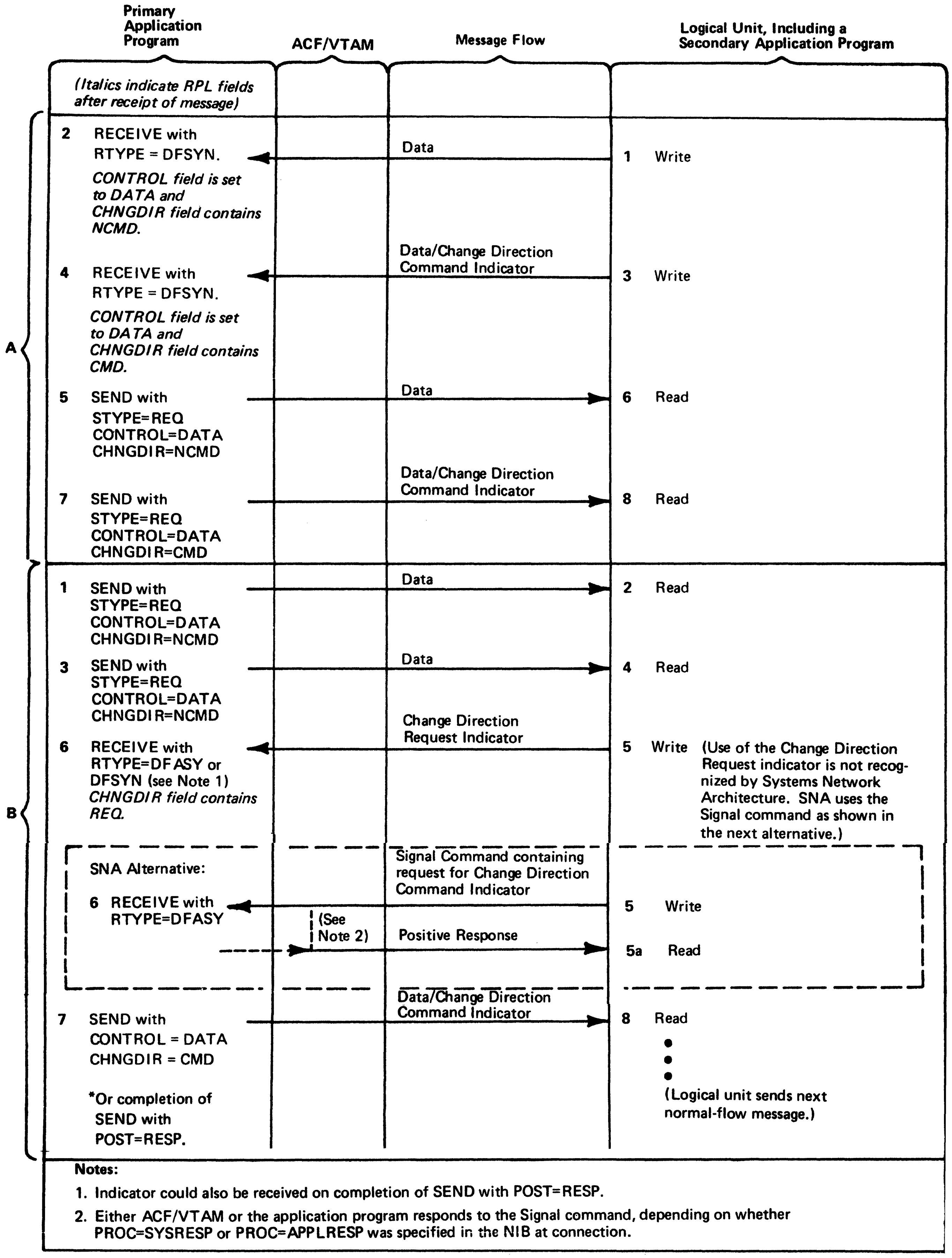
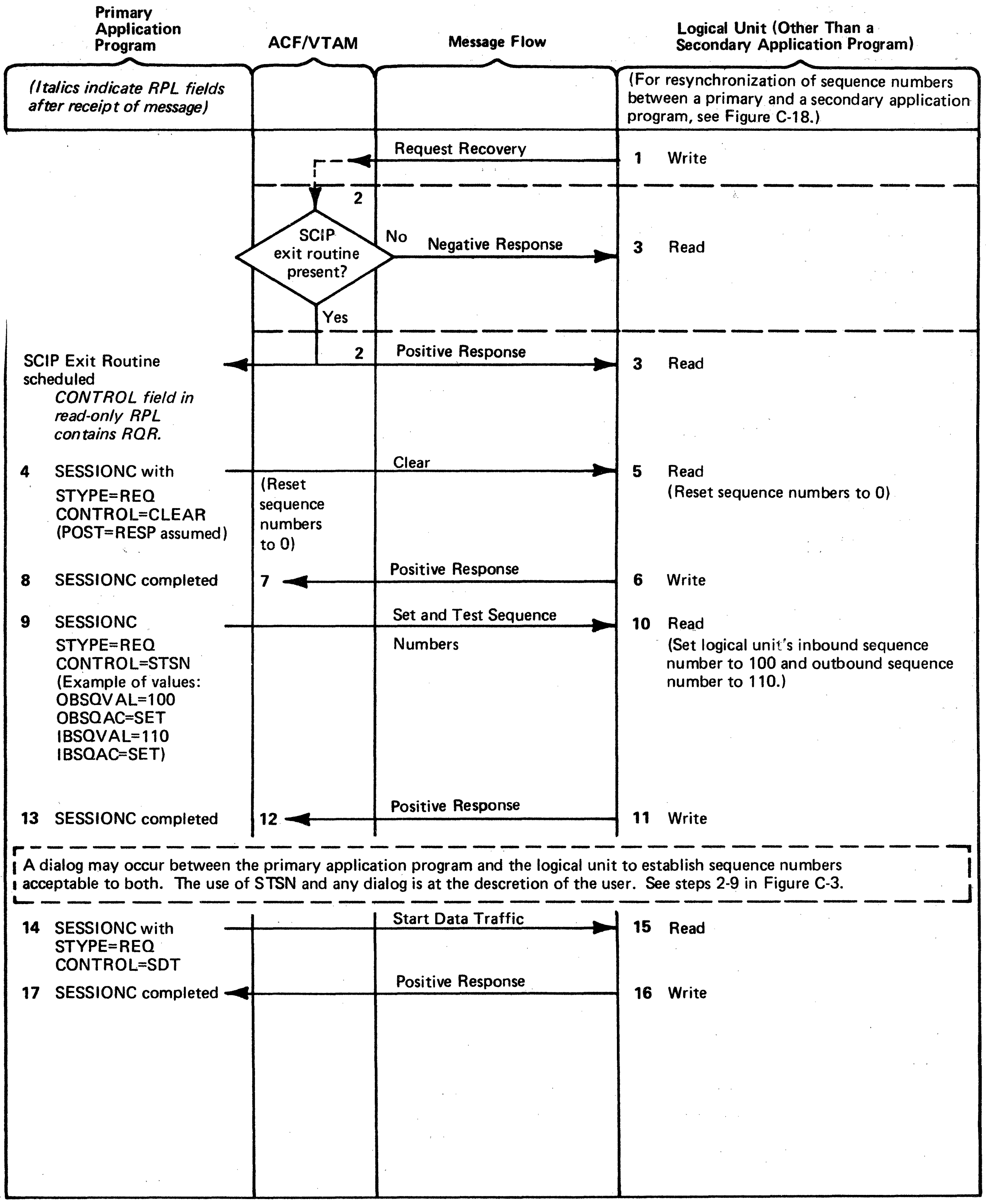
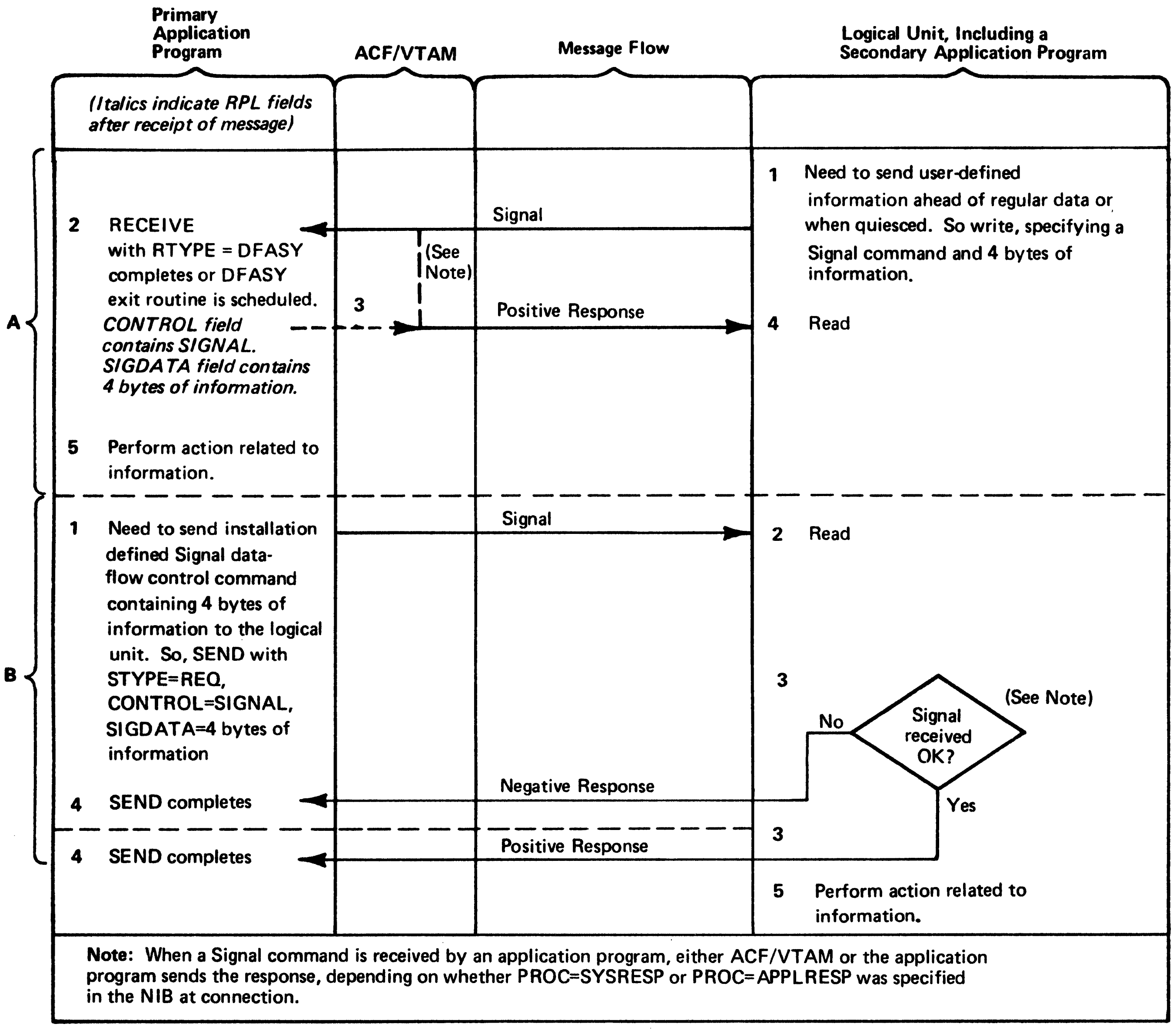

Primary Application Program
ACF/VTAM
Message Flow
Logical Unit, Including a Secondary Application Program
(Italics indicate RPL fields after receipt of message)
A
2 RECEIVE with RTYPE = DFSYN.
CONTROL field is set to DATA and CHNGDIR field contains NCMD.
Data
1 Write
Data/Change Direction Command Indicator
4 RECEIVE with RTYPE = DFSYN.
CONTROL field is set to DATA and CHNGDIR field contains CMD.
3 Write
5 SEND with STYPE=REQ CONTROL=DATA CHNGDIR=NCMD
Data
6 Read
7 SEND with STYPE=REQ CONTROL=DATA CHNGDIR=CMD
Data/Change Direction Command Indicator
8 Read
B
1 SEND with STYPE=REQ CONTROL=DATA CHNGDIR=NCMD
Data
2 Read
3 SEND with STYPE=REQ CONTROL=DATA CHNGDIR=NCMD
Data
4 Read
6 RECEIVE with RTYPE=DFASY or DFSYN (see Note 1) CHNGDIR field contains REQ.
Change Direction Request Indicator
5 Write (Use of the Change Direction Request indicator is not recognized by Systems Network Architecture. SNA uses the Signal command as shown in the next alternative.)
SNA Alternative:
6 RECEIVE with RTYPE=DFASY
Signal Command containing request for Change Direction Command Indicator
5 Write
(See Note 2)
Positive Response
5a Read
7 SEND with CONTROL = DATA CHNGDIR = CMD
Data/Change Direction Command Indicator
8 Read
●
●
●
(Logical unit sends next normal-flow message.)
*Or completion of SEND with POST=RESP.
Notes:
1. Indicator could also be received on completion of SEND with POST=RESP.
2. Either ACF/VTAM or the application program responds to the Signal command, depending on whether PROC=SYSRESP or PROC=APPLRESP was specified in the NIB at connection.

Figure C-8. The Application Program and the Logical Unit Use Change-Direction Protocol: (A) Where Only Change Direction Command Indicators Are Used, and (B) Where, in Addition, Change Direction Request Indicator (or Signal Command) Is Used



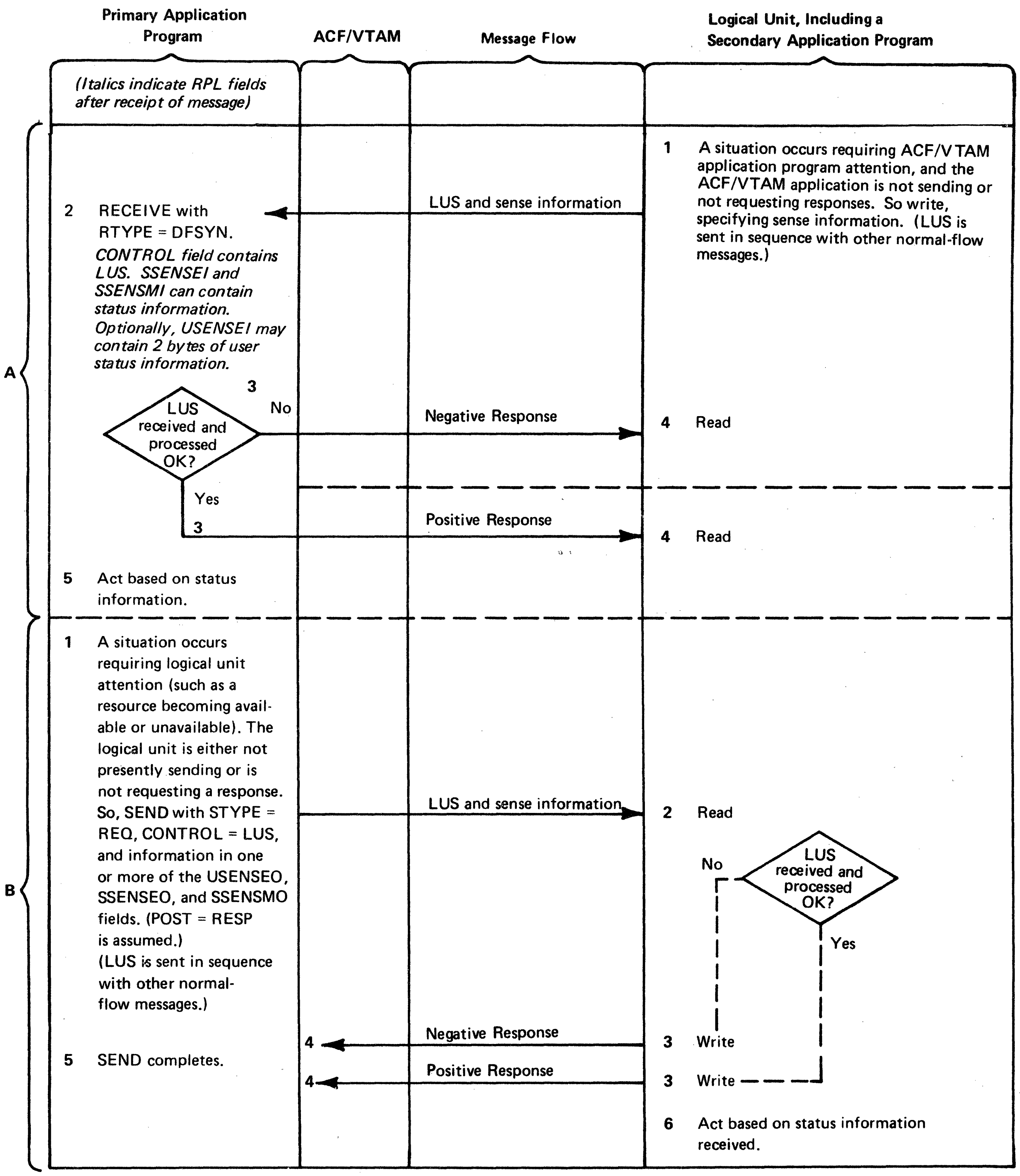
Primary Application Program
ACF/VTAM
Message Flow
Logical Unit (Other Than a Secondary Application Program)
(Italics indicate RPL fields after receipt of message)
(For resynchronization of sequence numbers between a primary and a secondary application program, see Figure C-18.)
Request Recovery
1 Write
2
SCIP exit routine present?
No
Negative Response
3 Read
Yes
2
Positive Response
SCIP Exit Routine scheduled
CONTROL field in read-only RPL contains RQR.
3 Read
4 SESSIONC with STYPE=REQ CONTROL=CLEAR (POST=RESP assumed)
Clear
(Reset sequence numbers to 0)
5 Read (Reset sequence numbers to 0)
8 SESSIONC completed
7
Positive Response
6 Write
9 SESSIONC STYPE=REQ CONTROL=STSN (Example of values: OBSQVAL=100 OBSQAC=SET IBSQVAL=110 IBSQAC=SET)
Set and Test Sequence Numbers
10 Read (Set logical unit's inbound sequence number to 100 and outbound sequence number to 110.)
13 SESSIONC completed
12
Positive Response
11 Write
A dialog may occur between the primary application program and the logical unit to establish sequence numbers acceptable to both. The use of STSN and any dialog is at the descretion of the user. See steps 2-9 in Figure C-3.
14 SESSIONC with STYPE=REQ CONTROL=SDT
Start Data Traffic
15 Read
17 SESSIONC completed
Positive Response
16 Write

Figure C-9. The Primary Application Program Resynchronizes Sequence Numbers with the Logical Unit


Primary Application Program
ACF/VTAM
Message Flow
Logical Unit, Including a Secondary Application Program
(Italics indicate RPL fields after receipt of message)
A
1 Need to send user-defined information ahead of regular data or when quiesced. So write, specifying a Signal command and 4 bytes of information.
Signal
2 RECEIVE with RTYPE = DFASY completes or DFASY exit routine is scheduled. CONTROL field contains SIGNAL. SIGDATA field contains 4 bytes of information.
(See Note)
3
Positive Response
4 Read
5 Perform action related to information.
B
1 Need to send installation defined Signal data-flow control command containing 4 bytes of information to the logical unit. So, SEND with STYPE=REQ, CONTROL=SIGNAL, SIGDATA=4 bytes of information
Signal
2 Read
3
Signal received OK?
(See Note)
No
Yes
Negative Response
4 SEND completes
3
Positive Response
4 SEND completes
5 Perform action related to information.

Note: When a Signal command is received by an application program, either ACF/VTAM or the application program sends the response, depending on whether PROC=SYSRESP or PROC=APPLRESP was specified in the NIB at connection.

Figure C-10. The Application Program and Logical Unit Use the Signal Command: (A) Sent by the Logical Unit, and (B) Sent by the Primary Application Program


Primary Application Program
ACF/VTAM
Message Flow
Logical Unit, Including a Secondary Application Program
(Italics indicate RPL fields after receipt of message)
A
B

1 A situation occurs requiring ACF/VTAM application program attention, and the ACF/VTAM application is not sending or not requesting responses. So write, specifying sense information. (LUS is sent in sequence with other normal-flow messages.)

LUS and sense information

2 RECEIVE with RTYPE = DFSYN. *CONTROL field contains LUS. SSENSEI and SSENSMI can contain status information. Optionally, USENSEI may contain 2 bytes of user status information.*

4 Read

4 Read

5 Act based on status information.

1 A situation occurs requiring logical unit attention (such as a resource becoming available or unavailable). The logical unit is either not presently sending or is not requesting a response. So, SEND with STYPE = REQ, CONTROL = LUS, and information in one or more of the USENSEO, SSENSEO, and SSENSMO fields. (POST = RESP is assumed.) (LUS is sent in sequence with other normal-flow messages.)

2 Read

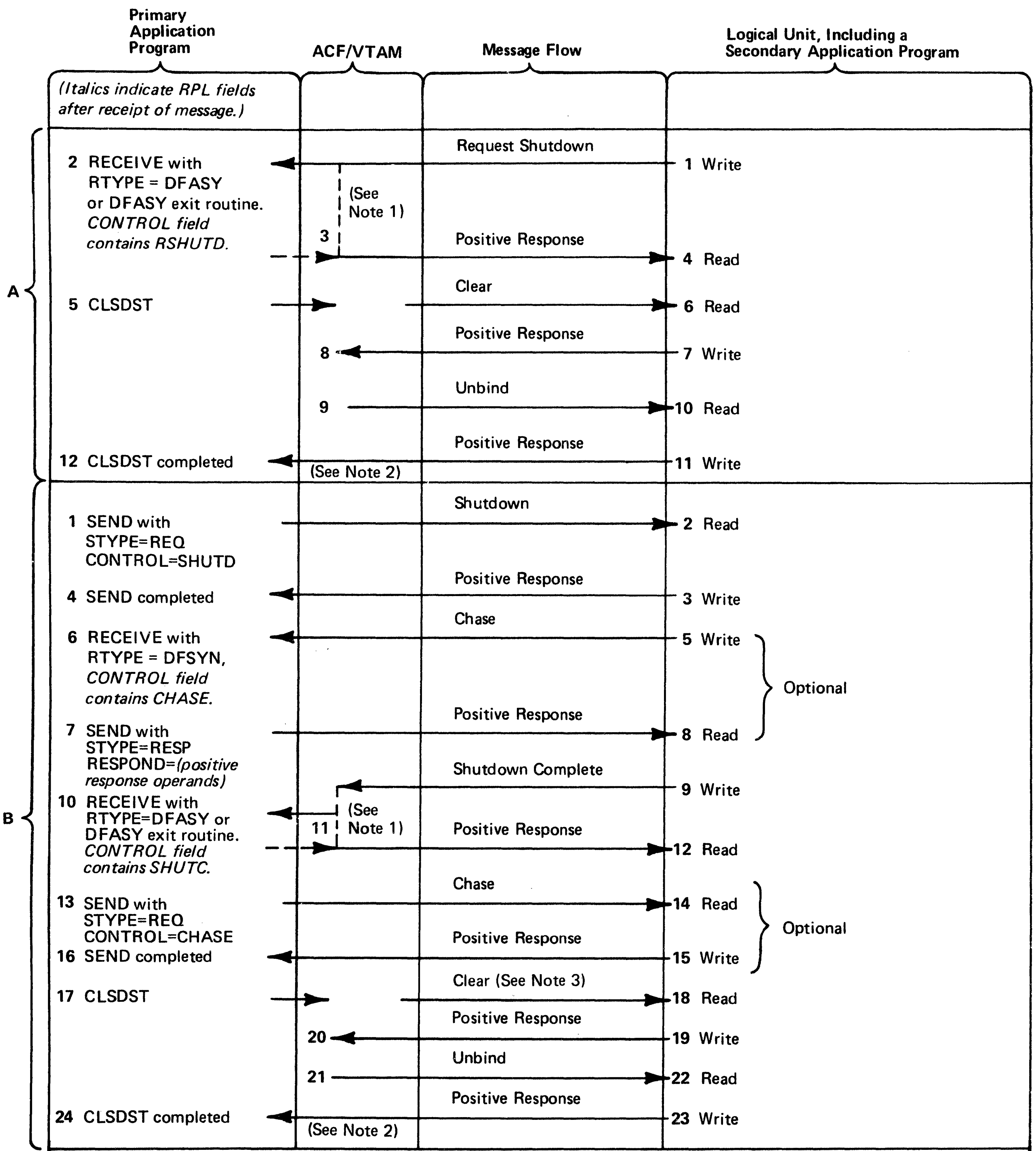
3 Write

3 Write

5 SEND completes.

6 Act based on status information received.

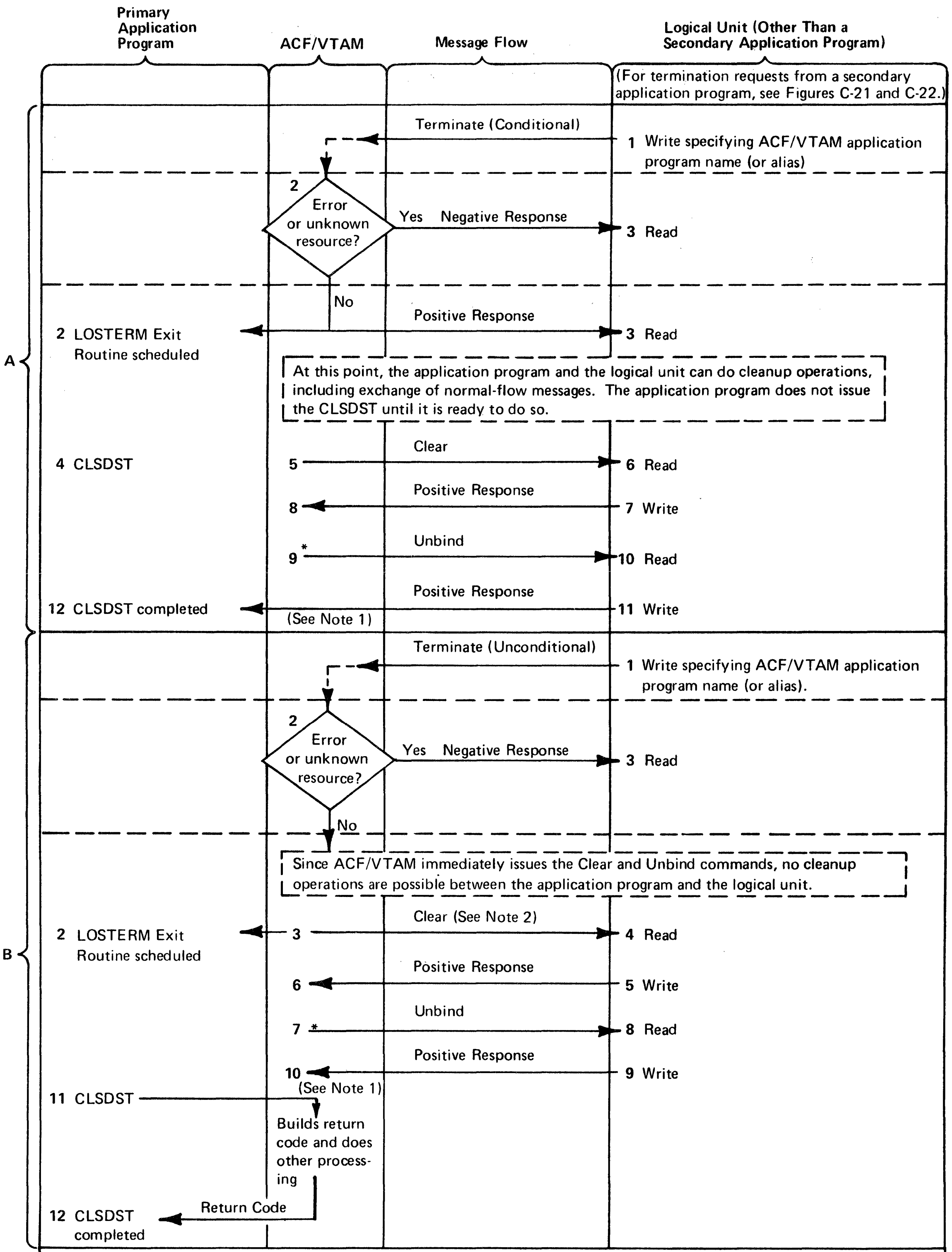
Figure C-11. The Application Program and Logical Unit Use the LUS Command: (A) Sent by the Logical Unit, and (B) Sent by the Primary Application Program



Notes:

1. When an application program receives an expedited-flow command (including Request Shutdown and Shutdown Complete), either ACF/VTAM or the application program sends the response to the command, depending on whether PROC=SYSRESP or PROC=APPLRESP was specified in the NIB at connection.
2. At this point, any outstanding RECEIVE with OPTCD=SPEC is posted complete as "cleared".
3. If permitted by the transmission services profile in the session parameters, the Clear command is sent before the Unbind command if (1) the logical unit is in a different domain from the primary application program, or (2) the logical unit is in the same domain but is attached to a communications controller containing NCP5.

Figure C-12. Operations Are Shut Down in an Orderly Fashion: (A) the Logical Unit Requests Shutdown, and (B) the Primary Application Program Orders Shutdown

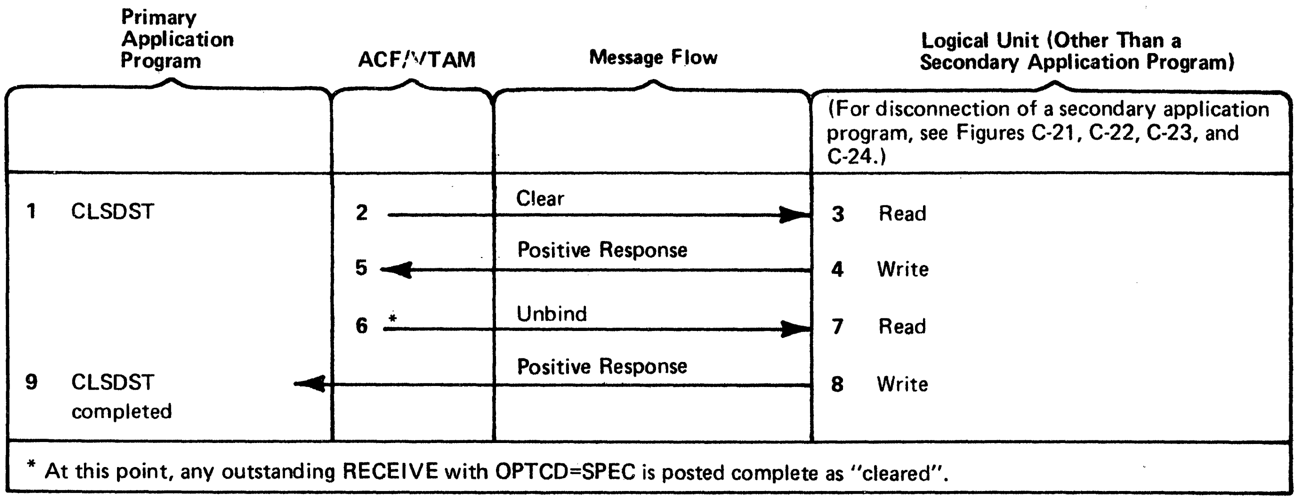


Primary Application Program
ACF/VTAM
Message Flow
Logical Unit (Other Than a Secondary Application Program)
(For termination requests from a secondary application program, see Figures C-21 and C-22.)
A
Terminate (Conditional)
1 Write specifying ACF/VTAM application program name (or alias)
2 Error or unknown resource?
Yes Negative Response
3 Read
No
Positive Response
2 LOSTERM Exit Routine scheduled
3 Read
At this point, the application program and the logical unit can do cleanup operations, including exchange of normal-flow messages. The application program does not issue the CLSDST until it is ready to do so.
4 CLSDST
Clear
5
6 Read
Positive Response
8
7 Write
Unbind
9 *
10 Read
Positive Response
12 CLSDST completed
(See Note 1)
11 Write
B
Terminate (Unconditional)
1 Write specifying ACF/VTAM application program name (or alias).
2 Error or unknown resource?
Yes Negative Response
3 Read
No
Since ACF/VTAM immediately issues the Clear and Unbind commands, no cleanup operations are possible between the application program and the logical unit.
2 LOSTERM Exit Routine scheduled
Clear (See Note 2)
3
4 Read
Positive Response
6
5 Write
Unbind
7 *
8 Read
Positive Response
10
9 Write
(See Note 1)
11 CLSDST
Builds return code and does other processing
Return Code
12 CLSDST completed

Notes:

1. At this point, any outstanding RECEIVE with OPTCD=SPEC is posted complete as "cleared".
2. If permitted by the transmission services profile in the session parameters, the Clear command is sent before the Unbind command if (1) the logical unit is in a different domain from the primary application program, or (2) the logical unit is in the same domain but is attached to a communications controller containing NCP5.

Figure C-13. The Logical Unit Initiates Disconnection: (A) Conditionally, and (B) Unconditionally


Primary Application Program
ACF/VTAM
Message Flow
Logical Unit (Other Than a Secondary Application Program)
(For disconnection of a secondary application program, see Figures C-21, C-22, C-23, and C-24.)
1 CLSDST
2
Clear
3 Read
Positive Response
5
4 Write
6 *
Unbind
7 Read
Positive Response
9 CLSDST completed
8 Write

* At this point, any outstanding RECEIVE with OPTCD=SPEC is posted complete as "cleared".

Figure C-14. The Primary Application Program Disconnects the Logical Unit

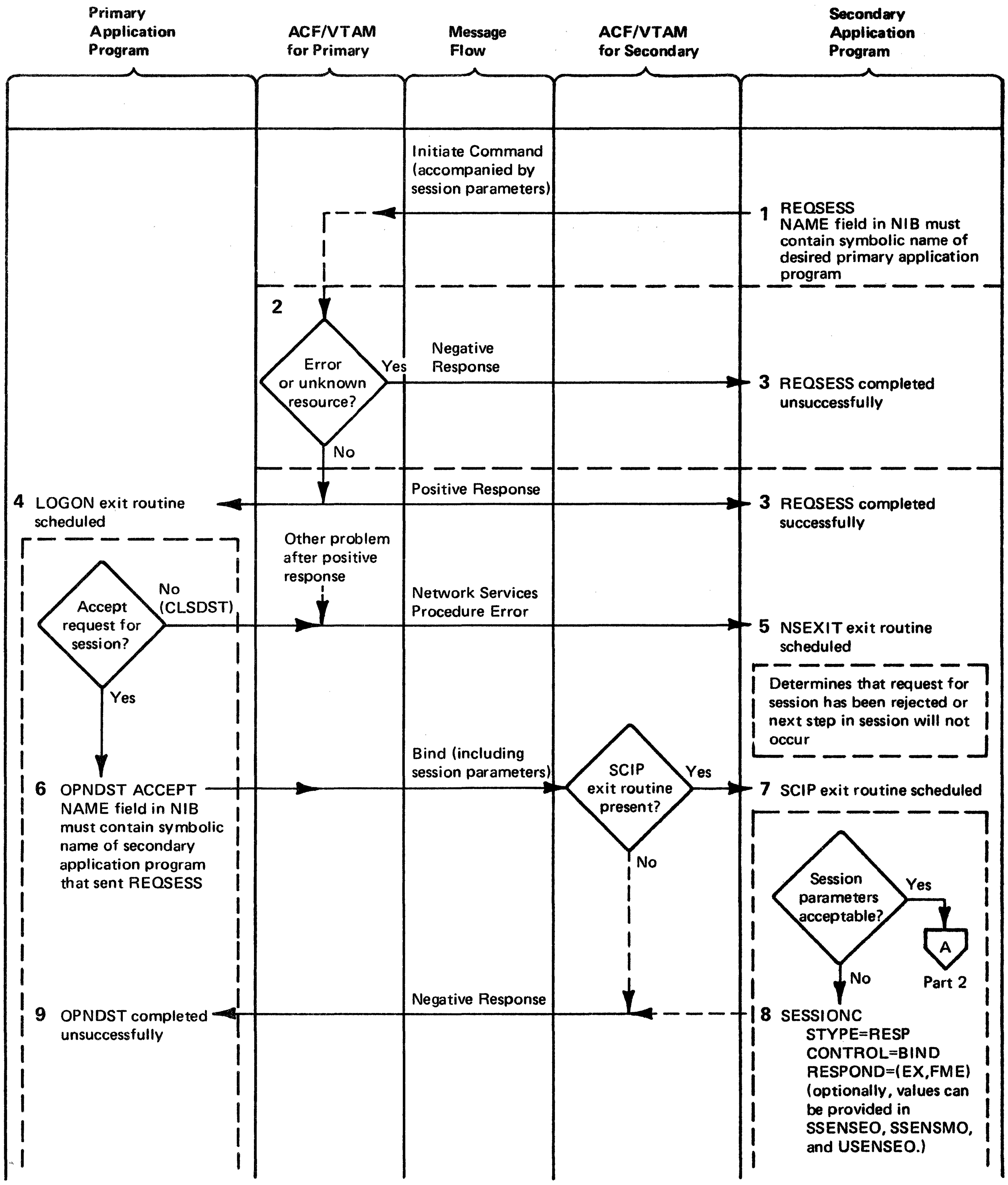

Primary Application Program
ACF/VTAM for Primary
Message Flow
ACF/VTAM for Secondary
Secondary Application Program
Initiate Command (accompanied by session parameters)
1 REQSESS NAME field in NIB must contain symbolic name of desired primary application program
2
Error or unknown resource?
Yes
No
Negative Response
3 REQSESS completed unsuccessfully
Positive Response
4 LOGON exit routine scheduled
3 REQSESS completed successfully
Other problem after positive response
Accept request for session?
No (CLSDST)
Yes
Network Services Procedure Error
5 NSEXIT exit routine scheduled
Determines that request for session has been rejected or next step in session will not occur
Bind (including session parameters)
6 OPNDST ACCEPT NAME field in NIB must contain symbolic name of secondary application program that sent REQSESS
SCIP exit routine present?
Yes
No
7 SCIP exit routine scheduled
Session parameters acceptable?
Yes
A
Part 2
No
Negative Response
9 OPNDST completed unsuccessfully
8 SESSIONC STYPE=RESP CONTROL=BIND RESPOND=(EX,FME) (optionally, values can be provided in SSENSEO, SSENSMO, and USENSEO.)

Figure C-15 (Part 1 of 2). The Secondary Application Program Reqeusts Connection to the Primary Application Program

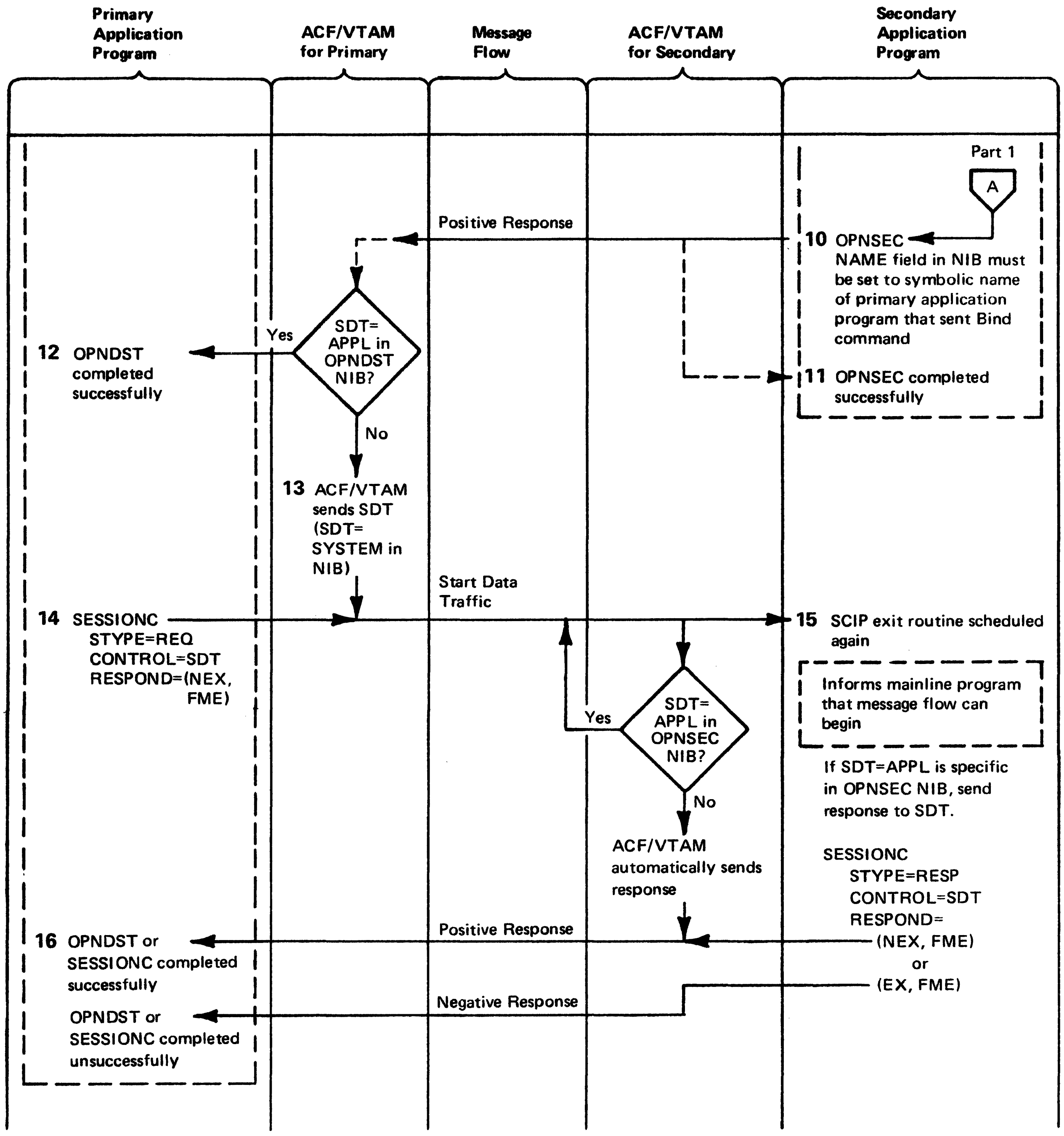
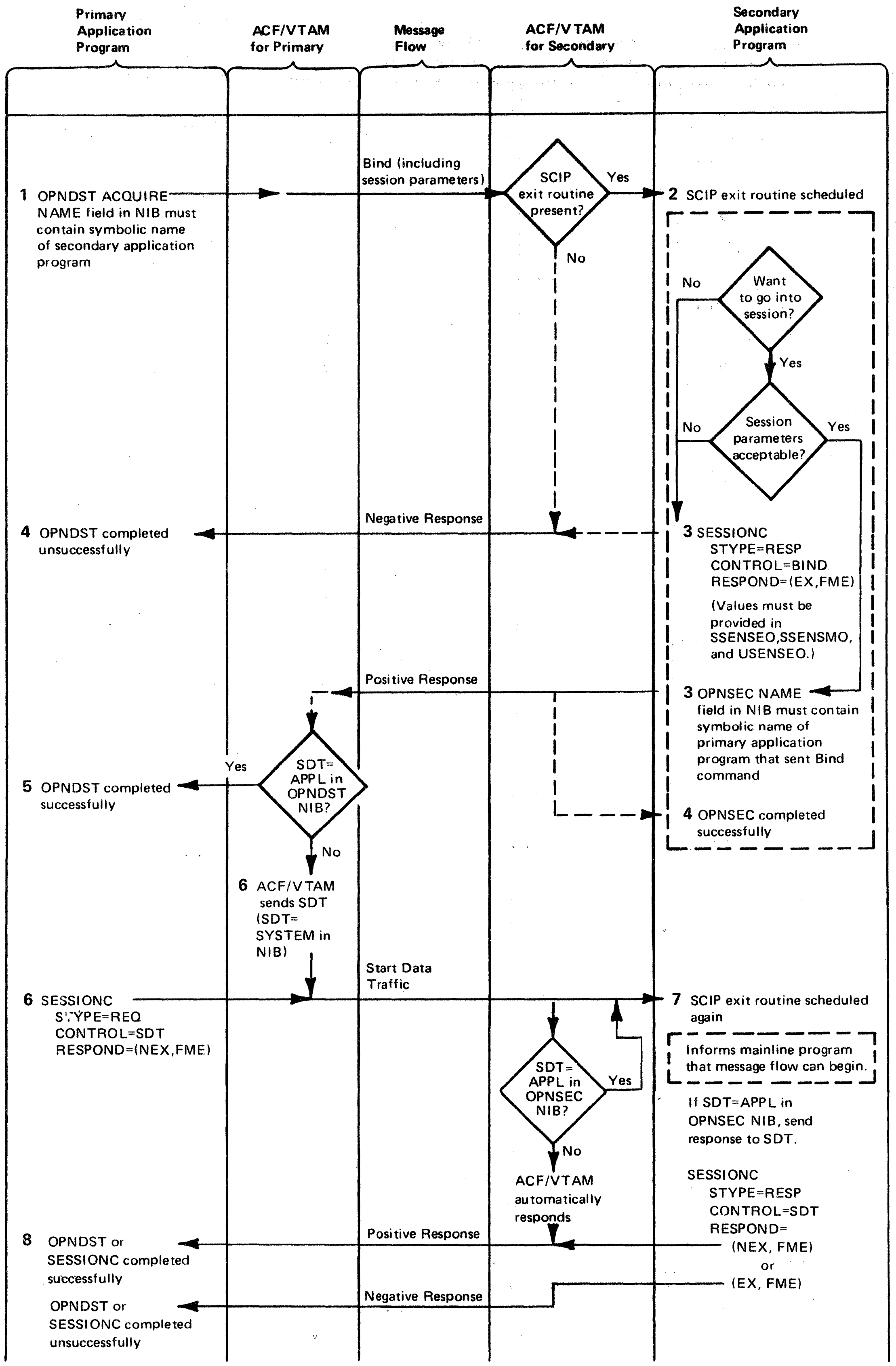

Primary Application Program
ACF/VTAM for Primary
Message Flow
ACF/VTAM for Secondary
Secondary Application Program
Part 1
A
Positive Response
10 OPNSEC
NAME field in NIB must be set to symbolic name of primary application program that sent Bind command
Yes
SDT= APPL in OPNDST NIB?
12 OPNDST completed successfully
11 OPNSEC completed successfully
No
13 ACF/VTAM sends SDT (SDT= SYSTEM in NIB)
Start Data Traffic
14 SESSIONC STYPE=REQ CONTROL=SDT RESPOND=(NEX, FME)
15 SCIP exit routine scheduled again
Informs mainline program that message flow can begin
Yes
SDT= APPL in OPNSEC NIB?
If SDT=APPL is specific in OPNSEC NIB, send response to SDT.
No
ACF/VTAM automatically sends response
SESSIONC STYPE=RESP CONTROL=SDT RESPOND= (NEX, FME) or (EX, FME)
16 OPNDST or SESSIONC completed successfully
Positive Response
Negative Response
OPNDST or SESSIONC completed unsuccessfully

Figure C-15 (Part 2 of 2). The Secondary Application Program Requests Connection to the Primary Application Program



Primary Application Program
ACF/VTAM for Primary
Message Flow
ACF/VTAM for Secondary
Secondary Application Program
1 OPNDST ACQUIRE NAME field in NIB must contain symbolic name of secondary application program
Bind (including session parameters)
SCIP exit routine present?
Yes
No
2 SCIP exit routine scheduled
Want to go into session?
No
Yes
Session parameters acceptable?
No
Yes
Negative Response
4 OPNDST completed unsuccessfully
3 SESSIONC STYPE=RESP CONTROL=BIND RESPOND=(EX,FME)
(Values must be provided in SSENSEO,SSENSMO, and USENSEO.)
Positive Response
3 OPNSEC NAME field in NIB must contain symbolic name of primary application program that sent Bind command
Yes
SDT= APPL in OPNDST NIB?
5 OPNDST completed successfully
4 OPNSEC completed successfully
No
6 ACF/VTAM sends SDT (SDT= SYSTEM in NIB)
Start Data Traffic
6 SESSIONC STYPE=REQ CONTROL=SDT RESPOND=(NEX,FME)
7 SCIP exit routine scheduled again
Informs mainline program that message flow can begin.
SDT= APPL in OPNSEC NIB?
Yes
No
If SDT=APPL in OPNSEC NIB, send response to SDT.
ACF/VTAM automatically responds
SESSIONC STYPE=RESP CONTROL=SDT RESPOND= (NEX, FME) or (EX, FME)
Positive Response
8 OPNDST or SESSIONC completed successfully
Negative Response
OPNDST or SESSIONC completed unsuccessfully

Figure C-16. The Primary Application Program Acquires the Secondary Application Program

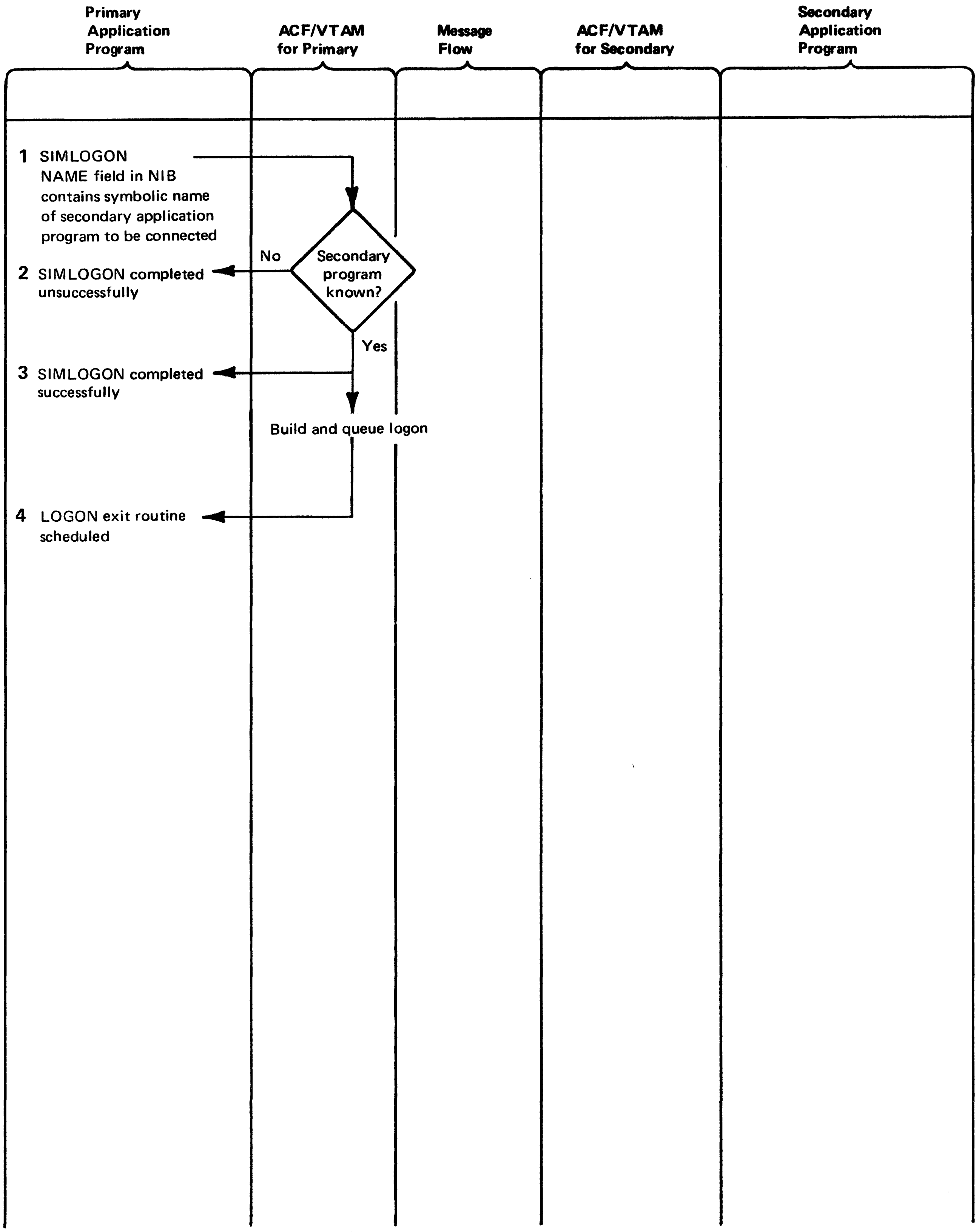
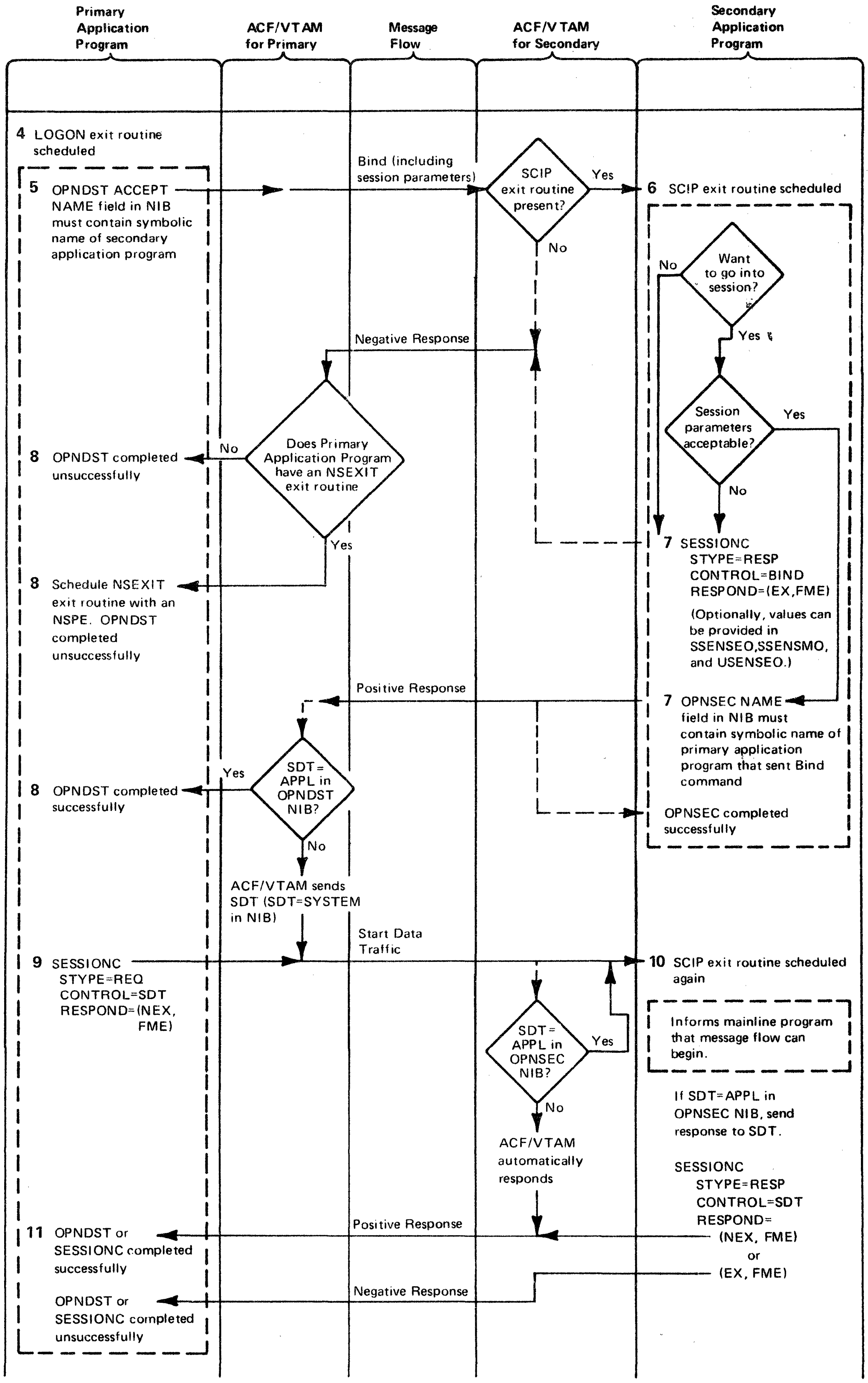

Primary Application Program
ACF/VTAM for Primary
Message Flow
ACF/VTAM for Secondary
Secondary Application Program
1 SIMLOGON NAME field in NIB contains symbolic name of secondary application program to be connected
Secondary program known?
No
2 SIMLOGON completed unsuccessfully
Yes
3 SIMLOGON completed successfully
Build and queue logon
4 LOGON exit routine scheduled

Figure C-17 (Part 1 of 2). The Primary Application Program Issues a SIMLOGON Macro Instruction to Acquire the Secondary Application Program



Primary Application Program
ACF/VTAM for Primary
Message Flow
ACF/VTAM for Secondary
Secondary Application Program
4 LOGON exit routine scheduled
5 OPNDST ACCEPT NAME field in NIB must contain symbolic name of secondary application program
Bind (including session parameters)
SCIP exit routine present?
Yes
No
6 SCIP exit routine scheduled
Want to go into session?
No
Yes
Negative Response
Session parameters acceptable?
Yes
No
Does Primary Application Program have an NSEXIT exit routine
No
Yes
8 OPNDST completed unsuccessfully
8 Schedule NSEXIT exit routine with an NSPE. OPNDST completed unsuccessfully
7 SESSIONC STYPE=RESP CONTROL=BIND RESPOND=(EX,FME)
(Optionally, values can be provided in SSENSEO,SSENSMO, and USENSEO.)
Positive Response
7 OPNSEC NAME field in NIB must contain symbolic name of primary application program that sent Bind command
SDT= APPL in OPNDST NIB?
Yes
No
8 OPNDST completed successfully
OPNSEC completed successfully
ACF/VTAM sends SDT (SDT=SYSTEM in NIB)
Start Data Traffic
9 SESSIONC STYPE=REQ CONTROL=SDT RESPOND=(NEX, FME)
10 SCIP exit routine scheduled again
Informs mainline program that message flow can begin.
SDT= APPL in OPNSEC NIB?
Yes
No
ACF/VTAM automatically responds
If SDT=APPL in OPNSEC NIB, send response to SDT.
SESSIONC STYPE=RESP CONTROL=SDT RESPOND= (NEX, FME) or (EX, FME)
Positive Response
Negative Response
11 OPNDST or SESSIONC completed successfully
OPNDST or SESSIONC completed unsuccessfully

Figure C-17 (Part 2 of 2). The Primary Application Program Issues a SIMLOGON Macro Instruction to Acquire the Secondary Application Program

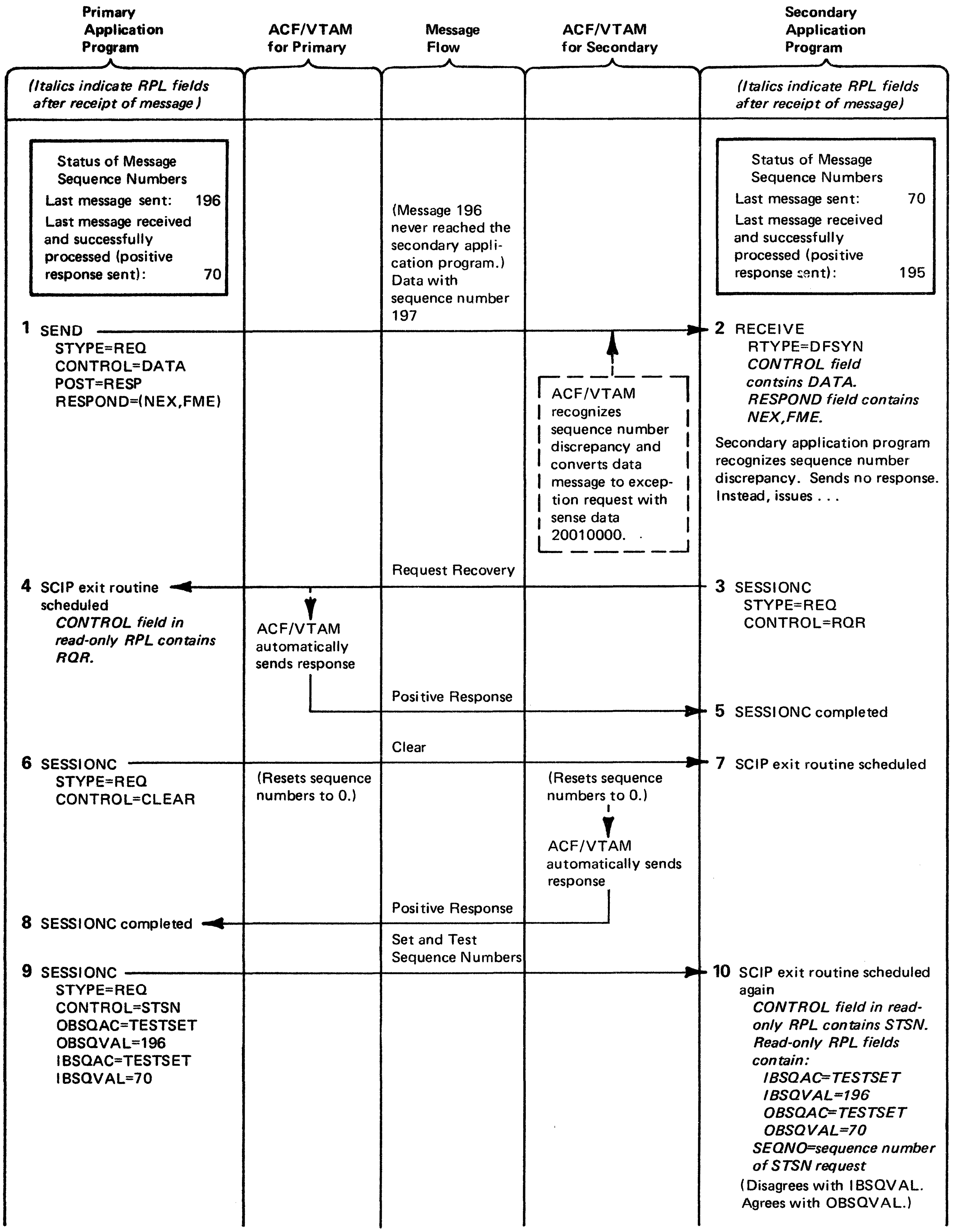

Primary Application Program
ACF/VTAM for Primary
Message Flow
ACF/VTAM for Secondary
Secondary Application Program
(Italics indicate RPL fields after receipt of message)
(Italics indicate RPL fields after receipt of message)
Status of Message Sequence Numbers
Last message sent: 196
Last message received and successfully processed (positive response sent): 70
(Message 196 never reached the secondary application program.) Data with sequence number 197
Status of Message Sequence Numbers
Last message sent: 70
Last message received and successfully processed (positive response sent): 195
1 SEND STYPE=REQ CONTROL=DATA POST=RESP RESPOND=(NEX,FME)
2 RECEIVE RTYPE=DFSYN CONTROL field contsins DATA. RESPOND field contains NEX,FME.
ACF/VTAM recognizes sequence number discrepancy and converts data message to exception request with sense data 20010000.
Secondary application program recognizes sequence number discrepancy. Sends no response. Instead, issues . . .
Request Recovery
4 SCIP exit routine scheduled CONTROL field in read-only RPL contains RQR.
3 SESSIONC STYPE=REQ CONTROL=RQR
ACF/VTAM automatically sends response
Positive Response
5 SESSIONC completed
Clear
6 SESSIONC STYPE=REQ CONTROL=CLEAR
7 SCIP exit routine scheduled
(Resets sequence numbers to 0.)
(Resets sequence numbers to 0.)
ACF/VTAM automatically sends response
Positive Response
8 SESSIONC completed
Set and Test Sequence Numbers
9 SESSIONC STYPE=REQ CONTROL=STSN OBSQAC=TESTSET OBSQVAL=196 IBSQAC=TESTSET IBSQVAL=70
10 SCIP exit routine scheduled again
CONTROL field in read-only RPL contains STSN.
Read-only RPL fields contain:
IBSQAC=TESTSET
IBSQVAL=196
OBSQAC=TESTSET
OBSQVAL=70
SEQNO=sequence number of STSN request
(Disagrees with IBSQVAL. Agrees with OBSQVAL.)

Figure C-18 (Part 1 of 2). The Primary Application Program Resynchronizes Sequence Numbers with the Secondary Application Program

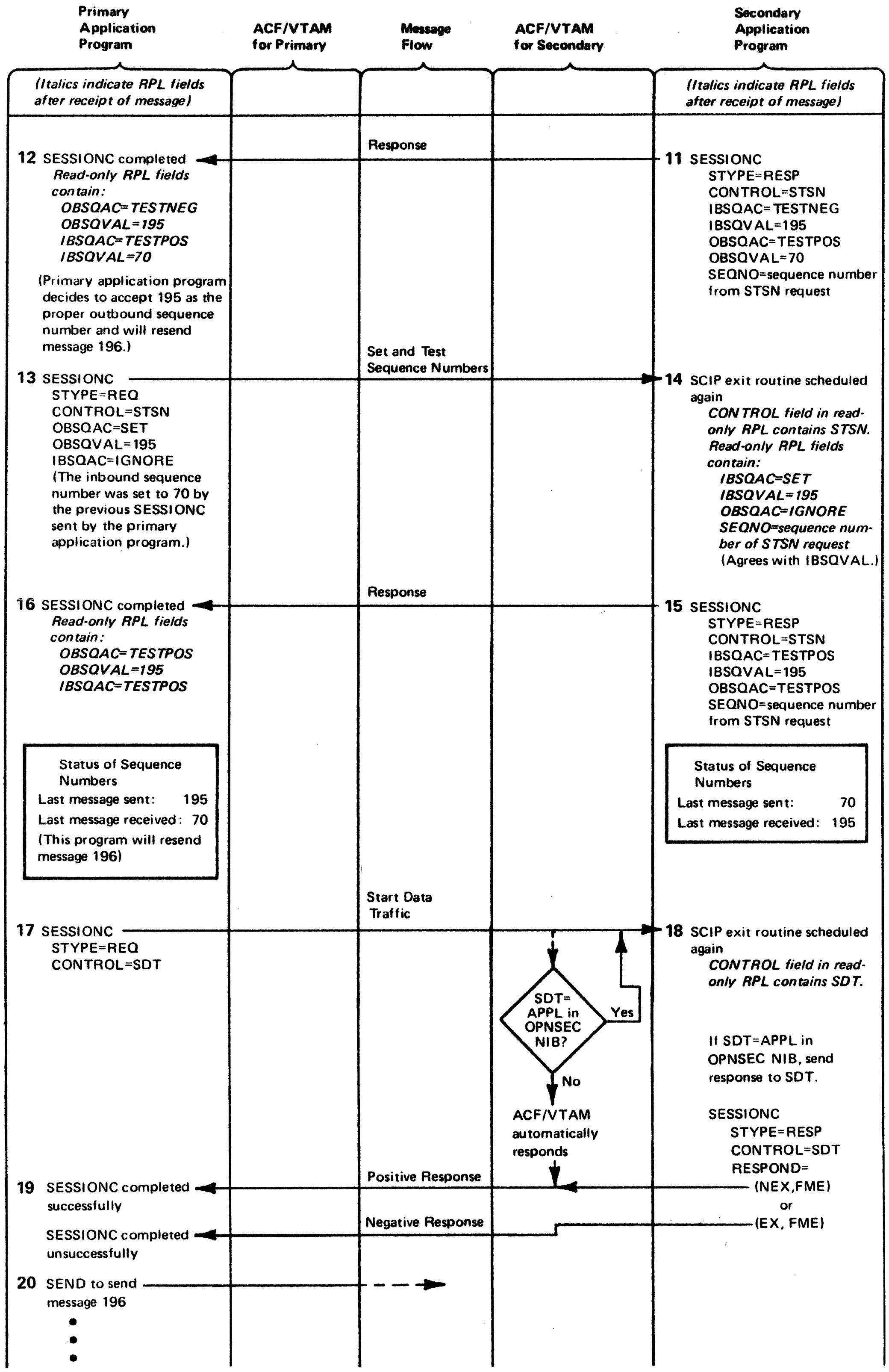
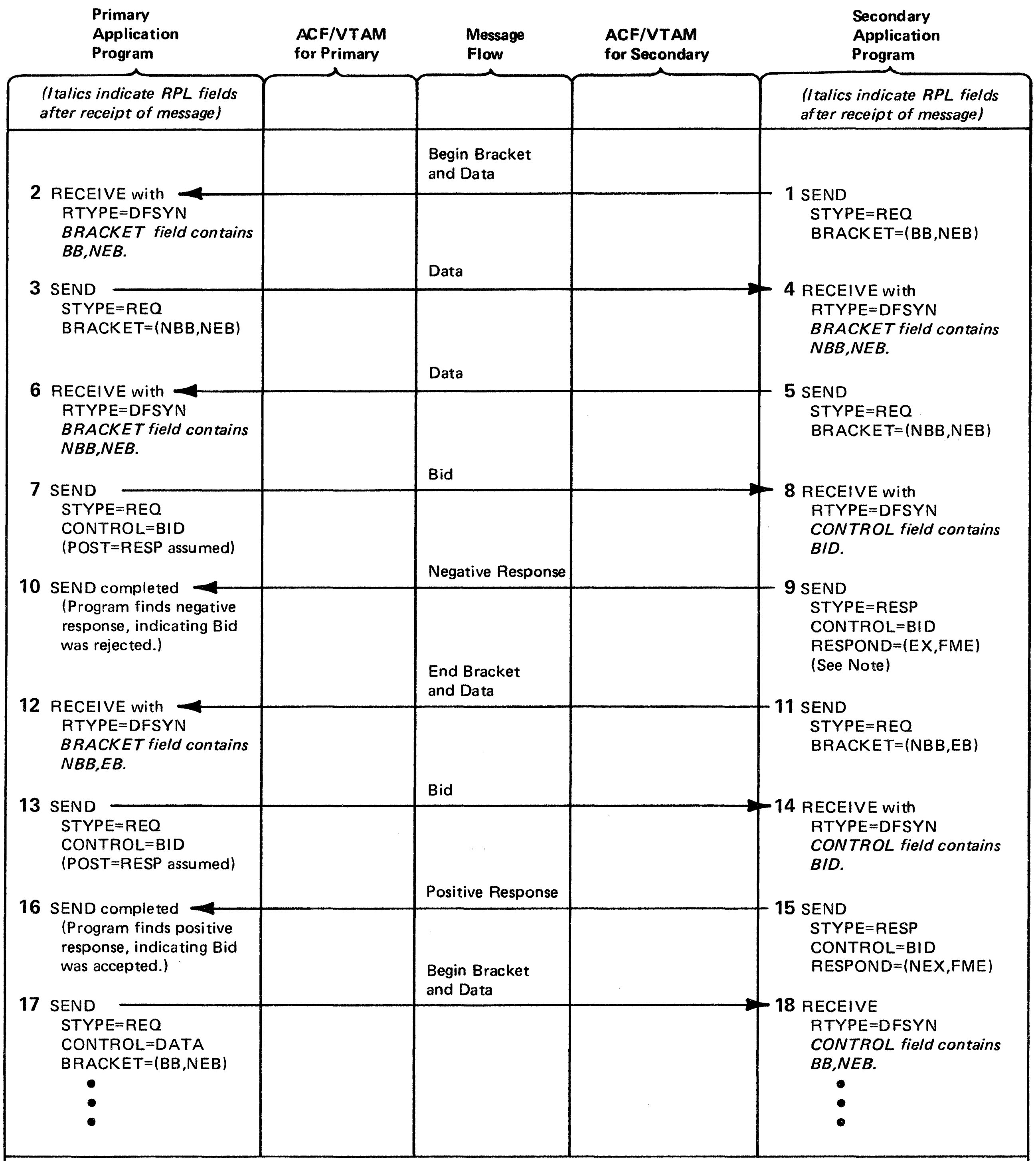
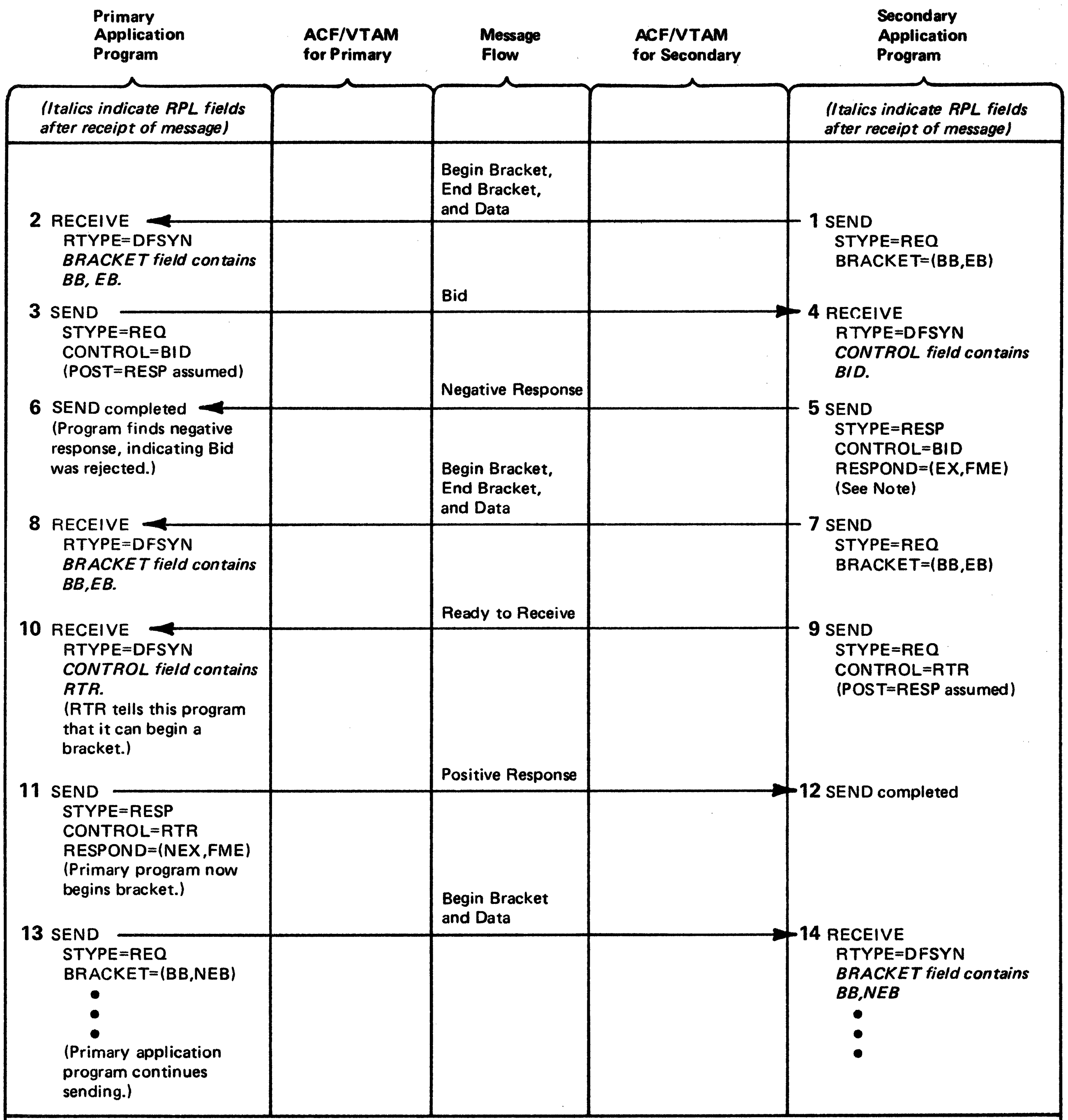

Primary Application Program
ACF/VTAM for Primary
Message Flow
ACF/VTAM for Secondary
Secondary Application Program
(Italics indicate RPL fields after receipt of message)
(Italics indicate RPL fields after receipt of message)
Response
12 SESSIONC completed
Read-only RPL fields contain:
OBSQAC=TESTNEG
OBSQVAL=195
IBSQAC=TESTPOS
IBSQVAL=70
(Primary application program decides to accept 195 as the proper outbound sequence number and will resend message 196.)
11 SESSIONC
STYPE=RESP
CONTROL=STSN
IBSQAC=TESTNEG
IBSQVAL=195
OBSQAC=TESTPOS
OBSQVAL=70
SEQNO=sequence number from STSN request
Set and Test Sequence Numbers
13 SESSIONC
STYPE=REQ
CONTROL=STSN
OBSQAC=SET
OBSQVAL=195
IBSQAC=IGNORE
(The inbound sequence number was set to 70 by the previous SESSIONC sent by the primary application program.)
14 SCIP exit routine scheduled again
CONTROL field in read-only RPL contains STSN.
Read-only RPL fields contain:
IBSQAC=SET
IBSQVAL=195
OBSQAC=IGNORE
SEQNO=sequence number of STSN request
(Agrees with IBSQVAL.)
Response
16 SESSIONC completed
Read-only RPL fields contain:
OBSQAC=TESTPOS
OBSQVAL=195
IBSQAC=TESTPOS
15 SESSIONC
STYPE=RESP
CONTROL=STSN
IBSQAC=TESTPOS
IBSQVAL=195
OBSQAC=TESTPOS
SEQNO=sequence number from STSN request
Status of Sequence Numbers
Last message sent: 195
Last message received: 70
(This program will resend message 196)
Status of Sequence Numbers
Last message sent: 70
Last message received: 195
Start Data Traffic
17 SESSIONC
STYPE=REQ
CONTROL=SDT
18 SCIP exit routine scheduled again
CONTROL field in read-only RPL contains SDT.
SDT=APPL in OPNSEC NIB?
Yes
No
If SDT=APPL in OPNSEC NIB, send response to SDT.
ACF/VTAM automatically responds
SESSIONC
STYPE=RESP
CONTROL=SDT
RESPOND=
(NEX,FME)
or
(EX, FME)
Positive Response
19 SESSIONC completed successfully
Negative Response
SESSIONC completed unsuccessfully
20 SEND to send message 196
•
•
•

Figure C-18 (Part 2 of 2). The Primary Application Program Resynchronizes Sequence Numbers with the Secondary Application Program


Primary Application Program
ACF/VTAM for Primary
Message Flow
ACF/VTAM for Secondary
Secondary Application Program
(Italics indicate RPL fields after receipt of message)
(Italics indicate RPL fields after receipt of message)
Begin Bracket and Data
2 RECEIVE with RTYPE=DFSYN *BRACKET field contains BB,NEB.*
1 SEND STYPE=REQ BRACKET=(BB,NEB)
Data
3 SEND STYPE=REQ BRACKET=(NBB,NEB)
4 RECEIVE with RTYPE=DFSYN *BRACKET field contains NBB,NEB.*
Data
6 RECEIVE with RTYPE=DFSYN *BRACKET field contains NBB,NEB.*
5 SEND STYPE=REQ BRACKET=(NBB,NEB)
Bid
7 SEND STYPE=REQ CONTROL=BID (POST=RESP assumed)
8 RECEIVE with RTYPE=DFSYN *CONTROL field contains BID.*
Negative Response
10 SEND completed (Program finds negative response, indicating Bid was rejected.)
9 SEND STYPE=RESP CONTROL=BID RESPOND=(EX,FME) (See Note)
End Bracket and Data
12 RECEIVE with RTYPE=DFSYN *BRACKET field contains NBB,EB.*
11 SEND STYPE=REQ BRACKET=(NBB,EB)
Bid
13 SEND STYPE=REQ CONTROL=BID (POST=RESP assumed)
14 RECEIVE with RTYPE=DFSYN *CONTROL field contains BID.*
Positive Response
16 SEND completed (Program finds positive response, indicating Bid was accepted.)
15 SEND STYPE=RESP CONTROL=BID RESPOND=(NEX,FME)
Begin Bracket and Data
17 SEND STYPE=REQ CONTROL=DATA BRACKET=(BB,NEB)
18 RECEIVE RTYPE=DFSYN *CONTROL field contains BB,NEB.*
•
•
•
•
•
•

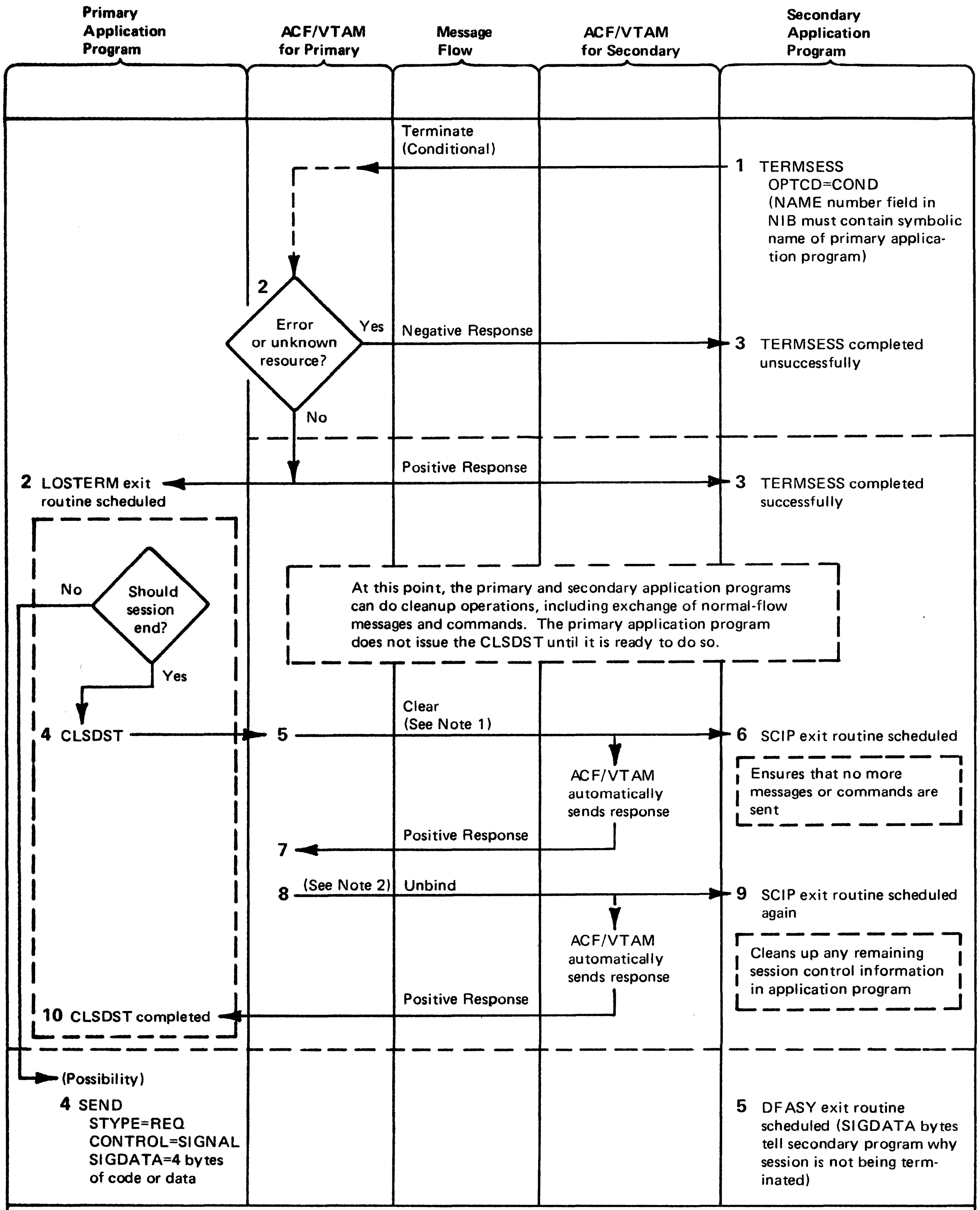
Note: The CONTROL=BID operand is not needed if the RPL being used for this SEND is the same as the one that was used to receive the Bid.

Figure C-19. A Primary Application Program and Secondary Application Program Use Bracket Protocol (a Bid Command Is First Rejected, Then Accepted)


Primary Application Program
ACF/VTAM for Primary
Message Flow
ACF/VTAM for Secondary
Secondary Application Program
(Italics indicate RPL fields after receipt of message)
(Italics indicate RPL fields after receipt of message)
Begin Bracket, End Bracket, and Data
2 RECEIVE RTYPE=DFSYN *BRACKET field contains BB, EB.*
1 SEND STYPE=REQ BRACKET=(BB,EB)
Bid
3 SEND STYPE=REQ CONTROL=BID (POST=RESP assumed)
4 RECEIVE RTYPE=DFSYN *CONTROL field contains BID.*
Negative Response
6 SEND completed (Program finds negative response, indicating Bid was rejected.)
5 SEND STYPE=RESP CONTROL=BID RESPOND=(EX,FME) (See Note)
Begin Bracket, End Bracket, and Data
8 RECEIVE RTYPE=DFSYN *BRACKET field contains BB,EB.*
7 SEND STYPE=REQ BRACKET=(BB,EB)
Ready to Receive
10 RECEIVE RTYPE=DFSYN *CONTROL field contains RTR.* (RTR tells this program that it can begin a bracket.)
9 SEND STYPE=REQ CONTROL=RTR (POST=RESP assumed)
Positive Response
11 SEND STYPE=RESP CONTROL=RTR RESPOND=(NEX,FME) (Primary program now begins bracket.)
12 SEND completed
Begin Bracket and Data
13 SEND STYPE=REQ BRACKET=(BB,NEB) • • • (Primary application program continues sending.)
14 RECEIVE RTYPE=DFSYN *BRACKET field contains BB,NEB* • • •

Note: The CONTROL=BID operand is not needed if the RPL being used for this SEND is the same as the one that was used to receive the Bid.

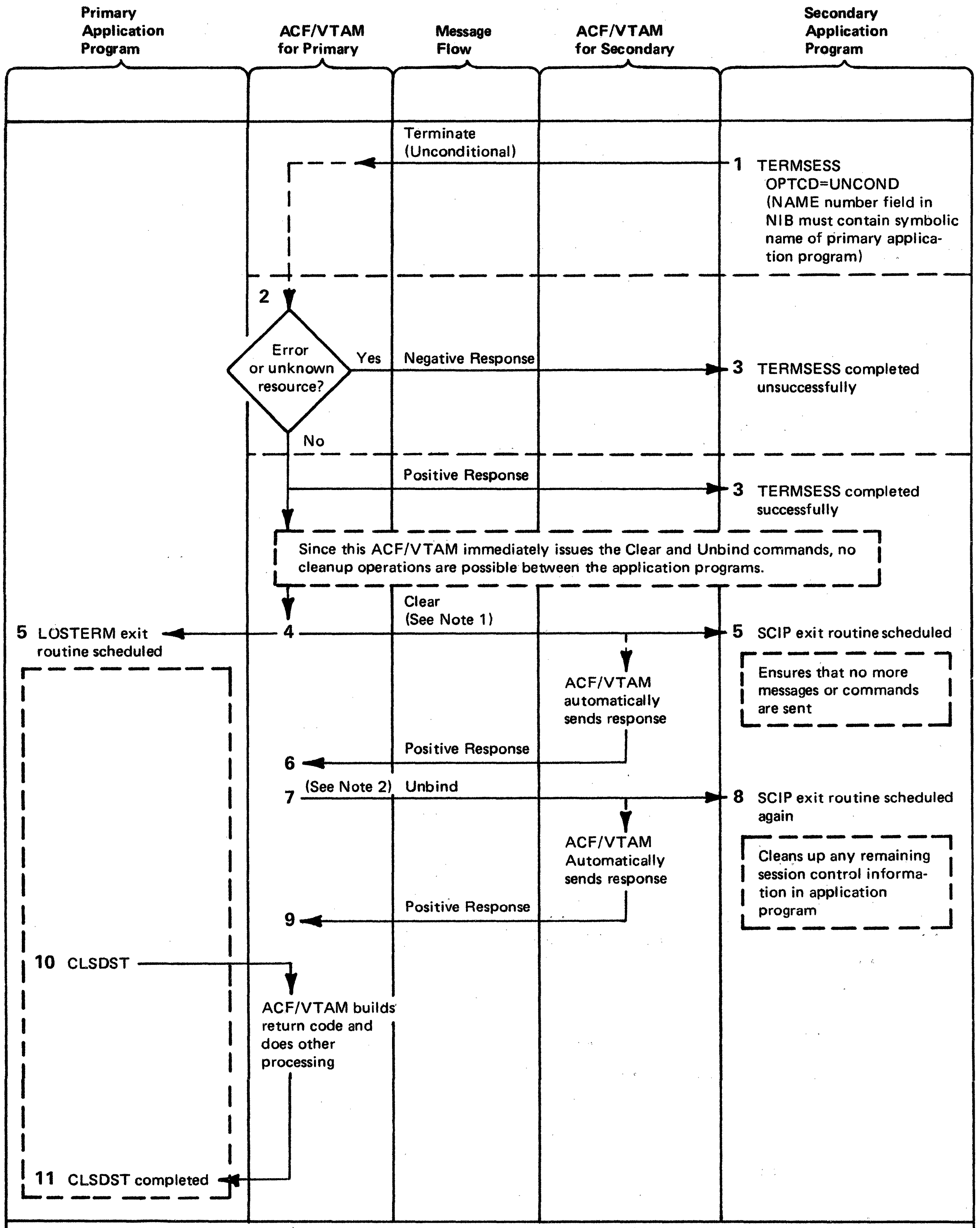
Figure C-20. A Primary Application Program and Secondary Application Program Use Bracket Protocol (Bid by Primary Program Is Rejected, But a Ready to Receive Command Follows)


Primary Application Program
ACF/VTAM for Primary
Message Flow
ACF/VTAM for Secondary
Secondary Application Program
Terminate (Conditional)
1 TERMSESS OPTCD=COND (NAME number field in NIB must contain symbolic name of primary application program)
2
Error or unknown resource?
Yes
Negative Response
3 TERMSESS completed unsuccessfully
No
Positive Response
2 LOSTERM exit routine scheduled
3 TERMSESS completed successfully
No
Should session end?
Yes
At this point, the primary and secondary application programs can do cleanup operations, including exchange of normal-flow messages and commands. The primary application program does not issue the CLSDST until it is ready to do so.
4 CLSDST
5
Clear (See Note 1)
6 SCIP exit routine scheduled
ACF/VTAM automatically sends response
Ensures that no more messages or commands are sent
Positive Response
7
8
(See Note 2)
Unbind
9 SCIP exit routine scheduled again
ACF/VTAM automatically sends response
Cleans up any remaining session control information in application program
Positive Response
10 CLSDST completed
(Possibility)
4 SEND STYPE=REQ CONTROL=SIGNAL SIGDATA=4 bytes of code or data
5 DFASY exit routine scheduled (SIGDATA bytes tell secondary program why session is not being terminated)

Notes:

1. If permitted by the transmission services profile in the session parameters, the Clear command is sent before the Unbind command if the secondary application program is in a different domain from the primary application program.
2. At this point, any outstanding RECEIVE with OPTCD=SPEC is posted complete as "cleared".

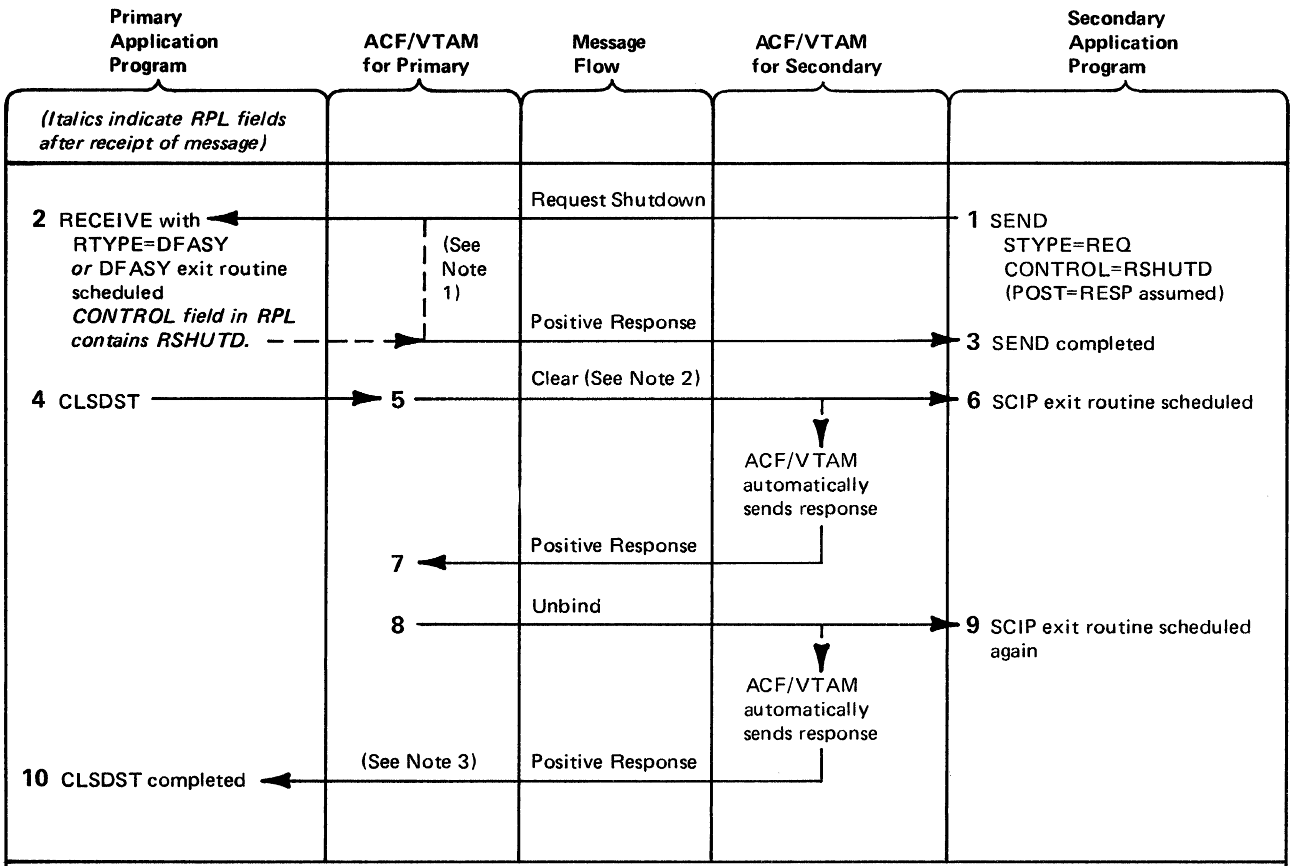
Figure C-21. The Secondary Application Program Sends a Conditional Request for Disconnection


Primary Application Program
ACF/VTAM for Primary
Message Flow
ACF/VTAM for Secondary
Secondary Application Program
Terminate (Unconditional)
1 TERMSESS OPTCD=UNCOND (NAME number field in NIB must contain symbolic name of primary application program)
2
Error or unknown resource?
Yes
Negative Response
3 TERMSESS completed unsuccessfully
No
Positive Response
3 TERMSESS completed successfully
Since this ACF/VTAM immediately issues the Clear and Unbind commands, no cleanup operations are possible between the application programs.
Clear (See Note 1)
5 LOSTERM exit routine scheduled
4
5 SCIP exit routine scheduled
Ensures that no more messages or commands are sent
ACF/VTAM automatically sends response
Positive Response
6
7
(See Note 2)
Unbind
8 SCIP exit routine scheduled again
ACF/VTAM Automatically sends response
Cleans up any remaining session control information in application program
Positive Response
9
10 CLSDST
ACF/VTAM builds return code and does other processing
11 CLSDST completed

Notes:

1. If permitted by the transmission services profile in the session parameters, the Clear command is sent before the Unbind command if the secondary application program is in a different domain from the primary application program.
2. At this point, any outstanding RECEIVE with OPNDST=SPEC is posted complete as cleared.

Figure C-22. The Secondary Application Program Sends an Unconditional Request for Disconnection


Primary Application Program
ACF/VTAM for Primary
Message Flow
ACF/VTAM for Secondary
Secondary Application Program
(Italics indicate RPL fields after receipt of message)
Request Shutdown
2 RECEIVE with RTYPE=DFASY or DFASY exit routine scheduled CONTROL field in RPL contains RSHUTD.
(See Note 1)
1 SEND STYPE=REQ CONTROL=RSHUTD (POST=RESP assumed)
Positive Response
3 SEND completed
Clear (See Note 2)
4 CLSDST
5
6 SCIP exit routine scheduled
ACF/VTAM automatically sends response
Positive Response
7
Unbind
8
9 SCIP exit routine scheduled again
ACF/VTAM automatically sends response
(See Note 3)
Positive Response
10 CLSDST completed

Notes:

1. When an application program receives an expedited-flow command (including the Request Shutdown command), either ACF/VTAM or the application program sends the response to the command, depending on whether PROC=SYSRESP or PROC=APPLRESP was specified in the NIB at connection.
2. If permitted by the transmission services profile in the session parameters, the Clear command is sent before the Unbind command if the secondary application program is in a different domain from the primary application program.
3. At this point, any outstanding RECEIVE with OPTCD=SPEC is posted complete as "cleared".

Figure C-23. The Secondary Application Program Sends a Request Shutdown Command

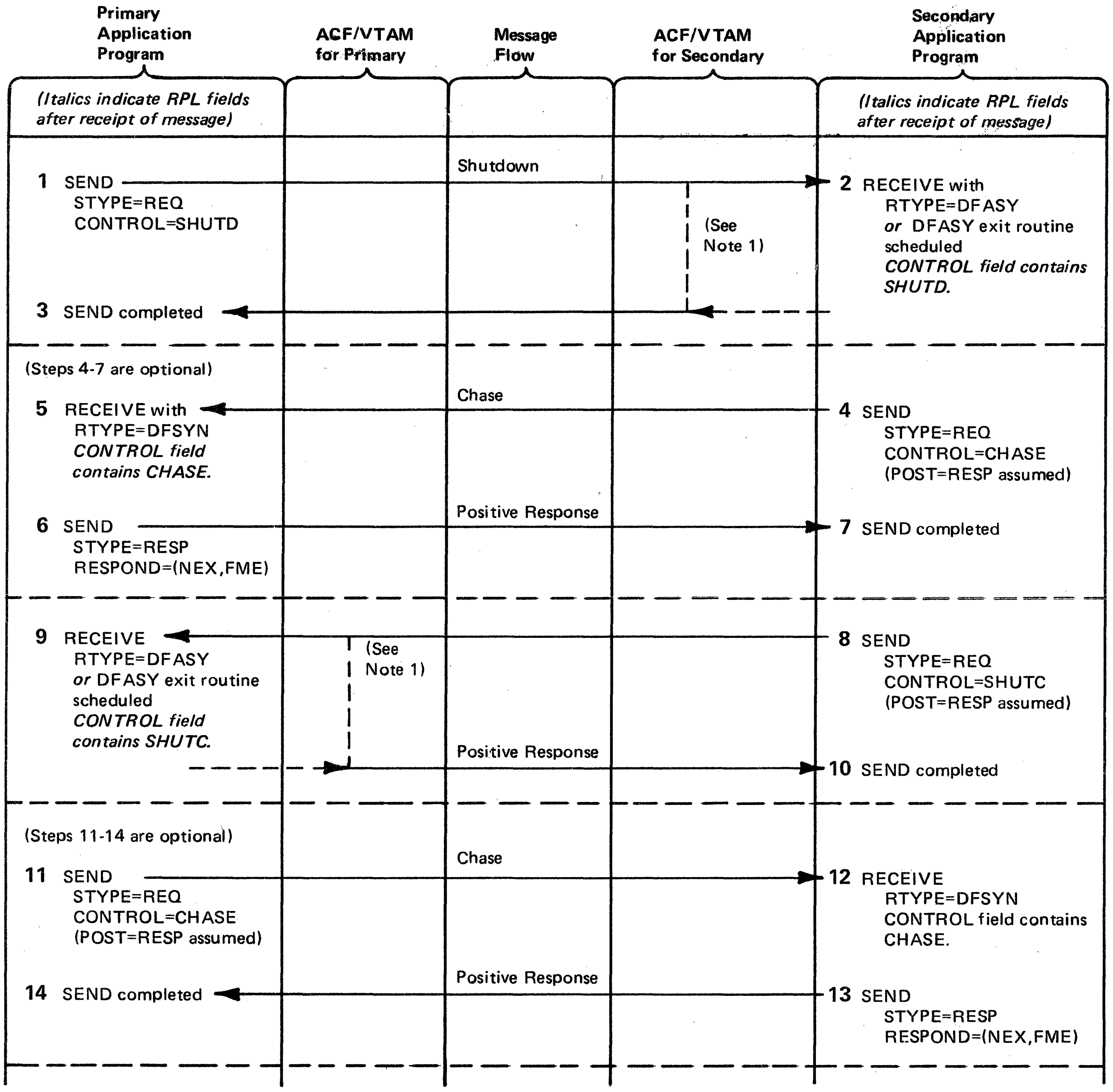
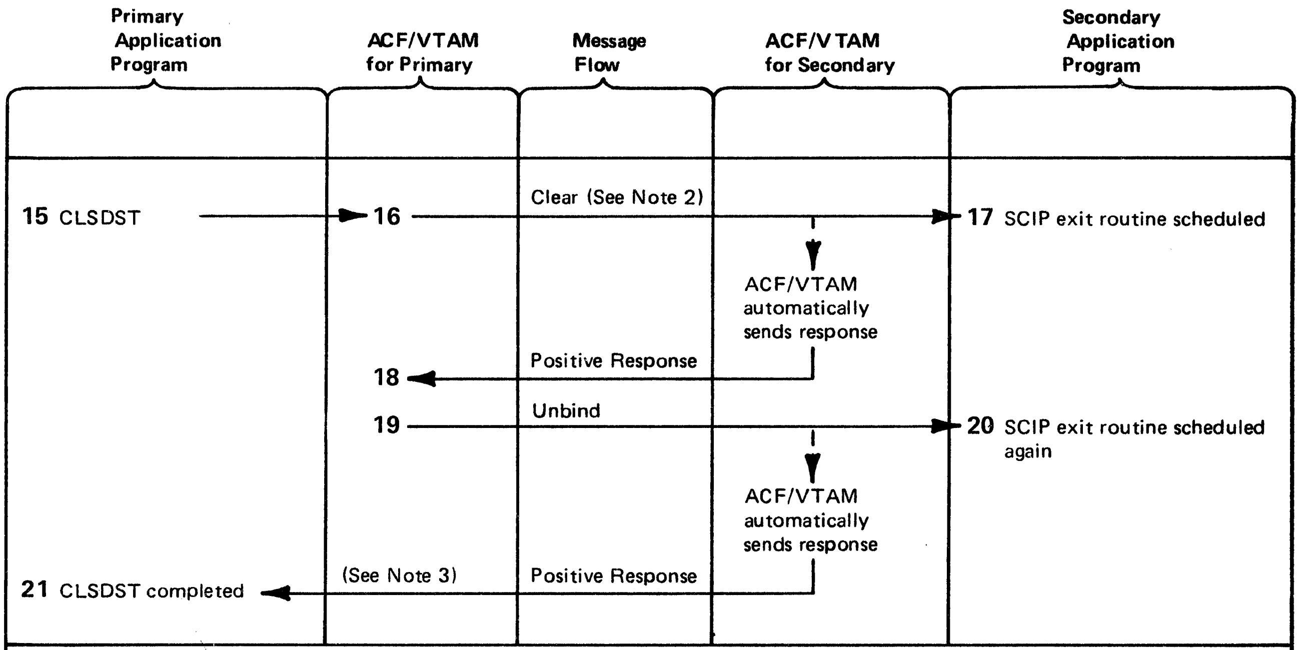

Primary Application Program
ACF/VTAM for Primary
Message Flow
ACF/VTAM for Secondary
Secondary Application Program
(Italics indicate RPL fields after receipt of message)
(Italics indicate RPL fields after receipt of message)
1 SEND STYPE=REQ CONTROL=SHUTD
Shutdown
2 RECEIVE with RTYPE=DFASY or DFASY exit routine scheduled CONTROL field contains SHUTD.
(See Note 1)
3 SEND completed
(Steps 4-7 are optional)
Chase
4 SEND STYPE=REQ CONTROL=CHASE (POST=RESP assumed)
5 RECEIVE with RTYPE=DFSYN CONTROL field contains CHASE.
Positive Response
6 SEND STYPE=RESP RESPOND=(NEX,FME)
7 SEND completed
8 SEND STYPE=REQ CONTROL=SHUTC (POST=RESP assumed)
9 RECEIVE RTYPE=DFASY or DFASY exit routine scheduled CONTROL field contains SHUTC.
(See Note 1)
Positive Response
10 SEND completed
(Steps 11-14 are optional)
Chase
11 SEND STYPE=REQ CONTROL=CHASE (POST=RESP assumed)
12 RECEIVE RTYPE=DFSYN CONTROL field contains CHASE.
Positive Response
13 SEND STYPE=RESP RESPOND=(NEX,FME)
14 SEND completed

Figure C-24 (Part 1 of 2). The Primary Application Program Shuts Down the Secondary Application Program


Primary Application Program
ACF/VTAM for Primary
Message Flow
ACF/VTAM for Secondary
Secondary Application Program
15 CLSDST
16
Clear (See Note 2)
17 SCIP exit routine scheduled
ACF/VTAM automatically sends response
18
Positive Response
19
Unbind
20 SCIP exit routine scheduled again
ACF/VTAM automatically sends response
21 CLSDST completed
(See Note 3)
Positive Response

Notes:

1. When an application program receives an expedited-flow command (including Shutdown or Shutdown Complete), either ACF/VTAM or the application program sends the response to the command, depending on whether PROC=SYSRESP or PROC=APPLRESP was specified in the NIB at connection.
2. If permitted by the transmission services profile in the session parameters, the Clear command is sent before the Unbind command if the secondary application program is in a different domain from the primary application program.
3. At this point, any outstanding RECEIVE with OPTCD=SPEC is posted complete as "cleared".

Figure C-24 (Part 2 of 2). The Primary Application Program Shuts Down the Secondary Application Program

Appendix D. Example of a Primary Application Program

This appendix contains the assembler language instructions for an ACF/VTAM application program, SAMP1. The logic for this program is similar to the logic for Sample Program 1 discussed in Chapter 11. The instructions shown in the listing on the following pages are written for a DOS/VS system. The program includes DSECTs for the ACB, EXLST, NIB, and RPL control blocks. Three other versions of the program can be obtained by substituting instructions that are shown in the listing as comments. The other versions are:

DOS/VS with manipulative macros (instead of DSECTs)

OS/VS with manipulative macros

OS/VS with DSECTs

The program is presented primarily to provide sets of integrated examples, showing how the ACF/VTAM macro instructions are used in real coding. The program is not intended to be coded and used by an installation.

What SAMP1 Does

SAMP1 is designed to communicate with one or more logical units in a network. SAMP1 is activated by input from a logical unit. (The logical unit may have created the input message as the result of terminal operator input received by the logical unit.) Depending on the code at the beginning of each message, SAMP1 performs a simple action, such as sending the message back to the logical unit.

Although SAMP1 can communicate with a number of different logical units during its execution, it is synchronous in its operation; that is, a reply is sent to one logical unit before a message is accepted from another logical unit. The use of synchronous processing simplifies program design but makes logical units unnecessarily interdependent, particularly during the I/O associated with OPNDST and CLSDST.

A program that would be executed in a logical unit with which SAMP1 might communicate is not shown.

How SAMP1 Relates to Sample Program 1 (Chapter 11)

SAMP1 is based on the general logic of Sample Program 1 described in Chapter 11. To understand the general logic of SAMP1, you should first read Chapter 11.

The logic of SAMP1 differs slightly from the logic of Sample Program 1. The request to receive input from any logical unit is specified as asynchronous in SAMP1. This allows the TPEND ECB to be checked following each issuance of a RECEIVE macro instruction and the program to be closed if the TPEND ECB has been posted. In the logic in Sample Program 1 in Chapter 11, there is no opportunity to notice that the TPEND ECB has been posted while waiting for input to arrive and the synchronously-specified RECEIVE to complete. Although the RECEIVE in SAMP1 is specified as asynchronous, it is effectively synchronous as far as handling logical unit input and output is concerned; the multiple wait waits only for posting of the one RECEIVE ECB or the TPEND ECB. (Note that a CHECK macro is required to test for successful completion of the RECEIVE and to free the RPL for reuse.)

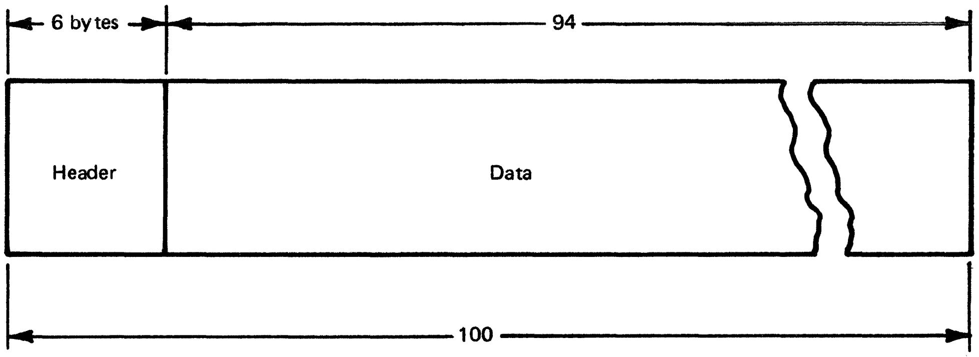
The TPEND, LERAD, SYNAD, and LOSTERM exit routines are not shown in Chapter 11 in order to simplify the example. They are included here since a complete program requires them. Although SAMP1's LOSTERM, LERAD, and SYNAD exit routines are not as complete in their ability to handle errors and special conditions as some installations may require, they provide an idea of the linkage and processing instructions that a LOSTERM, LERAD, or SYNAD exit routine might contain.

Sample Program 1 in Chapter 11 demonstrates general logic that a simple program might contain; no data or message interface is described. Since SAMP1 is a complete program that can be assembled and executed with online logical units, it requires a defined message interface.

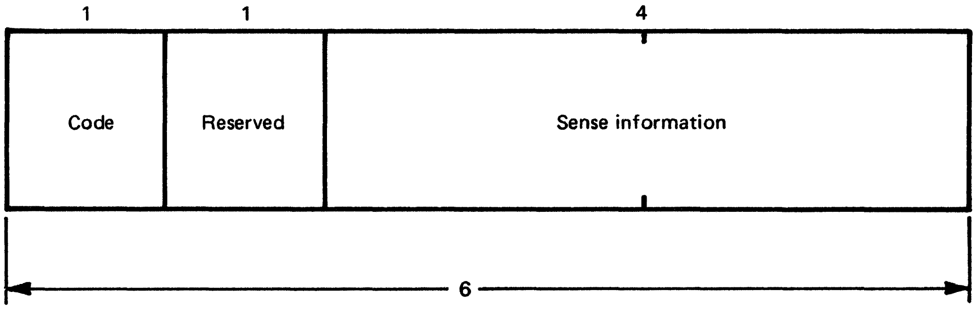
Note that this version of SAMP1 uses codes in binary rather than hexadecimal, in order to simplify the LU to LU operator interface. This is implemented in SAMP1 by means of TM and BO instructions in place of CLI and BE instructions.

The Message Interface between SAMP1 and Logical Units

Input: SAMP1 expects to receive a message in this format from any logical unit to which it has become connected:



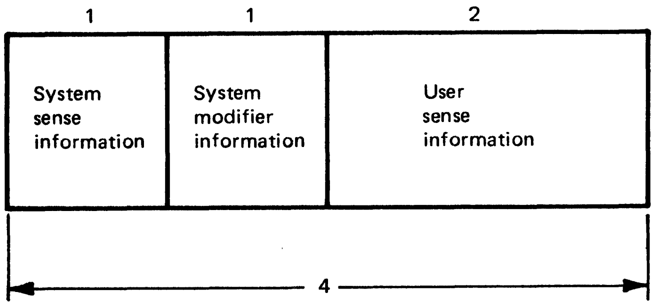
The header is in this format:



The code can contain:

B'xxxx xxxx1':	Means return a negative response as specified in the sense information message header.
B'xxxx x1xx':	Means send this message back to the logical unit.

The sense information is present on input only if the code is B'xxxx xxx1'. The sense information is in this format:



The sense information in the header allows the receiving of an exception message by SAMP1 to be simulated when the logical unit (or the terminal operator associated with the logical unit) requests it. The code will contain B'xxxx xxx1' and the sense information in the header will contain what ACF/VTAM would place in the SSENSEI, SSENSMI, and USENSEI fields of the RPL if a real exception message were to be received. To send the response to the simulated exception message, SAMP1 must move the sense information from the message header to the SSENSEO, SSENSMO, and USENSEO fields of the RPL that is used to send the response.

If the code contains B'xxxx x1x1' (that is, deliberate exception and echo), the logical unit not only expects a negative response to be returned, but wants a message sent that includes the sense information in the header of the message that just arrived (the simulated exception message). This will cause the message sent now by SAMP1 to be interpreted by the logical unit as an exception message, in turn causing the logical unit to return a negative response back to SAMP1, driving the RESP exit routine (which will call the SYNAD exit routine).

In addition to the header information, up to 94 bytes of data can be received in the input message (excess data is truncated and ignored).

When SAMP1 receives a response to an output message it has sent, ACF/VTAM schedules its RESP exit routine. Whether the response is positive or negative, no input area is required; the information is present in fields of the read-only RPL.

Output: SAMP1 sends a message using the same area and format that is used for input. On output, it sets the code to:

B'1xxx x011':	Means this is the exception message that you requested.
B'1000 0100':	Means forward this message to the terminal operator and then send it back to me. (This is used only from the LOGON exit routine and ensures that the main program RECEIVE will be driven whether or not there is a terminal operator.)

No output area is required when sending either a positive or a negative response.

Notes on SAMP1

The following notes supplement the general logical description of Sample Program 1 in Chapter 11 and the prologue and comments provided with the source listing of SAMP1, shown on the following pages.

Notes on the Main Program

The message input area, AREA1, is initialized to asterisks to aid in debugging. The message length as well as the content will be evident if asterisks (rather than blanks) are printed when the message is presented to the terminal operator who is driving the programs. (Other printable nonalphabetic characters could have been used as well.)

After the RECEIVE is completed, a CHECK is issued. If an error or special condition has occurred, the LERAD or SYNAD exit routine is entered. If the exit routine is able to recover successfully, it sets register 15 to 0; the main program is unaware that the exit routine was entered. If the exit routine is not able to recover successfully, it disconnects the logical unit or performs a SESSIONC CLEAR and SDT, and sets register 15 to nonzero. The main program continues with other input, looping back to reissue the RECEIVE.

The RECEIVE can be completed by receipt of an expedited-flow (DFASY) or normal-flow (DFSYN) message. For example, receipt of a Request Shutdown command (RSHUTD) causes SAMP1 to issue a CLSDST for that logical unit before reissuing the RECEIVE. Previous versions of SAMP1 provided no method of accepting expedited-flow messages, which meant that a logical unit would never get the response it was expecting.

A check is made to see if a simulated exception message has arrived; if so, a deliberate negative response must be returned. If a real exception message were to be received (requiring a negative response), the SYNAD exit routine would be scheduled as a result of the CHECK. In this case, the SYNAD exit routine would send a negative response, use the SESSIONC macro to clear data traffic, reset the logical unit to CA mode, and set an unsuccessful recovery indication in register 15 so that the main program will reissue its RECEIVE.

Responses: SAMP1 is intended to handle all the valid combinations of no responses, exception response only, and definite responses. When instructed to echo the data (B'xxxx x1xx' in the first byte of the data header), it leaves the response types (RESPOND=*values*) unchanged, making it possible to force SAMP1 to send a message to the logical unit asking for no response or exception response only. In such a case, SAMP1 makes sure that the logical unit's session is in continue-any mode on completion of the SEND rather than on receipt of a response by the RESP exit routine. When SAMP1 asks for a definite response, it leaves the logical unit's session in continue-specific mode until that response is processed by the RESP exit routine. The next message (or messages) may already be queued in ACF/VTAM's pageable buffers but will be ignored until the response has been handled.

Function Management Protocols: SAMP1 does not modify the settings related to FM protocol in the RPLs, except when requested to send an exception response. This means that fields like FMHDR, CHAIN, BRACKET, CODESEL, and CHNGDIR are echoed back to the logical unit, ignoring the fact that this may be a protocol violation. In addition, SAMP1 processes (for example, by echoing) each chain element separately.

Ending the Program: There are two ways to end the program. Either a message can be sent from the logical unit that says, "CLOSE ACB", or the ACF/VTAM network operator can halt the network, causing the TPEND exit routine to be driven. In the first case, a TPEND flag is set; in the second, a TPEND ECB is posted. The main program checks both of these during each of its loops, branching to close the ACB if a close indication is found. Prior to closing the ACB, the TPEND flag is set to hex FF to prevent any undesired activity while the CLOSE macro instruction is being executed by ACF/VTAM. The TPEND flag is checked by the LOGON, RESP, and LOSTERM exit routines when each is entered to make sure the exit routine has not been scheduled while closing the ACB is in progress. If the flag is set, the exit routine returns immediately to ACF/VTAM.

Notes on the LOGON Exit Routine

Notice in the manipulative macro version how the symbolic name of the logical unit for which a logon request has been received is placed in the NIB prior to connecting it. NAME=(*,0(4)) is specified in the MODCB macro.

The last four bytes of the 8-byte symbolic name are placed in the USERFLD of the NIB (USERFLD=(*,4(4)) specified in the MODCB macro) for aid in debugging. These bytes will identify the specific logical unit more easily than will the CID (located in the RPL ARG field). This part of the symbolic name will be available in the USER field of the RPL except in the case of the RPL used for OPNDST.

An arbitrary validation of the logical unit is used: a check for a symbolic name that begins with the characters "CT" (the name of each LU statement would be specified CTxxxxxx in the ACF/VTAM definition process).

The IBM-supplied macro, ISTDNIB, is used to generate a dummy control section for the NIB. This enables SAMP1 to check the device characteristics field for a logical unit device-type indication. The CLI DEVTCODE,DEVLU statement is possible only if the NIB DSECT has been used, generating the labels DEVTCODE and DEVLU in the assembled output. Note that a CSECT statement must follow the ISTDNIB statement in the constants area so that the SAMP1 CSECT can be resumed. Note also that none of the field names in SAMP1 begins with any of the reserved combinations (NIB, RPL, ACB, etc.).

A conditional completion code of nonzero following the INQUIRE at label INQUIRE, indicating no logon data, causes the logical unit to be disconnected. Since the logon data has already been determined to have a length of nonzero, a conditional completion code of nonzero here would be contradictory.

After the logical unit has been connected with an OPNDST, a message is sent to the logical unit confirming that the logical unit has been connected. (Note that OPTCD=SYN is used on the OPNDST. This, coupled with the fact that the macro is issued in an exit routine, means that all processing waits for the responses involved.) The message sent to the logical unit is initialized to contain hex 84 in the code byte of the header, meaning "Forward this message to the terminal operator and then send it back to the ACF/VTAM application program." Note that this version of SAMP1 includes the symbolic name of the logical unit in this message. This may help in cases where symbolic names are assigned dynamically.

Notes on the RESP Exit Routine

Since the read-only RPL whose address is provided on entry to the RESP exit routine cannot be used to reset the logical unit to CA mode, an RPL (PRPLR) is reserved in the RESP exit routine for this purpose. The address of the ACB is obtained from the parameters passed on entry. Note that, since only one exit routine can be executing at a time and all RPL-based requests in SAMP1's exit routines are synchronous, one RPL could have been shared among all the exit routines.

In the event a negative response is received, the RESP exit routine sets up the correct linkage and calls the SYNAD exit routine directly. If the SYNAD exit routine is able to recover successfully (SAMP1 does not attempt to resend any messages), it sets register 15 to 0 and the RESP exit routine restores registers 1-12 and resets the logical unit to CA mode. If it is not able to recover successfully, any necessary action, such as disconnecting the logical unit, is taken in the SYNAD exit routine. The RESP exit routine need only return to ACF/VTAM.

Notes on the LERAD and SYNAD Exit Routines

SAMP1's LERAD and SYNAD exit routines are reenterable because the LERAD and SYNAD exit routines can be entered as the result of RPL-based requests issued by both the main program and exit routines other than LERAD and SYNAD. For example, the RECEIVE in the main program could cause the SYNAD exit routine to be entered. While the SYNAD is being executed as an extension of the main program, it could be interrupted and the LOGON exit routine given control. The OPNDST in the LOGON exit routine could cause the SYNAD exit routine to be reentered, thus destroying any storage that might have values for the SYNAD exit routine as an extension of the main program. For this reason, SAMP1's LERAD and SYNAD exit routines obtain unique storage areas each time they are entered.

If the SYNAD exit routine is scheduled as the result of an RPL-based request that is issued within the SYNAD exit routine (that is, if the SYNAD exit routine is entered recursively), SAMP1 terminates with a dump. To determine recursion, SAMP1 uses the leftmost bit of register 1, which also contains the RPL address on entry to the SYNAD exit routine. Before issuing any RPL-based macro in the exit routine, this bit is set on; on entry, if this bit is found to have been set, the program terminates.

In the manipulative macro version, a series of TESTCB and MODCB macros are used to determine the SSENSEI setting so that it can be set in the SSENSEO field before sending a negative response. This is required because the value for SSENSEO cannot be specified using register notation or in the FIELDS parameter of SHOWCB because it is a bit-encoded field.

The LERAD exit routine illustrates three cases that cause entry to LERAD but are not logical errors in the context of SAMP1, which is programmed to ignore them. These are cases that can arise due to the use of asynchronous exits.

Notes on the LOSTERM Exit Routine

The LOSTERM exit routine determines why the logical unit was lost and takes appropriate action.

Upon entry, the exit-routine tests the TPEND flag to see if the system is being shut down. If the flag is set, the routine returns to ACF/VTAM to allow the closing operations to be completed. Otherwise, the routine inspects the code that was passed to the exit routine in the fourth word of the parameter list pointed to by register 1 when LOSTERM was entered.

The exit-routine then takes action on the basis of the decimal reason code. For code 24 (ACF/VTAM is attempting to restart the unit), the exit-routine merely returns to ACF/VTAM, knowing that control will be returned to the exit-routine at the end of the restart attempt. (Note that, in the context of SAMP1, which uses only logical-unit-initiated logons, it would be more natural to CLSDST immediately but the code used is intended to be illustrative.) When the exit routine is again entered, the fourth word of the parameter list will contain either code 16 (logical unit was successfully restarted) or code 12 (restart was unsuccessful and logical unit was lost). At this point, a CLSDST is required, but a program might use INQUIRE OPTCD=CIDXLATE to find out the symbolic name of the logical unit in order to send a suitable message to the network operator (for example, a message asking for a VARY LOGON). If the program knew that it normally acquires the lost logical unit, it could use OPTCD=PASS on the CLSDST (assuming it was properly authorized to do that in the APPL statement).

The other possible reasons for entering LOSTERM are equally application and environment dependent and CLSDST may not always be the most suitable action to take. For more information on the reason codes passed to the LOSTERM exit routine, see the general description of the LOSTERM exit routine in Chapter 7.

The Source Statements for SAMP1

```
***********************************************************************
***********************************************************************
*
*  SAMP1 (SAMPLE PROGRAM 1) IS DESIGNED TO BE
*  RELATIVELY EASY TO UNDERSTAND.    IT ILLUSTRATES:
*
*     O  OPENING AND CLOSING A PROGRAM, INCLUDING A TPEND
*        EXIT-ROUTINE.
*
*     O  CONNECTING LOGICAL UNITS IN A LOGON EXIT-ROUTINE.
*
*     O  RECEIVING AND SENDING MESSAGES AS SYNCHRONOUS
*        OPERATIONS.  NOTE THAT THE RECEIVE USES OPTCD=ASY
*        AND AN ECB FOLLOWED BY A MULTIPLE WAIT, IN ORDER TO ALLOW
*        WHAT TPEND DOES (E.G. SET A CLOSEDOWN SWITCH FOR MAINLINE)
*        TO TAKE EFFECT EVEN DURING A LULL IN I/O ACTIVITY.
*
*     O  RESETTING A LOGICAL UNIT TO CONTINUE-ANY MODE.
*     O  RECEIVING AND SENDING RESPONSES TO MESSAGES.
*
*     O  RESP, TEPDN, LOSTERM, LERAD, AND SYNAD EXIT-ROUTINES.
*
*     O  THE LINKAGE BETWEEN THE MAIN PROGRAM, VTAM, AND
*        EXIT-ROUTINES.
*
*
*  ALTHOUGH DOS/VS CODE IS SHOWN, IT
*  DIFFERS ONLY IN MINOR RESPECTS FROM OS/VS CODE.
*
*  SAMP1 (SAMPLE PROGRAM 1) IS ORGANIZED INTO:
*
*     O  MAIN PROGRAM.
*     O  LOGON EXIT-ROUTINE.
*     O  RESP EXIT-ROUTINE.
*     O  LERAD EXIT-ROUTINE.
*     O  SYNAD EXIT-ROUTINE.
*     O  TPEND EXIT-ROUTINE.
*     O  LOSTERM EXIT-ROUTINE.
*
***********************************************************************
```

```
**********************************************************************
*
*   NAME = MAIN PROGRAM
*
*   FUNCTION = OPENS THE ACB, RECEIVES INPUT FROM ANY CONNECTED LOGICAL
*       UNIT, SENDS A RESPONSE IF REQUESTED, FORWARDS INPUT TO A PRO-
*       CESSOR, SENDS A REPLY PREPARED BY THE PROCESSOR, AND LOOPS BACK
*       TO RECEIVE MORE INPUT AFTER SENDING THE REPLY.  CLOSES THE
*       ACB (CLOSES THE PROGRAM) IF A TPEND ECB IS POSTED BY VTAM HALT
*       OR IF 'CLOSE ACB' IS ENTERED AS A MESSAGE.
*       NOTE:  THE PROGRAM HANDLES ONE ELEMENT OF AN INPUT CHAIN AT A    |
*
*       TIME.  BE CAREFUL WITH HDX-FF PROTOCOLS.
*   ENTRY POINT=SAMP1
*
*   INPUT = MESSAGES RECEIVED FROM CONNECTED LOGICAL UNITS; A POSTED
*       TPEND ECB.  EACH MESSAGE CONTAINS A SIX-BYTE HEADER DESCRIBING
*       THE ACTION TO BE TAKEN (SEE EQUATES IN MAIN PROGRAM CONSTANTS).
*       NOTE THAT DFASY INPUT CAUSES A CLSDST.  BRACKET PROTOCOL IS NOT  |
*       SUPPORTED AND CAUSES UNPREDICTABLE RESULTS.                      |
*
*   OUTPUT = MESSAGES AND RESPONSES SENT TO LOGICAL UNITS AS A RESULT
*       OF INPUT MESSAGES.  PROGRAM TERMINATION AND A DUMP IF THE PRO-
*       GRAM CANNOT CONTINUE.
*
*   EXTERNAL REFERENCES = OPEN, DUMP, SETLOGON, RECEIVE, CLSDST, TESTCB,
*       WAITM, CHECK, MODCB, AND SEND.
*
*   EXIT, NORMAL = BR 14
*
*   EXIT, ABNORMAL = DUMP (DIRECTLY OR VIA SYNAD OR LERAD).
*   ATTRIBUTES = NOT SERIALLY REUSABLE
*
*   REGS USED
*
*     3 = BASE
*     4 = WORK REG
*     5 = A(PRPL)
*    13 = A(SAVEO)
*
**********************************************************************
SAMP1     CSECT
*OS       STM   R14,R12,12(R13)         SAVE REGISTERS FOR CALLER
          BALR  R3,R0                   ESTABLISH BASE
          USING *,R3                    ESTABLISH ADDRESSABILITY
*OS       ST    R13,SAVEO+4             STORE HIGH-SAVEAREA POINTER
          LA    R15,SAVEO
*OS       ST    R15,8(R13)              STORE LOW-SAVEAREA POINTER
          LR    R13,R15                 POINT R13 TO OUR SAVEAREA
*           OPEN THE ACB
OPNACB   EQU    *                                                        |
          SR    R15,R15                 CLEAR REGISTER 15 (DOS/VS)
          OPEN  PACB                    CONNECT THE PROGRAM TO VTAM
          LTR   R15,R15                 TEST FOR ERRORS
          BZ    OPENOK
```

```
*  IT WOULD BE NORMAL HERE TO TEST FOR INVALID APPLID AND COMMUNICATE
*  WITH THE OPERATION RATHER THAN TO DUMP.
DUMP       ST    R1,R1CONTS           SAVE THE CONTENTS OF REG 1
           DUMP                        IF UNSUCCESSFUL REQUEST
VERSION    DC    C'DATE OF LAST CHANGE MAY 3 76.'                        |
OPENOK    EQU    *
           LA    R5,PRPL              SET UP BASE FOR RPL DSECT
           USING IFGRPL,R5
           SETLOGON RPL=PRPL,OPTCD=START  ALLOW LOGON REQUESTS
           LTR   R15,R15              TEST FOR ERRORS
           BNZ   DUMP
*            REQUEST INPUT FROM ANY LOGICAL UNIT
RECANY     MVI   AREA1,C'*'           SET ASTERISK IN 1ST BYTE OF
*                                     AREA1 (FOR DEBUGGING PURPOSES)
           MVC   AREA1+1(99),AREA1    ROLL IT
           RECEIVE RPL=PRPL,AREA=AREA1,AREALEN=100,
                 OPTCD=(ASY,ANY,CS),ECB=RCVECB,  RESP HANDLED BY EXIT
                 RTYPE=(DFSYN,DFASY)                                     |
*  OPTCD=CS IS USED TO FORCE ROTATION OF THE CONNECTIONS WHICH
*  SATISFY RECANY.  OTHERWISE A BUSY CONNECTION MIGHT LOCK OUT
*  OTHER CONNECTIONS.
           LTR   R15,R15              TEST FOR ACCEPTANCE
           BNZ   DUMP                 DUMP IF NOT ACCEPTED
           TM    RCVECB+2,X'80'       IS RECEIVE ALREADY POSTED?
*OS        TM    RCVECB,X'40'
           BO    CHECK                YES, BYPASS WAITM SVC
MWAIT      WAITM RCVECB,TPENDECB      WAIT FOR RECEIVE ANY OR TPEND
*OS        WAIT ECBLIST=ECBLST
           TM    TPENDECB+2,X'80'     IS POSTED ECB THE TPEND ECB?
*OS        TM    TPENDECB,X'40'
           BO    RETURN1              YES, GO TO CLOSE ACB
CHECK     EQU    *                                                       |
          OI    RESETCAF,X'FF' INIT RESETSR CA NEEDED FLAG               |
           CHECK RPL=PRPL             NO, CHECK COMPLETION OF RECEIVE
           LTR   R15,R15              TEST FOR SUCCESSFUL COMPLETION
           BNZ   RECANY              NO, CONTINUE WITH NEXT INPUT
*  SYNAD WILL HAVE EITHER CLSDST THE FAILING CONNECTION OR HAVE CLEARED
*  THE EXCEPTION MESSAGE AND RESTORED THE CONNECTION TO CA MODE.
           TM    RPLSRTYP,RPLDFASY    DFASY RECEIVED?                    |
           BNO   TESTRRN              NO                                 |
**         TESTCB AM=VTAM,RPL=PRPL,RTYPE=DFASY,ERET=DUMP
**         BNE   TESTRRN
           CLSDST RPL=PRPL,OPTCD=SYN                                     |
*          IGNORE POSSIBLE FAILURE OF CLSDST.  SYNAD/LERAD COPE          |
           B     CHCKTPND             ALLOW FOR PROGRAM CLOSEDOWN        |
TESTRRN    EQU   *                                                       |
           TM    RPLVTFL2,RPLRRN     RRN RESPONSE WANTED                 |
           BO    TESTEXCP                                                |
**         TESTCB AM=VTAM,RPL=PRPL,RESPOND=RRN,ERET=DUMP
**         BE    TESTEXCP
           TM    RPLVTFL2,RPLNFME     TEST FOR NO RESPONSE               +
           BO    PROCESS                                                 +
**         TESTCB AM=VTAM,RPL=PRPL,ERET=DUMP,  TEST FOR NO RESPONSE      *
**              RESPOND=NFME          REQUESTED
**         BE    PROCESS              BYPASS RESPONSE SENDING
```

```
TESTEXCP  EQU  *                                                   *P
          TM   AREACODE,AEXCEPT     DELIBERATE EXCEPTION RESP WANTED?|
*  THE PRECEDING TEST CHANGED TO TM FROM CLI TO ALLOW EBCDIC HEADERS.
          BNO  RESPTEST             NO, CHECK FOR DEFINITE RESPONSE  |
          MVC  RPLSSEO,AREASENS     SET SYS SENSE OUTPUT             +
          MVC  RPLSSMO,AREASENS+1                                    +
**        IC   R7,AREASENS+1        PICK UP SSENSMO
          MVC  RPLUSNSO,AREASENS+2                                   +
          OI   RPLVTFL2,RPLEX                                        +
         NI    RPLOPT5,X'FF'-RPLDLGIN  SET OPTCD=CA FOR SENDD        |
**        LH   R8,AREASENS+2        PICK UP USENSEO
**        MODCB AM=VTAM,RPL=PRPL,RESPOND=EX,  SET EX                 *
**             SSENSMO=(R7),USENSEO=(R8),OPTCD=CA
**        LTR  R15,R15
**        BNZ  DUMP
          B    SENDRESP             GO TO SEND THE EXCEPTION RESPONSE
RESPTEST  TM   RPLVTFL2,RPLEX                                        +
         BNO   SENDRESP           DEFINITE RESP SO LEAVE OPTCD=CS    |
         NI    RPLOPT5,X'FF'-RPLDLGIN  SET OPTCD=CA FOR SENDD        |
**        MODCB AM=VTAM,RPL=PRPL,OPTCD=CA
**        LTR  R15,R15
**        BNZ  DUMP
          B    PROCESS                                               |
**SPTEST  TESTCB AM=VTAM,RPL=PRPL,RESPOND=NEX,     DEFINITE RESPONSE *
**             ERET=DUMP                            REQUESTED?
**        BNE   PROCESS              NO
SENDRESP EQU   *
          SEND RPL=PRPL,STYPE=RESP,OPTCD=(SYN,SPEC) SEND PREPARED RES|
          LTR  R15,R15              TEST FOR SUCCESSFUL COMPLETION
          BNZ  DUMP                 DUMP IF SEND COULD NOT BE
*                                   SCHEDULED
         NI    RESETCAF,X'00' TURN OFF RESETSR CA NEEDED FLAG
*** THE PROCESS ROUTINE HAS BEEN MODIFIED TO SUPPORT INPUT CHAINS.   |
PROCESS   EQU  *
*
*            PROCESS INPUT AND PREPARE REPLY
*
TEST1    EQU   *
CLOSETST  CLC  AREADATA(9),=C'CLOSE ACB'  IS CLOSE ACB REQUESTED
         BNE   TEST2
          OI   TPENDFLG,X'80'       SET ON TPEND FLAG TO CLOSE ACB
TEST2    EQU   *
         OI    AREACODE,ASAD
         TM    AREACODE,AECHOB    IS AN ECHO WANTED TO THE TERMINAL
         BNO   TEST3
         OI    AREACODE,AECHO     YES SET ON HERE IS YOUR ECHO FLAG
         NI    AREACODE,X'FF'-AECHOB    TURN OFF PLEASE ECHO BACK FLG
*           SEND THE REPLY (SCHEDULE ITS SENDING)
SENDDATA EQU   *                                                   *P
SENDD     SEND RPL=PRPL,STYPE=REQ,                   NOTE THAT THIS*P*
               OPTCD=SYN,POST=SCHED   LEAVES CA,CS AS SET ABOVE.
          LTR  R15,R15              TEST FOR SUCCESSFUL COMPLETION
          BNZ   DUMP                DUMP IF SEND COULD NOT BE SCHED
         NI    RESETCAF,X'00'     TURN OFF RESETSR CA FLAG
```

```
TEST3    EQU   *
*** THIS ROUTINE ALLOWS THE NEXT INPUT CHAIN ELEMENT FROM THIS LU TO  |
***  SATISFY THE RECEIVE ANY.  IT IS NEEDED WHEN NO OR ONLY EXCEPTION |
***  RESPONSES ARE EXPECTED.                                         |
         CLI   RESETCAF,X'FF'       IS A RESETSR CA NEEDED
         BNE   TEST4
         RESETSR RPL=PRPL,OPTCD=(CA,SYN)
         LTR   R15,R15
         BNZ   DUMP
TEST4    EQU   *
*         CHECK TPEND FLAG FOR CLOSEDOWN
CHCKTPND  CLI   TPENDFLG,X'80'      SEE IF TPEND IS SIGNALLED
          BNE   RECANY               IF NOT, BRANCH BACK TO RECEIVE
*         CLOSE THE PROGRAM, DISCONNECTING ALL LOGICAL UNITS
RETURN1   MVI   TPENDFLG,X'FF'       SIGNAL CLOSE IN PROGRESS TO EXITS
          CLOSE PACB                 CLOSE THE ACB
          EOJ
*OS       L     R13,SAVE0+4        PICK UP HIGH-SAVEAREA ADDRESS
*OS       LM    R14,R12,12(R13)    RESTORE CALLER'S REGISTERS
*OS       BR    R14                RETURN TO CALLER (OS)
***
*  MAIN PROGRAM CONSTANTS
***
PACB      ACB   AM=VTAM,APPLID=APPL1,EXLST=EXLST1,MACRF=LOGON THE ACB
EXLST1    EXLST AM=VTAM,LOGON=LOGON1,SYNAD=SYNAD1,LERAD=LERAD1,       *
               RESP=RESP1,TPEND=TPEND1,LOSTERM=LOSTERM1
PRPL      RPL   AM=VTAM,ACB=PACB
R1CONTS   DC    F'0'                SAVE AREA FOR REG 1 IN DUMP
ECBLST    DC    A(RCVECB)                                         *OS
          DC    X'80'               OS END OF ECB LIST MARKER     *OS
          DC    AL3(TPENDECB)                                     *OS
RCVECB    DC    F'0'                ECB USED FOR RECANY
TPENDECB  DC    F'0'                ECB POSTED BY TPEND EXIT
TPENDFLG  DC    X'00'               FLAG SET BY MAINLINE TO FORCE CLOSE
RESETCAF DC    X'00'       RESETSR CA NEEDED IF 00
SAVE0     DC    18F'0'              SAVE AREA NEEDED FOR MAIN PROGRAM
APPL1     DC    X'08'               APPLIED FOR ACB
          DC    CL8'PROG1'
          DS    0H
AREA1     DS    0CL100              I/O DATA AREA
AREAHEAD  DS    0CL6                HEADER
AREACODE  DS    XL1
*                    EQUATES FOR INPUT
AEXCEPT   EQU   X'01'               PLEASE RETURN AN EXCEPTION RESPONSE
*                                   AS SPECIFIED IN AREASENS
AECHOB    EQU   X'04'               PLEASE ECHO THIS BACK TO ME
*                    EQUATES FOR OUTPUT
ASAD      EQU   X'80'               SEND THIS MESSAGE TO THE SCREEN
AECHO     EQU   X'02'               THIS IS THE ECHO YOU REQUESTED
          DS    XL1                 RESERVED
AREASENS  DS    XL4                 SENSE FIELD WHEN AREACODE='08'|'01'
AREADATA  DS    CL94                DATA FIELD
AREAOFLO  DC   C'*THIS SHOULD NOT BE DISPLAYED:  CHECK RECLEN'
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
           LTORG
           EJECT
           IFGRPL AM=VTAM
           EJECT
           ISTUSFBC
           EJECT
           IFGACB AM=VTAM
           EJECT
           IFGEXLST AM=VTAM
SAMP1      CSECT                 RESTART CSECT
************************************************************************
```

```
***********************************************************************
*
*   NAME = LOGON EXIT ROUTINE
*
*   FUNCTION = CONNECT AND SEND A GOOD MORNING MESSAGE TO ANYTHING
*   THAT LOGS ON IF IT IS A LOGICAL UNIT, ITS SYMBOLIC
*   NAME BEGINS WITH 'CT' AND ANY LOGON DATA STARTS WITH 'XYZ';
*   OTHERWISE, REJECT IT.
*
*   ENTRY POINT = LOGON1
*
*   INPUT
*      REGISTERS
*            0      =  UNPREDICTABLE
*            1      =  POINTER TO 4-WORD PARAMETER LIST
*            2-13   =  UNPREDICTABLE
*            14     =  ADDRESS TO RETURN CONTROL TO
*            15     =  ENTRY ADDRESS OF THIS ROUTINE
*      PARAMETER LIST - 4 WORDS
*            1      =  ACB ADDRESS
*            2      =  POINTER TO SYMBOLIC NAME OF NODE
*            3      =  UNPREDICTABLE
*            4      =  LENGTH OF LOGON MESSAGE
*
*   OUTPUT
*      A REQUEST TO VTAM TO CONNECT OR REJECT THE LOGICAL UNIT
*      OR PROGRAM TERMINATION AND A DUMP IF UNABLE TO CONTINUE.
*      IF SUCCESSFUL CONNECTION A GOOD MORNING MESSAGE IS SENT
*      TO THE LOGICAL UNIT SPECIFYING EXCEPTION RESPONSE ONLY.
*
*   EXTERNAL REFERENCES = MODCB, INQUIRE, OPNDST, CLSDST, SEND,
*      AND DUMP.
*
*   EXIT, NORMAL = BR 14
*
*   EXIT, ABNORMAL = DUMP
*
*   ATTRIBUTES = SERIALLY REUSABLE
*
*   REGS USED
*
*     3 = BASE
*     4 = A(SYMBOLIC NAME OF LU)
*     5 = A(PRPLCONN),IFGRPL
*     6 = A(LOGON EXIT PARM LIST)
*     7 = A(PNIB),ISTDNIB
*     8 = LENGTH OF LOGON MESSAGE
*     9 = ACB ADDRESS
*    13 = A(SAVE2)
*
***********************************************************************
LOGON1   BALR  R3,R0
         USING *,R3 ESTABLISH ADDRESSABILITY
         LA    R13,SAVE2            PROVIDE SAVE AREA FOR MACROS
```

```
          ST    R14,SAVE1               SAVE RETURN ADDRESS TO VTAM
          L     R6,=A(TPENDFLG)         POINT TO TPENDFLG
          TM    0(R6),X'FF'             IS CLOSE ACB IN PROGRESS
          BO    RETURN2                 YES, ALLOW CLOSE TO REJECT LOGONS
          LR    R6,R1                   SAVE THE PARAMETER LIST ADDRESS
          L     R9,0(R6)                PICK UP ACB ADDRESS
          L     R4,4(R6)                POINT TO THE SYMBOLIC NAME OF
*                                       THE LOGICAL UNIT
          LA    R5,PRPLCONN             SET UP BASE FOR RPL DSECT        +
          USING IFGRPL,R5                                                +
          USING ISTDNIB,R7
          LA    R7,PNIB                 LOAD BASE FOR NIB DSECT
          MVC   NIBSYM,0(R4)                                             +
          MVC   NIBUSER,4(R4)                                            +
          MVC   RPLUSFLD,4(R4)          PUT USER FIELD IN OPNDST RPL ... |
*                                       VTAM DOES NOT SET IT ON OPNDST   |
*   THE ABOVE MVC IS USEFUL IF OPTCD=ASY USED.  NOTE THAT IT CANNOT
*     BE DONE WITH MODCB.
          MVC   FIRSTMID(8),0(R4)  PUT ID IN GOOD MORNING MESSAGE
          B     VALIDATE                                                 +
**        MODCB AM=VTAM,NIB=PNIB,NAME=(*,0(R4)),  PUT IN NIB             *
**              USERFLD=(*,4(R4))       MAY HELP DEBUGGING (UNIQUE PER LU)
**        LTR   R15,R15                 MODCB OK?
**        BZ    VALIDATE                YES, GO TO VALIDATE LOGON MESSAGE
CANCEL2   ST    R1,R1CONTS2             SAVE THE CONTENTS OF REGISTER 1
          DUMP
*           VALIDATE THE LOGON MESSAGE
VALIDATE  EQU   *
          INQUIRE  RPL=PRPLCONN,OPTCD=DEVCHAR,    PUT INQUIRE            *
                AREA=DEVCHAR,AREALEN=8,             OUTPUT               *
                ACB=(R9),NIB=PNIB                 INTO PNIB
          LTR   R15,R15
          BNZ   DISCONN
          CLI   DEVTCODE,DEVLU          IS THIS A LOGICAL UNIT?
          BNE   DISCONN                 IF NOT, REJECT LOGON
          L     R8,12(R6)               PUT LENGTH OF LOGON MSG IN 8
          LTR   R8,R8                   IS LOGONMSG LENGTH ZERO?
          BZ    CONNECT                 YES -- CONNECT
*       CLEAR LOGON MESSAGE AREA
          MVI   MSGAREA,C'*'
          MVC   MSGAREA+1(79),MSGAREA
INQUIRE   INQUIRE  RPL=PRPLCONN,OPTCD=LOGONMSG,NIB=PNIB,  OBTAIN         *
                AREA=MSGAREA,AREALEN=L'MSGAREA,           LOGON          *
                ACB=(R9)                                  MESSAGE
          LTR   R15,R15
          BNZ   CANCEL2
          LTR   R0,R0                   IS CONDITIONAL COMPLETION CODE 0?
          BZ    COMPARE                 YES, CHECK MESSAGE
          B     DISCONN                 NO, SHOULD NOT OCCUR
COMPARE   CLC   MSGAREA(3),=C'XYZ'      CHECK PASSWORD IN USER LOGON DATA
          BNE   DISCONN                 IF NOT, CANNOT GRANT REQUEST
*          CONNECT THE LOGICAL UNIT
CONNECT   OPNDST RPL=PRPLCONN,OPTCD=(SYN,ACCEPT,CA)
          LTR   R15,R15                 CONNECTED SUCCESSFULLY
          BZ    SNDFIRST                                                 |
*** IF THE OPNDST FAILS DO NOT ATTEMPT CLSDST AS THAT WILL DRIVE LERAD|
          B     RETURN2
SNDFIRST   EQU   *
          SEND  RPL=PRPLCONN,AREA=FIRSTMSH, SEND FIRST MESSAGE.
               RECLEN=L'FIRSTMSG+6+L'FIRSTMID, OPTCD=CA STILL SET .   |
               RESPOND=(EX,FME)             INPUT MAY SATISFY RECANY.
          LTR   R15,R15
          BNZ   RETURN2               ABANDON THIS CONNECTION ......
*                                     SYNAD WILL HAVE CLSDST FOR US
RETURN2   L     R14,SAVE1               RESTORE REG 14
          BR    R14                     RETURN TO VTAM
*          DISCONNECT THE LOGICAL UNIT
*  IT MIGHT BE BETTER TO SEND A REJECTION MESSAGE TO THE VTAM OPERATOR
*  BEFORE CLOSING.
DISCONN  EQU   *
          CLSDST RPL=PRPLCONN,OPTCD=SYN
*    IF CONTROL RETURNS HERE THERE IS NO NEED TO TEST FOR SUCCESS OR
*    FAILURE SINCE LERAD OR SYNAD COPE WITH FAILURE.
          B     RETURN2                 IF SO, BRANCH TO RETURN
***
*  LOGON EXIT-ROUTINE CONSTANTS
***
SAVE1     DS    F                     SAVEAREA FOR REG14 RETURN ADDRESS
SAVE2     DS    18F                   SAVEAREA FOR MACROS IN VTAM EXITS
R1CONTS2  DC    F'0'                  SAVEAREA FOR REG 1 FOR DUMP
SAVESENS DC    F'0'                   SENSE FROM FAILED OPNDST
PNIB      NIB   MODE=RECORD,               ALLOW USE OF RESP EXIT FOR LU
               PROC=(RESPX,TRUNC)          AND TRUNCATE EXCESS INPUT DATA
PRPLCONN  RPL   AM=VTAM
MSGAREA   DC    CL80' '                AREA FOR LOGON MESSAGE
FIRSTMSH  DC    XL6'840000000000'      HEADER CODE FOR DISPLAY ON TERMINAL
*                                       AND ECHO BACK TO PROG1.
FIRSTMID  DC   CL9'********-'                                          |
FIRSTMSG  DC    C'LOGON ACCEPTED.  VTAM PROG READY FOR FIRST INPUT'    |
          LTORG
          EJECT
          ISTDNIB                      ,INVOKE NIB, DEVCH, AND PROC DSECT
SAMP1     CSECT                       CONTINUE SAMP1 CSECT
```

```
*********************************************************************
*
*  NAME = RESP EXIT ROUTINE
*
*  FUNCTION = RECEIVE A RESPONSE TO THE MESSAGE SENT IN THE MAIN
*      PROGRAM.  IF THE RESPONSE IS NORMAL (POSITIVE), RESET THE
*      LOGICAL UNIT TO CONTINUE-ANY MODE SO THAT THE MAIN PROGRAM
*      RECEIVE WITH OPTCD=ANY SPECIFIED WILL ACCEPT INPUT FROM IT.
*      IF THE RESPONSE IS AN EXCEPTION, CALL SYNAD1 TO ANALYZE THE
*      EXCEPTION AND TAKE WHATEVER ACTION IS POSSIBLE.  SYNAD1S' ACTION
*      WILL BE EITHER TO CLSDST THE FAILING CONNECTION OR TO PERFORM A
*      SESSIONC CLEAR AND SDT.  IN BOTH CASES CONTROL IS RETURNED TO
*      THIS EXIT AT LABEL SYNRTURN.
*
*  ENTRY POINT = RESP1
*
*  INPUT
*     REGISTERS
*          0      =  UNPREDICTABLE
*          1      =  ADDRESS OF A 5-WORD PARAMETER LIST
*          2-13   =  UNPREDICTABLE
*          14     =  ADDRESS TO RETURN CONTROL TO
*          15     =  ENTRY ADDRESS TO THIS ROUTINE
*       PARAMETER LIST - 5 WORDS
           1      =  ADDRESS OF THE ACB
*          2      =  THE CID OF THE LOGICAL UNIT
*          3      =  THE CONTENTS OF THE USERFLD (FROM
*                    THE NIB SPECIFIED AT OPNDST)
*          4      =  UNPREDICTABLE
*          5      =  THE ADDRESS OF A READ-ONLY RPL THAT IS
*                    USED TO DETERMINE WHAT KIND OF RESPONSE
*                    HAS BEEN RECEIVED
*
*  OUTPUT  =  A RESETTING TO CONTINUE-ANY MODE FOR ANY
*             LOGICAL UNIT FROM WHICH A RESPONSE IS RE-
*             CEIVED.
*
*  EXTERNAL REFERENCES = TESTCB, MODCB, RESETSR, SYNAD1, AND DUMP.
*
*  EXIT, NORMAL =  BR 14
*
*  EXIT, ABNORMAL =  DUMP
*
*  ATTRIBUTES  =  SERIALLY REUSABLE
*
*  REGS USED
*
*    3 = BASE
*    4 = A(PRPLR),IFGRPL
*    5 = A(VRPL),IFGRPL
*    6 = WORK,A(RESP1 PARM LIST)
*    8 = CID
*    9 = A(ACB)
*   13 = A(SAVE2)
*
*********************************************************************
```

```
RESP1      BALR   R3,R0                   ESTABLISH BASE
           USING  *,R3                    ESTABLISH ADDRESSABILITY
           L      R13,=A(SAVE2)           POINT TO EXIT ROUTINE SAVEAREA
           ST     R14,SAVE4               SAVE RETURN ADDRESS
           L      R6,=A(TPENDFLG)
           TM     0(R6),X'FF'             IS CLOSE ACB IN PROGRESS?
           BO     RETURN3                 YES, IGNORE RESPONSES
           LR     R6,R1                   SAVE PARAMETER LIST ADDRESS
           L      R9,0(R6)                PICK UP ACB ADDRESS
           L      R5,16(R6)                PUT ADDRESS OF READ-ONLY RPL IN 5
           LA     R4,PRPLR                INITIALIZE R4                     +
           DROP   R5                     FROM LOGON EXIT USE
           USING  IFGRPL,R4               BASE ON PRPLR
           MVC    RPLARG,4(R6)            MOVE CID TO PRPLR FOR RESETSR     |
           NI     RPLEXTDS,X'FF'-RPLNIB  TURN OFF NIB FLAG
           DROP   R4
           USING  IFGRPL,R5                BASE ON READY-ONLY RPL
           TM     RPLVTFL2,RPLEX          NORMAL RESPONSE?                  +
           BO     EXCEPTN                                                   +
**         TESTCB AM=VTAM,RPL=(R5),RESPOND=NEX, TEST FOR NORMAL RESPONS*
**              ERET=CANCEL3
**         BNE    EXCEPTN                 NOT NEX SO EXCEPTION (EX)
           B      RESET                                                     +
**         L      R8,4(R6)                IF NORMAL, PUT IDENTITY OF
**                                        LOGICAL UNIT IN REGISTER 8
**         MODCB  AM=VTAM,RPL=PRPLR,ARG=(R8) PUT IDENTITY INTO PRLPR
**         LTR    R15,R15                 MODCB WORK OK
**         BZ     RESET                   YES, BRANCH TO RESETSR
CANCEL3    ST     R1,R1CONTS3             OTHERWISE, MUST TERMINATE AND
           DUMP                           DUMP THE PROGRAM
*            RESET THE MODE TO CONTINUE-ANY MODE
*            THIS RESETSR WILL BE EXECUTED WITH NULL EFFECT IF THE LU
*              IS ALREADY IN CA MODE.  EG AS A RESULT OF MAINLINE CODE.
*            NOTE THAT THIS PROGRAM MAKES NO ATTEMPT TO RESEND A MESSAGE
*             WHEN A NEGATIVE RESPONSE IS RECEIVED.
RESET      RESETSR RPL=PRPLR,OPTCD=CA, RESET THE LOGICAL UNIT FOR          *
                RTYPE=DFSYN,ACB=(R9)      DFSYN INPUT.  OPTCD=(SYN,SPEC)
           LTR    R15,R15                 SEE IF RESETSR REQUEST ACCEPTED
           BNZ    CANCEL3                 IF NOT, GO TO TERMINATE AND DUMP
RETURN3    L      R14,SAVE4               RESTORE REG 14 RETURN ADDRESS
           BR     R14                     RETURN TO VTAM
*
EXCEPTN    EQU    *                       SET UP LINKAGE FOR SYNAD1
           STM    R14,R12,12(R13)         SAVE REGISTERS
           LA     R0,4                    INDICATE EXTRAORDINARY COMPLETION
           L      R15,=A(SYNAD1)
           LR     R1,R5                   POINT TO READ-ONLY RPL
           BALR   R14,R15                 CALL SYNAD ROUTINE
SYNRTURN   LM     R1,R12,24(R13)          RESTORE RESP EXIT REGS
           LTR    R15,R15                 SUCCESSFUL RECOVERY?
           BZ     RESET                   YES, ALLOW  NEXT TRANSACTION IN
*       SYNAD1 SETS R15=12 IF CLSDST WAS NECESSARY IN WHICH CASE
*        NO RESETSR SHOULD BE EXECUTED.
           B      RETURN3                 RETURN TO VTAM
```

```
PRPLR      RPL   AM=VTAM                      COULD BE SAME ONE AS PRPLCONN
*                                             SINCE BOTH ARE USED SYNCHRONOUSLY
*                                              IN VTAM EXITS.
SAVE4      DS    F                            SAVEAREA FOR EXIT RETURN ADDRESS
R1CONTS3   DC    F'0'                         SAVEAREA FOR REG 1 AT DUMP
           LTORG
```

```
***********************************************************************
*
* NAME = LERAD EXIT ROUTINE
*
* FUNCTION = HANDLE TELEPROCESSING-ORIENTED LOGICAL ERRORS
*
* ENTRY POINT = LERAD1
*
* INPUT
*    REGISTERS
*        0    =  RECOVERY ACTION RETURN CODE
*        1    =  RPL ADDRESS
*        2-12 =  UNPREDICTABLE
*        13   =  ADDRESS OF SAVE AREA SUPPLIED TO MACRO THAT
*                CAUSED LERAD ENTRY
*        14   =  RETURN ADDRESS
*        15   =  ADDRESS OF THIS ROUTINE'S ENTRY POINT
*
* OUTPUT = NONE
*
* EXTERNAL REFERENCES = TESTCB, GETVIS, FREEVIS, DUMP.
*
* EXIT, NORMAL = BR 14
*
* EXIT ABNORMAL = DUMP
*
* ATTRIBUTES = SERIALLY REUSABLE.
*
* REGS USED
*
*   2 = ADDRESS OF PARMLIST FOR TESTCB
*   3 = BASE
*   6 = A(RPL),IFGRPL
*  12 = RETURN ADDRESS
*  13 = A(SAVEAREA)
*
***********************************************************************
LERAD1   EQU   *                 LERAD EXIT-ROUTINE ENTRY POINT
         LR    R3,R15            REG 15 HAS ADDRESS OF LERAD1
         USING LERAD1,R3         SPECIFY BASE REGISTER
         LR    R12,R14           SAVE RETURN ADDRESS
         LR    R6,R1             SAVE RPL ADDRESS
         LA    R0,96             SAVEAREA SIZE                        +
***      LA    R0,96+LTESTLE     SAVEAREA + TESTCB PARMLIST SIZE
         GETVIS ADDRESS=(R1),LENGTH=(R0)  GET STORAGE FOR SAVEAREA
*OS      GETMAIN R,LV=(R0)      (NOT TESTED)
         LTR   R15,R15
         BNZ   LEOVERID          DUMP IF NO STORAGE AVAILABLE
         ST    R13,4(R1)         SAVE HIGH SAVEAREA ADDRESS IN NEW
         LR    R13,R1            POINT TO NEW SAVEAREA
        DROP  R5                    USED IN RESP EXIT
        USING IFGRPL,R6
*      THE FOLLOWING TESTCBS AVOID DUMPING ON ERRORS CAUSED BY
*       CLOSING A CONNECTION OR THE ACB.
         CLI   RPLFDB2,X'12'                                          +
**         LA    R2,96(R13)
**         TESTCB AM=VTAM,RPL=(R6),ERET=LEOVERID,                          *
**               FDBK2=18,MF=(G,(R2),LTESTLE) CLSDST IN PROGRESS?
          BE    IGNORE
           CLI   RPLFDB2,X'13'                                             |
**         TESTCB AM=VTAM,RPL=(R6),ERET=LEOVERID,                          *
**               FDBK2=19,MF=(E,(R2)) IS CID INVALID?
          BE    IGNORE
          CLI   RPLFDB2,X'60'       CLSDST WITH SYMBOLIC NAME FAILED?      |
**         TESTCB AM=VTAM,RPL=(R6),ERET=LEOVERID,                          *
**               FDBK2=96,MF=(E,(R2))
          BE    IGNORE              YES SO IGNORE
           B     LEOVERID           BRANCH AROUND DUMP ID
           DC    C'LERAD1'          DUMP ID
R1DUMP     DC    F'0'               REG 1 CONTENTS AT DUMP
LEOVERID   EQU   *
           ST    R1,R1DUMP          SAVE REG 1 FOR DUMP
           DUMP
IGNORE     EQU   *
           FREEVIS ADDRESS=(R1),LENGTH=(R0)  FREE SAVEAREA
*OS        FREEMAIN R,LV=(R0),A=(R1)      (NOT TESTED)
           LTR   R15,R15
           BNZ   LEOVERID
           SR    R15,R15            INDICATE SUCCESSFUL
           SR    R0,R0               COMPLETION OF LERAD.
           LR    R14,R12            RESTORE RETURN ADDRESS
           BR    R14                RETURN TO CALLER VIA VTAM
           LTORG
```

```
***********************************************************************
*
* NAME = TPEND EXIT ROUTINE
*
* FUNCTION = SET AN INDICATION FOR THE MAIN PROGRAM
*            TO CLOSE THE ACB AND TERMINATE
*
* ENTRY POINT = TPEND1
*
* INPUT
*     REGISTERS
*        0  =  UNPREDICTABLE
*        1  =  ADDRESS OF A 2-WORD PARAMETER LIST
*      2-13 =  UNPREDICTABLE
*        14 =  RETURN ADDRESS
*        15 =  ADDRESS OF THIS ROUTINE'S ENTRY POINT
*     PARAMETER LIST  - 2 WORDS
*        1  =  ADDRESS OF THE ACB
*        2  =  A VALUE INDICATING WHY TPEND WAS ENTERED
*
* OUTPUT  =  INDICATION TO CLOSE ACB SET FOR MAIN PROGRAM
*
* EXTERNAL REFERENCES = POST.
*
* EXIT, NORMAL = BR 14
*
* EXIT, ABNORMAL = NONE
*
* ATTRIBUTES = SERIALLY REUSABLE.
*
*  REGS USED
*
*    3 = BASE
*    4 = A(TPENDECB)
*
***********************************************************************
TPEND1    BALR  R3,R0
          USING *,R3
          ST   R14,TPENDS14
          LR     R6,R1                SAVE PARMLIST ADDRESS
          L      R4,=A(TPENDECB)      POINT TO MAINLINE'S CLOSEDOWN ECB
          POST  (R4)                  INDICATE TPEND REQUIRED
         L       R14,TPENDS14
          BR     R14                  RETURN TO VTAM
TPENDS14 DC    F'0' SAVE AREA FOR VTAM RETURN ADDRESS
          LTORG
```

```
************************************************************************
*
* NAME = SYNAD EXIT ROUTINE
*
* FUNCTION = HANDLE ERRORS AND SPECIAL CONDITIONS OTHER THAN
*            TELEPROCESSING LOGICAL ERRORS.  ATTEMPTS TO CLEAR
*            THE CONDITION OR CLOSE THE CONNECTION.
*
* ENTRY POINT = SYNAD1
*
* INPUT
*    REGISTERS
*           0    =  RECOVERY ACTION RETURN CODE
*           1    =  RPL ADDRESS (HIGH-ORDER BIT ON IF RECURSIVE ENTRY)
*         2-12   =  UNPREDICTABLE
*          13    =  ADDRESS OF SAVE AREA SUPPLIED PRIOR TO CAUSING
*                   SYNAD ENTRY
*          14    =  RETURN ADDRESS
*          15    =  ADDRESS OF THIS ROUTINE'S ENTRY POINT
*
* OUTPUT   =  A VALUE SET IN REGISTER 15:
*           0  =  SUCCESSFUL RECOVERY
*           8  =  EXCEPTION REQUEST RECEIVED
*          12  =  CLSDST PERFORMED
*
* EXTERNAL REFERENCES = SHOWCB, TESTCB, SESSIONC, MODCB, SEND,
*            RESETSR, CLSDST, GETVIS, GENCB, FREEVIS, DUMP, AND EXECRPL.
*
* EXIT, NORMAL = BR 14
*
* EXIT, ABNORMAL = DUMP
*
* ATTRIBUTES = QUASI-REENTERABLE.  THIS ROUTINE IS REENTERED IF
*              A MACRO IT ISSUES FAILS.  IN THIS CASE, INDICATED BY THE
*              HIGH-ORDER BIT OF REG 1 BEING ON ON ENTRY TO SYNAD1,
*              THE PROGRAM TERMINATES AND A DUMP IS REQUESTED.
*              OTHERWISE IF SYNAD IS REENTERED PROCESSING CONTINUES.
*
*  REGS USED
*
*    2 = A(PARMLIST FOR MANIP MACROS)
*    3 = BASE
*    5 = A(RPL),IFGRPL
*    6 = A(GETVIS RPL),IFGRPL
*    7 = REG0 RETURN CODE
*    8 = REG15 RETURN CODE, A(PARMLIST FOR MANIP MACROS)
*    9 = A(PACB)
*   10 = LINKAGE TO SESSIONC
*   11 = CID
*   12 = RETURN ADDRESS
*   13 = A(GETVIS SAVEAREA),SAVE5
*
************************************************************************
SYNAD1   BALR  R3,R0
         USING *,R3
         LR    R4,R0               SAVE PROGRAMMER ACTION CODE
         LR    R5,R1               SAVE RPL ADDRESS
         LR    R12,R14             SAVE RETURN ADDRESS
         LA    R0,96                                                   +
**       L     R0,SGWORKL          SAVEAREA | MANIP MACRO PARMLISTS
         GETVIS ADDRESS=(R1),LENGTH=(R0) GET STORAGE
*OS      GETMAIN R,LV=(R0)   (NOT TESTED)
         LTR   R15,R15
         BNZ   CANCEL4
         ST    R13,4(R1)            SAVE HIGH SAVEAREA ADDRESS
         LR    R13,R1               POINT TO NEW SAVEAREA
         DROP  R6                   USED IN LERAD EXIT
         USING  IFGRPL,R5
         USING SDSECT,R13           SET BASE FOR REENTRANT WORKAREA
*      CHECK FOR RECURSIVE ENTRY TO SYNAD1
         ST    R5,REGNWORK
         TM    REGNWORK,X'80'       IS THIS RECURSIVE ENTRY TO SYNAD?
         BO    CANCEL4              YES -- CANCEL
         OI    REGNWORK,X'80'       NO  -- INDICATE RECURSION
         L     R5,REGNWORK          SAVE RPL ADDRESS
*      IF WE LEAVE SYNAD1 CORRECTLY THE RECURSION FLAG WILL
*       BE OVERWRITTEN BY THE NEXT RPL-BASED MACRO EXECUTED
*       OUTSIDE OF SYNAD.
         LA    R0,SRPLEND-SRPL      SET LENGTH OF RPL IN R0           +
         GETVIS ADDRESS=(R1),       GET ENOUGH STORAGE FOR            +*
              LENGTH=(R0)            AN RPL.                          +
*OS      GETMAIN R,LV=(R0)
**       GENCB AM=VTAM,BLK=RPL      GET A NEW RPL
         LTR   R15,R15
         BNZ   CANCEL4
         MVC   0(SRPLEND-SRPL,R1),SRPL  COPY SRPL TO GETVIS STORAGE   +
         ST    R0,SAVE6             SAVE LENGTH FOR FREEVIS
         ST    R1,REGNWORK          POINT TO SYNAD1'S OWN RPL
         OI    REGNWORK,X'80'       SET HIGH-ORDER BIT OF R6
         L     R6,REGNWORK           (RPLSYN ADDRESS) FOR RECURSION.
         L     R9,=A(PACB)          PICK UP ADDRESS OF ACB
         L     R11,RPLARG                                              +
**       LA    R1,SHOWL1            POINT TO 1ST PARMLIST WORKAREA
**       SHOWCB  AM=VTAM,RPL=(R5),AREA=(S,SARG),                       *
**            LENGTH=4,FIELDS=ARG,  MOVE CID TO SARG                   *
**            MF=(G,(R1),SHOWL1E)
**       LTR   R15,R15
**       BNZ   CANCEL4
**       L     R11,SARG             PICK UP CID OF THE LU
         CH    R4,=H'16'            IS IT OVER MAX FOR SYNAD?
         BH    CANCEL4              YES, GIVE UP
         B     *+4(R4)              USE ACTION CODE IN BRANCH TABLE
         B     SNORM                CODE=X'00' SHOULD NOT OCCUR
         B     SXTRA                CODE=X'04' EXTRAORDINARY COMPLETION
         B     SRETRY               CODE=X'08' RETRYABLE
         B     SDAMAGE              CODE=X'0C' DAMAGE
         B     SENVIR               CODE=X'10' ENVIRONMENT ERROR
*
SNORM    SR    R7,R7                INDICATE SUCCESSFUL
         SR    R8,R8                 COMPLETION.
```

```
SABNORM  L     R0,SAVE6                  LENGTH OF STORAGE TO BE FREED
         SLL   R6,1                      GET RID OF HIGH-ORDER BIT
         SRL   R6,1                       FROM R6 FOR FREEVIS.
*        LA    R6,0(R6)                   THIS DOES THE SAME AS THE SLL,SRL.
         FREEVIS ADDRESS=(R6),LENGTH=(R0)  FREE SYNAD'S RPL
*OS      FREEMAIN R,LV=(R0),A=(R6)    (NOT TESTED)
         LTR   R15,R15
         BNZ   CANCEL4
         LR    R1,R13                    POINT TO SAVEAREA TO BE FREED
         L     R13,SAVE5+4               RESTORE HIGH SAVE AREA ADDRESS
         LA    R0,96                                                   +
**       L     R0,SGWORKL
         FREEVIS ADDRESS=(R1),LENGTH=(R0)  FREE SAVEAREA
*OS      FREEMAIN R,LV=(R0),A=(R1)    (NOT TESTED)
         LTR   R15,R15
         BNZ   CANCEL4
         LR    R0,R7
         LR    R15,R8
         LR    R14,R12                   RESTORE RETURN ADDRESS
         BR    R14                       RETURN TO NSI VIA VTAM
*
SXTRA    EQU   *                         EXTRAORDINARY COMPLETION
*  THIS SHOWCB IS NOT NECESSARY WITH DSECTS                            +
**       LA    R2,SHOWL1
**       SHOWCB  AM=VTAM,RPL=(R5),  DISPLAY FEEDBACK FIELDS            *
**            AREA=(S,SHOWWORK),                                       *
**            FIELDS=(FDBK2,SSENSMI),                                  *
**            LENGTH=8,MF=(G,(R2),SHOWL2E)
**       LTR   R15,R15
**       BNZ   CANCEL4
SXPATHE  TM    RPLSSEI,RPLPATHI                                        +
         BO    SDISCONN                                                +
**PATHE  TESTCB  AM=VTAM,RPL=(R5),  WAS IT PATH ERROR?                 *
**            SSENSEI=PATH,                                            *
**            ERET=CANCEL4,MF=(G,(R2),TESTL1E)
**       BE    SDISCONN                  YES--GO TO DISCONNECT LU
         CLI   RPLFDB2,X'03'
**       CLC   SFDBK2,=F'3'              NO, IS REASON CODE 3?
         BE    EXMSG                     YES--EXCEPTION MESSAGE RECEIVED
*                                        NO--EXCEPTION RESPONSE RECEIVED
         LA    R10,SNORM                 PREPARE FOR NORMAL RETURN
*
         DROP  R5
        USING  IFGRPL,R6
SESSIONC SESSIONC  RPL=(R6),ACB=(R9),ARG=(R11),    CLEAR SESSION        *
              CONTROL=CLEAR,STYPE=REQ,OPTCD=SYN
         LTR   R15,R15
         BNZ   SDISCONN
*** THE FOLLOWING STSN IS REDUNDANT NOW CLEAR RESETS SEQUENCE TO 0.    |
*       THE STSN AVOIDS OUT OF SEQUENCE SITUATIONS.
         SESSIONC  RPL=(R6),CONTROL=STSN,                               *
              OBSQAC=SET,OBSQVAL=0,IBSQAC=SET,IBSQVAL=0
         LTR   R15,R15
         BNZ   SDISCONN
         SESSIONC  RPL=(R6),CONTROL=SDT          START DATA TRAFFIC
         LTR   R15,R15                   SUCCESSFUL RECOVERY?
         BNZ   SDISCONN                  NO, DISCONNECT
```

```
          BR    R10                    YES, RETURN TO CALLER
          DROP  R6
          USING IFGRPL,R5
*
EXMSG     EQU   *         THIS CANNOT BE REACHED FROM THE RESP EXIT
          TM    RPLVTFL2,RPLNFME                                      +
          BO    STBAL                                                 +
**        LA    R1,SHOWL1+TESTL1E   POINT TO NEXT AVAILABLE SPACE
**        TESTCB AM=VTAM,RPL=(R5),ERET=SDISCONN,   NO RESPONSE        *
**             RESPOND=NFME,MF=(G,(R1),TESTL2E)        WANTED?
**        BE    STBAL                  YES--BYPASS SENDING RESPONSE
*  MOVE SSENSEI TO SSENSEO
          MVC   RPLSSEO,RPLSSEI                                       +
**        LA    R8,SHOWL1+TESTL1E   POINT TO NEXT AVAILABLE SPACE
**  SET UP LIST FORM OF MODCB
**        MODCB AM=VTAM,RPL=(R5),SSENSEO=CPM,MF=(L,(R8),MODL1E)
**        TESTCB AM=VTAM,RPL=(R5),ERET=SDISCONN,SSENSEI=CPM,          *
**            MF=(E,(R2))
**        BNE   ST2
**        MODCB  AM=VTAM,RPL=(R5),SSENSEO=CPM,MF=(E,(R8))
**        LTR   R15,R15
**        BNZ   SDISCONN
**        B     STEND
**2       TESTCB AM=VTAM,RPL=(R5),ERET=SDISCONN,SSENSEI=STATE,        *
**             MF=(E,(R2))
**        BNE   ST3
**        MODCB  AM=VTAM,RPL=(R5),SSENSEO=STATE,MF=(E,(R8))
**        LTR   R15,R15
**        BNZ   SDISCONN
**        B     STEND
**3       TESTCB AM=VTAM,RPL=(R5),ERET=SDISCONN,SSENSEI=FI,           *
**             MF=(E,(R2))
**        BNE   ST4
**        MODCB  AM=VTAM,RPL=(R5),SSENSEO=FI,MF=(E,(R8))
**        LTR   R15,R15
**        BNZ   SDISCONN
**        B     STEND
**4       TESTCB AM=VTAM,RPL=(R5),ERET=SDISCONN,SSENSEI=RR,           *
**             MF=(E,(R2))
**        BNE   STEND
**        MODCB  AM=VTAM,RPL=(R5),SSENSEO=(RR,MF=(E,(R8))
**        LTR   R15,R15
**        BNZ   SDISCONN
STEND     EQU   *
          MVC   RPLSSMO,RPLSSMI
**        L     R7,SSENSMI             PICK UP SSENSMI FOR EXCEPTION RESP
          SEND  RPL=(R5),STYPE=RESP,  SEND THE EXCEPTION RESPONSE      *
               OPTCD=SYN                                              +
**             SSENSMO=(R7),             USING THE RECEIVE RPL        *
**             OPTCD=SYN                 (RECANY HAS OPTCD=ASY)
         LTR   R15,R15
         BNZ   SDISCONN
*  THE NEXT OPERATION WILL BE A SESSIONC CLEAR WHICH MAY OVERTAKE THE
*    RESPONSE WE HAVE JUST SENT.  HOWEVER THE LU SHOULD UNDERSTAND.
          DROP  R5
         USING  IFGRPL,R6
STBAL    BAL    R10,SESSIONC          GO THROUGH CLEAR AND SDT
         RESETSR  RPL=(R6),RTYPE=DFSYN,    RESTORE TO CA MODE      *
              OPTCD=(SYN,CA)
         LTR    R15,R15
         BNZ    CANCEL4
         LA     R8,8                 SIGNAL UNSUCCESSFUL COMPLETION
         B      SABNORM              RETURN TO RECANY
*
SDISCONN EQU    *                    UNRECOVERABLE ERRORS
         CLSDST  RPL=(R6),ACB=(R9),ARG=(R11)
         LTR    R15,R15
         BNZ    CANCEL4
         LA     R8,12               SIGNAL UNSUCESSFUL RECOVERY
         B      SABNORM
*
CANCEL4  DUMP
SRETRY   EQU    *
*** IF POSSIBLE A PROGRAM SHOULD PAUSE BEFORE RETRY TO GIVE TIME FOR  |
***  VTAM STORAGE TO BE FREED.  IDEALLY DO A RECEIVE FIRST.           |
*** A PROGRAM MUST BE DESIGNED TO KNOW THAT NO RPL PARMS CAN HAVE BEEN|
***  ALTERED BY VTAM BEFORE INDICATING RETRY IS POSSIBLE.             |
         EXECRPL  RPL=(R5)           RETRY FAILED MACRO
         LTR    R15,R15
         BNZ    CANCEL4
         B      SNORM                RETURN TO ORIGINAL NSI
*
SDAMAGE  EQU    *
         CLI    RPLREQ,RPLRCVCD                                        +
**       LA     R2,SHOWL1            POINT TO PARMLIST SPACE
**       TESTCB AM=VTAM,RPL=(R5),ERET=SDISCONN,   IS IT A RECEIVE      *
**            REQ=35,MF=(G,(R2),TESTL3E)           THAT IS FAILING?
         BNE    SNORM                NO, PRETEND COMPLETION WAS OK
         LA     R8,16                YES, SET R15 CODE REG NON-ZERO
         B      SABNORM              RETURN TO NSI, WHICH MAY BE ABLE
*                                    TO IGNORE THE ERROR
*
SENVIR   EQU    *                                                      +
         CLI    RPLREQ,RPLSNDCD
**       LA     R2,SHOWL1            POINT TO NEXT AVAILABLE SPACE
**       TESTCB AM=VTAM,RPL=(R5),ERET=CANCEL4,  IS IT A SEND?          *
**            REQ=34,MF=(G,(R2),TESTL4E)
         BE     SDISCONN             ATTEMPT TO CLSDST
         CLI    RPLREQ,RPLRSRCD                                        +
**       TESTCB AM=VTAM,RPL=(R5),ERET=CANCEL4,  IS IT A RESETSR?       *
**            REQ=36,MF=(E,(R2))
         BE     SDISCONN             ATTEMPT TO CLSDST LU
         LA     R8,20                SET NON-ZERO CODE AND ALLOW IN-LINE
         B      SABNORM               CODE TO RECOVER. (MAY BE AN OPNDST
*                                     OR INQUIRE)
**WORKL  DC     A(96+TESTL1E+TESTL2E) SGWORKL MAX SIZE NEEDED
*
SDSECT   DSECT
SAVE5    DS     18F                  NEW SAVEAREA
SHOWWORK DS     0F
```

```
SFDBK2    DS     F                     SPECIFIC REASON CODE
SSSENSMI  DS     F                     SYSTEM SENSE MODIFIER INPUT
REGNWORK  DS     F                     FOR RETRYABLE ERRORS
SAVE6     DS     F                     LENGTH OF RPL FROM GENCB
SARG      DS     F                     ARG - CID VALUE
**OWL1    DS     OF                    SHOWL1 - START OF MANIP PARMLISTS
SAMP1     CSECT
SRPL      RPL    AM=VTAM               RPL TO BE COPIED TO GETVIS STORAGE+
SRPLEND   EQU    *                     END OF SRPL FOR LENGTH CALC.       +
          LTORG
```

```
***********************************************************************
*
* NAME = LOSTERM EXIT ROUTINE
*
* FUNCTION = HANDLE SITUATIONS IN WHICH A LOGICAL UNIT HAS
*            UNEXPECTEDLY BECOME UNAVAILABLE
*
* ENTRY POINT = LOSTERM1
*
* INPUT
*    REGISTERS
*          0    =  UNPREDICTABLE
*          1    =  ADDRESS OF A 4-WORD PARAMETER LIST
*        2-13   =  UNPREDICTABLE
*          14   =  RETURN ADDRESS
*          15   =  ADDRESS OF THIS ROUTINE'S ENTRY POINT
*    PARAMETER LIST - 4 WORDS
*          1    =  ADDRESS OF THE ACB
*          2    =  THE CID OF THE LOGICAL UNIT
*          3    =  THE CONTENTS OF THE USERFLD (FROM THE NIB
*                  SPECIFIED AT OPNDST)
*          4    =  A VALUE INDICATING WHY LOSTERM WAS ENTERED
*
* OUTPUT  =  DISCONNECTION OF THE LOGICAL UNIT
*
* EXTERNAL REFERENCES = CLSDST, DUMP.
*
* EXIT, NORMAL = BR 14
*
* EXIT, ABNORMAL = DUMP
*
* ATTRIBUTES = SERIALLY REUSABLE.
*
*  REGS USED
*
*    3 = BASE
*    4 = A(PRPLCONN)
*    5 = A(ACB)
*    6 = CID
*    7 = A(TPENDFLG)
*   13 = A(SAVE2)
*
*
***********************************************************************
LOSTERM1  BALR  R3,R0
          USING *,R3
          L     R13,=A(SAVE2)       POINT TO EXIT-ROUTINE SAVEAREA
          ST    R14,SAVELOST
          L     R4,=A(PRPLCONN)     POINT TO OPNDST/CLSDST RPL
          L     R5,0(R1)            PICK UP ACB ADDRESS
          USING IFGRPL,R4           BASE ON PRPLCONN
          MVC   RPLUSFLD,8(R1)      MOVE USER FIELD FOR PJWM
          L     R6,4(R1)            PICK UP CID OF LOST TERMINAL
          L     R7,=A(TPENDFLG)
          TM    0(R7),X'FF'         IS CLOSE ACB IN PROGRESS?
```

```
         BO    RETURNL             YES, IGNORE LOSTERM NOTIFICATION
         LR    R8,R1        POINT TO PARMLIST
*** THE FOLLOWING ALLOWS FOR POSSIBLE RECOVERABLE ERRORS. IDEALLY THE |
***  PROGRAM SHOULD CLSDST AT CODE 18 IF THE LU INITIATES LOGONS.     |
         CLI   15(R8),X'18'        IS RESTART IN PROGRESS?
         BE    RETURNL
*** THE PROGRAM MIGHT CHOOSE TO TEST OTHER VALUES AND TAKE SPECIAL
***  ACTIONS. EG. NOT ISSUING CLSDST FOR THE FIRST OCCURRENCE OF CODE
***  X'24' IF IT MAY HAVE BEEN A TEMPORARY HOLD UP THAT CAUSED THE
***  BUFFER LIMIT TO BE EXCEEDED. ALL SUCH CHOICES ARE APPLICATION
***  ENVIRONMENT DEPENDENT.
LOSTCLOS EQU   *
         CLSDST  RPL=(R4),ACB=(R5),ARG=(R6),OPTCD=(RELEASE,SYN)
         LTR   R15,R15
         BZ    RETURNL
         DUMP
RETURNL  L     R14,SAVELOST
         BR    R14                  RETURN TO VTAM
*
SAVELOST DC    F'0'
         LTORG
         END   SAMP1
```

Appendix E. Example of Authorized Path

This sample program, SAMP2, shows an application program in an OS/VS2 system using the authorized path feature under the control of both a TCB (task control block) and an SRB (service request block). The logic for this sample application program is similar to the logic for Sample Program 1 in Chapter 11. Chapter 11 also lists coding rules for using authorized forms of SEND, RECEIVE, RESETSR, and SESSIONC.

SAMP2 uses the authorized forms of the SEND, RECEIVE, and RESETSR macro instructions to perform I/O processing. SAMP2 shows:

How to enter supervisor state

Which OS/VS2 MVS system macro instructions to use

This sample can be used as a guideline for coding application programs that plan to use authorized path; the program is not intended to be coded and used by an installation.

Notes about SAMP2

SAMP2 physically consists of one program, but logically consists of two programs. The first logical program is labeled "AUTHPATH," and the second logical program is labeled "AUTHEXIT."

AUTHPATH begins by opening an ACB and connecting itself to a logical unit. In order to use the authorized forms of SEND, RECEIVE, RESETSR, or SESSIONC, the application program must have authorization to change from problem program state to supervisor state. (See the publication *OS/VS2 SPL: Supervisor* for a description of how to become an authorized program.)

SAMP2 changes into supervisor state by issuing the OS/VS2 MVS system macro instruction MODESET. The MODESET obtains the zero protection key needed to use authorized path. To schedule an SRB, an application must be in supervisor state with a key of zero. SAMP2 has now met these requirements.

At this point, SAMP2 departs from normal ACF/VTAM application programming. It is now operating as a system program, using a system key. The application program issues the RECEIVE macro instruction in its asynchronous form with an exit routine specified. The operand BRANCH=YES causes authorized path to be used. The exit routine will run under control of an SRB.

The exit routine, AUTHEXIT, issues the CHECK macro, which delimits processing of the RECEIVE macro issued in the main program. Then the input is tested for three possibilities:

Echo

No echo

An error condition

If an echo is desired, the SEND macro instruction is used to return the data just received. This SEND uses the authorized path because it is issued under control of the SRB under which the exit routine was scheduled. The main program is then posted to continue issuing the RECEIVE.

If no echo is desired, and an error has not been encountered, the RESETSR macro is issued to change the logical unit back to continue-any mode so that it can be addressed by the next issuance of the RECEIVE macro. RESETSR takes the authorized path because it is issued under control of the SRB under which the exit routine was scheduled.

If an error condition was encountered, the main program is posted to issue CLSDST and CLOSE ACB after offering the user the option of an ABEND dump.

Otherwise, the main program is posted to continue issuing the RECEIVE ANY to accept data from the logical unit. Since the exit routine runs under an SRB, the branch entry to POST, rather than the POST SVC, must be used, which in turn requires obtaining and releasing the local lock via the SETLOCK macro.

The main program checks the input for the string "LOGOFF" and, if it finds it, issues CLSDST and CLOSE ACB to shut down the application program. (Notice that the CLSDST macro uses the BRANCH=NO operand to turn off the RPLBRANC flag turned on by the RECEIVE using the BRANCH=YES operand.) If the "LOGOFF" string is not found, the main program continues issuing the RECEIVE and the exit will again get control.

SAMP2 Assembler Language Code

```
AUTHPATH CSECT
         PRINT NOGEN
R15      EQU   15
R14      EQU   14
R13      EQU   13
R12      EQU   12
R11      EQU   11
R10      EQU   10
R9       EQU   9
R5       EQU   5
R4       EQU   4
R1       EQU   1
R0       EQU   0
         SAVE  (14,12),T,*
         LR    R12,R15          * GET BASE
         USING AUTHPATH,R12     * COVER
         LR    R9,R13           * MOVE SAVE PTR
         LA    R13,SAVE         * PT TO SAVE AREA
         ST    R13,8(0,R9)      * FORWARD PTR
         ST    R9,4(0,R13)      * BACKWARD PTR
         LA    R4,AUTHRPL       * BASE FOR RPL
         USING IFGRPL,R4        * COVER RPL
         WTO   'AUTHPATH APPLICATION ENTERED' * TELL OPER WERE HERE
*
*        OPEN THE ACB
*
         OPEN  AUTHACB          * OPEN VTAM ACB
         MVI   THRU+3,4         * OUTPUT MSG 1, OPEN FAILED
         LTR   R15,R15          * GOOD
         BNZ   BAD              * NO
*
*        CONNECT TO TERMINAL
*
```

```
         OPNDST RPL=AUTHRPL,OPTCD=(SYN,ACQUIRE) * CONNECT UP
         MVI THRU+3,8        * OUTPUT MSG 2, OPEN DEST FAILED
         LTR    R15,R15      * GOOD
         BNZ    BAD          * NO
*
*
*        MODESET TO SUP STATE
*
         MODESET MODE=SUP           * SUPERVISOR STATE
*
*        ISSUE RECEIVE MACRO
*
AUTHRECV RECEIVE RPL=AUTHRPL,OPTCD=(ASY,Q,ANY,CS),BRANCH=YES,
               AREA=INPUT00,AREALEN=100,EXIT=AUTHEXIT,RTYPE=DFSYN
         MVI    THRU+3,12      * OUTPUT MSG 3, RECEIVE VALID CHECK FAIL
         LTR    R15,R15        * CHECK RETURN CODE FROM RECEIVE
         BNZ    BAD            * NOT 0, GO HANDLE SITUATION
*
*  HERE THIS APPLICATION COULD BE DOING SOME TYPE
*  OF PROCESSING BEFORE NOTIFICATION FROM AUTHEXIT.
*
*
         WAIT   1,ECB=THRU     * WAIT FOR AUTH EXIT
         CLI    THRU+3,0       * EVERYTHING OK IN EXIT?
         BNZ    BAD            * NO, HANDLE IT.
         CLC    LOGMSG,INPUT00+6 * USER WANT TO LOG OFF?
         BE     CLOSE1         * HE WANTS TO QUIT
         XC     THRU,THRU      * CLEAR ECB
         MVI    INPUT00,X'40'
         MVC    INPUT00+1(99),INPUT00 * CLEAR INPUT AREA
         B      AUTHRECV       * REPEAT RECEIVE
*
*        CLOSE UP AND GO HOME
*
CLOSE1   CLSDST RPL=AUTHRPL,OPTCD=SYN,BRANCH=NO * DISCONNECT FROM TERM
CLOSE2   CLOSE AUTHACB         * CLOSE ACB
CLOSE3   LA     R1,MSG0        * ISSUE ENDED MESSAGE
         WTO    MF=(E,(1))
         L      R13,4(0,R13)   * POINT
         RETURN (14,12),T,RC=0
BAD      L      R5,THRU        * USE POST CODE AS INDEX
         L      R1,MSGS(R5)    * PT TO APPROPRIATE MSG
         WTO    MF=(E,(1))
         WTOR   'ENTER ''Y'' FOR ABEND DUMP',REPLY,1,WTORECB
         WAIT   1,ECB=WTORECB
         CLI    REPLY,C'Y'       * DOES HE WANT A DUMP?
         BNE    CHEKCLOS         * DETERMINE PROPER CLOSE
         ABEND 100,DUMP,STEP     * KILL IT
CHEKCLOS CLI THRU+3,8            * HOW FAR DID WE GET
         BH     CLOSE1           * PRETTY FAR, CLOSE EVERYTHING
         BE     CLOSE2           * SO SO, JUST CLOSE ACB
         BL     CLOSE3           * TCH TCH
AUTHEXIT DS     0H
         USING AUTHEXIT,R15    * TEMPORARY BASE
         STM    14,12,SAV2+12  * SAVE CALLERS REGS
         LR     R12,R15        * LOAD BASE
         DROP  15
         USING AUTHEXIT,R12
         LA    R13,SAV3        * VTAM DOES NOT PASS A SAVE AREA
*                              * SO SKIP STANDARD LINKAGE
         CHECK RPL=AUTHRPL     * CHECK STATUS OF REQUEST
         MVI   POSTCODE+3,16   * SET CODE NOT EQUAL 0?
         LTR   R15,R15         * RETURN CODE EQUAL 0?
         BNZ   GOPOST          * NO, GO HANDLE SITUATION
*
*        THE SEND BELOW WILL USE THE AUTHORIZED PATH BECAUSE
*        IT IS ISSUED UNDER CONTROL OF THE SRB SCHEDULED BY
*        VTAM AS THE EXIT FOR THE AUTHORIZED PATH RECEIVE.
*
         MVC   OUTPUT01,INPUT00 * MOVE WHAT WE READ TO OUTPUT AREA
         TM    INPUT00,X'01'   * ERROR RESPONSE REQUESTED?
         BC    1,CLOSIT        * YES, CLOSE DEST AND QUIT
         TM    INPUT00,X'04'   * DOES APB WANT AN ECHO?
         BC    8,NOECHO        * NOPE
*
*  ECHO WAS REQUESTED
*
         MVI   OUTPUT01,X'82' * SET ECHO RESPONSE CODE
         XC    OUTPUT01+1(5),OUTPUT01+1 * CLEAR OTHER CONTROL
         CLC   LOGMSG,INPUT00+6  * USER WANT TO LOGOFF?
         BNE   CONTINUE          * NO
         MVC   OUTPUT01+13(35),OUTPUT02  * MOVE IN FINAL MSG
CONTINUE DS    OH
*
         SEND OPTCD=(SYN,CA),CONTROL=DATA,STYPE=REQ,RTYPE=DFSYN,      X
               RECLEN=100,AREA=OUTPUT01,RPL=AUTHRPL,POST=SCHED,       X
               RESPOND=(NEX,NFME,NRRN) * SEND SOME DATA OUT
         MVI   POSTCODE+3,20   *  SET UP MSG5, SEND VALIDITY FAILED
         LTR   R15,R15         * RC=0?
         BNZ    GOPOST         *  NO, POST MAINLINE
         MV    POSTCODE+3,24   * SET MSG6- RPL RETURN CODE
         CLI   RPLRTNCD,0      * RETURN CODE OF ZERO?
         BNE   GOPOST          * POST MAINLINE
         CLI   RPLFDB2,0       * CHECK FEEDBACK CODE
         BNE   GOPOST          * POST MAINLINE
AOK      XC    POSTCODE,POSTCODE * MADE IT THRU WITHOUT A HITCH
*
*        BRANCH ENTRY TO POST REQUIRES LOCAL LOCK
*
GOPOST   DS    OH
LOCK     SETLOCK OBTAIN,TYPE=LOCAL,MODE=UNCOND,REGS=USE,              X
               RELATED=(LOCAL,AUTHPATH(UNLOCK))
         L     R10,POSTCODE    * GET POST CODE
         LA    R11,THRU        * POINT TO ECB
         L     R15,16          * CVT PTR
         L     R15,CVTOPT02-CVT(0,R15) * BR ENT TO POST
         BALR  R14,R15         * POST ECB
UNLOCK   SETLOCK RELEASE,TYPE=LOCAL,REGS=USE,                         X
               RELATED=(LOCAL,AUTHPATH(LOCK))
AUTHEXO  L     R13,4(0,R13)    * PT TO SAVE AREA
         RETURN (14,12),T,RC=0
```

```
NOECHO DS OH                    * APB DOES NOT WANT AN ECHO
*  MUST ISSUE RESETSR TO SWAP MODE BACK TO CONTINUE ANY
         RESETSR RPL=AUTHRPL,OPTCD=(SYN,CA),RTYPE=DFSYN
         MVI   POSTCODE+3,28    * MESSAGE 7
         LTR   R15,R15          * RESET OK?
         BNZ   GOPOST           * NOPE
         B     AOK              * OK, CONTINUE
CLOSIT   DS    OH               * ERROR RESPONSE- CLOSE DEST
         MVI   POSTCODE+3,32    * SET SPECIAL CLOSE DEST CODE
         B     GOPOST           * POST MAINLINE
THRU     DC    F'0'             * ECB TO WAIT ON SRB
WTORECB  DC    F'0'             * ECB FOR WTOR
SYSDATE  DC    CL8'&SYSDATE'    * DATE OF ASSEMBLY
SAVE     DC    18F'0'
SAV2     DS    OF
         DC    2F'0'
         DC    A(SAV3)
         DC    15F'0'
SAV3     DS    OF
         DC    F'0'
         DC    A(SAV2)
         DC    16F'0'
REPLY    DC    C' '             * USERS REPLY TO DUMP MSG
LOGMSG   DC    CL7'LOGOFF '     * USER WANTS TO LOGOFF
POSTCODE DC    F'0'             * POST CODE GOES HERE
OUTPUT01 DC    CL100' '
INPUT00  DC    CL100' '
OUTPUT02 DC    CL35' *** AUTHPATH WORKS, PASS IT ON ***'
APPL5ID  DC    X'05',CL5'APPL5'        * ID AND PASSWORD

MSGS     DC    A(MSG0,MSG1,MSG2,MSG3,MSG4,MSG5,MSG6,MSG7,MSG8)
MSG0     WTO   'AUTHPATH ENDED',ROUTCDE=(1),DESC=(5),MF=L
MSG1     WTO   'OPEN ACB FAILED',ROUTCDE=(1),DESC=(5),MF=L
MSG2     WTO   'OPEN DESTINATION FAILED',ROUTCDE=(1),DESC=(5),MF=L
MSG3     WTO  'RECEIVE VALIDITY CHECK FAILED',ROUTCDE=(1),DESC=(5),MF=L
MSG4     WTO   'RECEIVE FAILED, EXIT ENTERED',ROUTCDE=(1),DESC=(5),MF=L
MSG5     WTO   'SEND VALIDITY CHECK FAILED',ROUTCDE=(1),DESC=(5),MF=L
MSG6     WTO   'SEND FAILED',ROUTCDE=(1),DESC=(5),MF=L
MSG7     WTO   'RESETSR FAILED',ROUTCDE=(1),DESC=(5),MF=L
MSG8     WTO   'APB REQUESTED CLOSE DEST',ROUTCDE=(1),DESC=(5),MF=L
         DS    OH
PATCH    DC   10CL8'PATCHES'
AUTHRPL  RPL   ACB=AUTHACB,AM=VTAM,NIB=AUTHNIB2
AUTHACB  ACB   AM=VTAM,APPLID=APPL5ID,PASSWD=APPL5ID,MACRF=NLOGON
AUTHNIB2 NIB   NAME=CTJ10LU1,MODE=RECORD,USERFLD=C' NEW',LISTEND=YES
         IFGRPL
         IFGRPLVT
         IHASRB
         IHAPSA
         IHAFRRS
         CVT   DSECT=YES,LIST=YES
         END
```

Glossary

This glossary defines terms and abbreviations that are important in this book. It does not include terms previously established for IBM operating systems and IBM products used with ACF/VTAM. Additional terms can be found by referring to the index, to prerequisite and corequisite books, and to the *IBM Data Processing Glossary*, GC20-1699.

IBM is grateful to the American National Standards Institute (ANSI) for permission to reprint its definitions from the *American National Standard Vocabulary for Information Processing* (Copyright © 1970 by American National Standards Institute, Incorporated), which was prepared by Subcommittee X3K5 on Terminology and Glossary of the American National Standards Committee X3. A complete commentary taken from ANSI is identified by an asterisk that appears between the term and the beginning of the commentary; a definition taken from ANSI is identified by an asterisk after the item number for that definition.

The symbol *ISO* at the beginning of a definition indicates that it has been discussed and agreed upon at meetings of the International Organization for Standardization Technical Committee 97/Subcommittee 1 (Data Processing), and has also been approved by ANSI.

The symbol *SC1* at the beginning of a definition indicates that it is reprinted from an early working document of ISO Technical Committee 97/Subcommittee 1 and that final agreement has not yet been reached among its participating members.

A

ACB. Access method control block.

ACB name. (1) The name of an ACB macro instruction. (2) A name that can be specified in the ACBNAME parameter of an APPL statement. This name allows an ACF/VTAM application program that is used in more than one domain to specify the same application program identification (pointed to by the APPLID parameter of the program's ACB statement) in each copy. ACF/VTAM knows the program by both its ACB name and its network name (the name of the APPL statement). Program users within the domain can request logon using the ACB name or the network name; program users in other domains must use the network name (which must be unique in the network). Contrast with *network name.*

accept. In ACF/VTAM, to connect a terminal to a primary application program as the result of a logon. The logon may be originated by the terminal, the network operator, another primary application program, or ACF/VTAM. Contrast with *acquire (1).*

access method control block (ACB). A control block that links an application program to VSAM or ACF/VTAM.

accounting exit routine. In ACF/VTAM, an optional, user-written routine that collects statistics about connections and disconnections in the communication network.

ACF. Advanced Communications Function.

ACF/VTAM. Advanced Communications Function for the Virtual Telecommunications Access Method.

ACF/VTAM application program. A program that has opened an ACB to identify itself to ACF/VTAM. It can now issue ACF/VTAM macro instructions.

ACF/VTAM definition. The process of defining the communication network to ACF/VTAM (which is called "network definition") and modifying IBM-defined characteristics to suit the needs of the user.

ACF/VTAM definition library. The DOS/VS files or OS/VS data sets that contain the definition statements and start options filed during ACF/VTAM definition.

ACF/VTAM system. The resources defined to and controlled by ACF/VTAM.

acquire. (1) In relation to an ACF/VTAM application program, to connect a terminal to the application program in the absence of a logon. The connection occurs at the primary application program's initiative. Contrast with *accept.* (2) In relation to ACF/VTAM resource control, to take over resources (communications controllers or physical units) that were formerly controlled by a data communication access method in another domain, or to assume control of resources that were controlled by this domain but released. Contrast with *release*. See also *resource takeover.*

active. Pertaining to a major node that has been made known to ACF/VTAM by operator command and is available for use or pertaining to a minor node that is connected to, or available for connection to, an ACF/VTAM application program. Contrast with *inactive.*

adjacent domain. A domain that is physically connected to another domain by a single cross-domain link or by a shared local communications controller.

adjacent node. A node that is physically connected to another node by a single data link.

Advanced Communications Function (ACF). A group of program products for users of DOS/VS and OS/VS that can provide improved single-domain and, optionally, multidomain data communication capability.

Advanced Communications Function for the Virtual Telecommunications Access Method (ACF/VTAM). A program product that provides improved single-domain data communication capability and, optionally, multidomain capability.

any-mode. In ACF/VTAM: (1) The form of a read or receive request that obtains data from one unspecified terminal. (2) The form of solicit request that solicits data from all eligible connected terminals. (3) The form of connection request that connects one unspecified terminal that has logged on. (4) Contrast with *specific-mode.* See also *continue-any mode.*

application layer. In SNA, the functional layer of each individual session in which the end user's application program is executed. See also *function management, transmission subsystem.*

application program identification. The symbolic name by which an application program is identified to ACF/VTAM. It is specified in the APPLID parameter of the ACTS macro instruction. It corresponds to the ACBNAME parameter in the APPL statement or, if the ACBNAME is defaulted, to the name of the APPL statement.

application program major node. In ACF/VTAM, a member (OS/VS) or book (DOS/VS) of the ACF/VTAM definition library that contains one or more APPL statements, each representing an application program.

APPLID routine. Synonym for *logon-interpret routine.*

asynchronous operation. In ACF/VTAM, an operation such as connection or data transfer in which the application program is allowed to continue execution while ACF/VTAM performs the operation. ACF/VTAM interrupts the program as soon as the operation is completed.

asynchronous request. In ACF/VTAM, a request for an asynchronous operation.

authorization exit routine. In ACF/VTAM, an optional, user-written routine that approves or disapproves requests for connection and disconnection.

authorized path. In ACF/VTAM for OS/VS2 MVS, a facility that enables an authorized application program to specify that a data transfer or related operation be carried out in a faster manner than usual.

automatic logon. A process by which ACF/VTAM creates a logon for a terminal or logical unit to a designated application program whenever the terminal or logical unit is not connected to another program. Specifications for the automatic logon can be made when the terminal or logical unit is defined or can be made by the network operator in the VARY LOGON command. See also *controlling application program.*

available. In ACF/VTAM: (1) Pertaining to a terminal that supports only one session, is active, is not connected to an application program, and for which there is no pending logon. (2) Pertaining to an exit routine that has been specified by an application program and that is not being executed.

B

basic information unit (BIU). In SNA, the unit of data and control information that is passed between connection point managers. It consists of a request/response header (RH) followed by a request/response unit (RU).

basic mode. In ACF/VTAM, a mode of data transfer in which the application program can communicate with non-SNA terminals. Contrast with *record mode.*

basic transmission unit (BTU). (1) In the network control program, the unit of exchange between the host processor and the communications controller. It consists of control information and may also include data. The control information consists of a basic transmission header (BTH) and a basic device unit (BDU). All data transferred between the host and the communications controller is preceded by a BTU. (2) In SNA, the unit of data and control information passed between path control components. The BTU can consist of one or more path information units (PIUs), depending on whether blocking is done by the path control that builds the BTU.

block. In the basic mode of ACF/VTAM, a unit of data that is transmitted between an ACF/VTAM application program and a terminal.

boundary function. In SNA: (1) A general term used for any one of several capabilities of a host node or a communications controller node: (a) transforming the network address form to a local address form, and vice versa, for attached terminals or cluster controller nodes; (b) performing physical unit services and sequence numbering for attached, low-function terminals within its subarea; and (c) providing pacing of the data flows for secondary LUs within a subarea. (2) The programming component and functional structure that performs the above capabilities.

bracket. In ACF/VTAM, an uninterruptible unit of work, consisting of one or more chains of request units and their responses, exchanged between an application program and a terminal. Examples are data base inquiries/replies, update transactions, remote job entry output sequences to work stations, and similar applications.

bracket protocol. In SNA, a data flow control protocol in which exchanges between logical units (LUs) are achieved through the use of brackets, with one logical unit designated at session initiation as the first speaker, and the other logical unit as the bidder. The bracket protocol involves bracket initiation and termination rules.

C

cancel closedown. A closedown in which ACF/VTAM is abnormally terminated as the result of an operator command.

CDRSC. Cross-domain resource.

CDRM. Cross-domain resource manager.

change-direction protocol. In ACF/VTAM, a method of communication in which the sender stops sending on its own initiative, having signaled this fact to the receiver on the last request sent, and prepares to receive.

character-coded. In ACF/VTAM, pertaining to a logon or logoff command usually entered by a terminal operator from a keyboard and sent by a logical unit in character (unformatted) form. Contrast with *field-formatted.*

checkpoint. A point in time at which the status of a data communication system is recorded. The system can then be reconstructed to its status at or near the time of failure.

CID. Communication identifier.

closedown. The deactivation of a device, program, or system. See also *cancel closedown, orderly closedown,* and *quick closedown.*

cluster controller. See *cluster control unit* and *SDLC cluster controller.*

cluster control unit. A device that can control the input/output operations of more than one device. A remote cluster control unit is attached to a host computer only through a communications controller. A local cluster control unit is attached through a channel. A cluster control unit may be controlled by a program stored and executed in the unit; for example, the IBM 3601 Finance Communication Controller. Or it may be controlled entirely by hardware; for example, the IBM 2972 Station Control Unit. See also *communications controller* and *SDLC cluster controller.*

command. (1) A request from a terminal for the performance of an operation or the execution of a particular program. (2) In SNA, a request unit initiating an action or beginning a protocol; it is used in contrast with reply, which is a request unit (not a response) that is sent in reaction to a command. For example: Quiesce (a data flow control request), is a command, while Quiesce Complete is the reply. (3) In SNA, a data flow control or session control request that may be sent or received by an application program using record mode.

common network. In SNA, the network consisting of path control and data link control elements that routes and moves path information units between any two transmission control elements.

communication control character. *A control character intended to control or facilitate transmission of data over communication networks.

communication control unit. A communication device that controls the transmission of data over lines in a telecommunication network. Communication control units include transmission control units and communications controllers.

communication identifier (CID). In ACF/VTAM, a key for locating the control blocks that represent a session. The key is created during the session establishment procedure and deleted when the session ends.

communication line. Any physical link, such as a wire or a telephone circuit, that connects one or more remote terminals to a communication control unit, or connects one communication control unit with another.

communications controller. A type of communication control unit whose operations are controlled by a program stored and executed in the unit. Examples are the IBM 3704 and 3705 Communications Controllers.

configuration restart. In ACF/VTAM, the facility for immediate recovery after a failure in the NCP or communications controller or after a loss of contact with a physical unit or logical unit, or for delayed recovery after a failure or deactivation of a major node, ACF/VTAM, or the host computer. Recovery may include reloading the NCP or restoring the network by means of a checkpoint. Restarting by means of a checkpoint requires the user to specify one or more VSAM data sets in which ACF/VTAM keeps a record of changes to initial configuration data.

connection. (1) In ACF/VTAM, the linking of control blocks in such a way that an application program is in session with a terminal. Connection includes establishing and preparing the network path between the program and the terminal. (2) A physical capability of communicating between two end points. Also called *physical connection*. See also *queued for connection.*

connection point manager (CP manager). In SNA, one of the three components of transmission control; it provides a common mechanism by which session control, network control, and network addressable units communicate with their corresponding elements through the common network. The unit of information handled by the connection point manager is a request/response unit (RU). The unit of control information built by the sending connection point manager and interpreted by the receiving connection point manager is a request/response header (RH). See also *session control, network control.*

continue-any mode. In ACF/VTAM, a state into which a terminal is placed that allows its input to satisfy an input request issued in any-mode. While this state exists, input from the terminal can also satisfy input requests issued in specific-mode. Contrast with *continue-specific mode.*

continue-specific mode. In ACF/VTAM, a state into which a terminal is placed that allows its input to satisfy only input requests issued in specific-mode.

controlling application program. An application program to which a terminal (other than a secondary application program) is automatically logged on whenever the terminal is active and available. See also *automatic logon.*

conversational write operation. In the basic mode of ACF/VTAM, an operation wherein data is first sent to a terminal and data is then read from that terminal.

converted command. An intermediate form of a character-coded logon or logoff command produced by ACF/VTAM through use of an unformatted system services definition table. The format of a converted logon or logoff command is fixed; the unformatted system services definition table must be constructed so that the character-coded command (as entered by a *logical unit*) is converted into the predefined, converted command format. See also *character-coded.*

cross-domain. Pertaining to control or resources involving more than one domain.

cross-domain link. A data communication line physically connecting two domains. See also *local-to-local link.*

cross-domain resource (CDRSC). A resource owned by another domain but known in this domain by name and associated cross-domain resource manager.

cross-domain resource manager (CDRM). The portion of the system services control point (SSCP) that controls cross-domain sessions.

cross-domain session. A session between network addressable units in different domains.

D

data communication. The transmission, reception, and validation of data.

data flow. In SNA, any of four flows in a given session, characterized as either primary-to-secondary or secondary-to-primary, each of which may be normal or expedited.

data flow control. In SNA, a set of protocols and control functions used by the network addressable unit to assist in controlling the flow of requests and responses within a session. Contrast with *session control.*

data flow control protocol. In SNA, the sequencing rules for requests and responses by which network addressable units in a communication network coordinate and control data transfer and other operations. For example, see *bracket protocol.*

data link. (1) (SC1) An assembly of those parts of two data terminal equipments that define the protocol together with their interconnecting data circuit. This assembly enables a data source to transfer data to a data sink. (2) The communication channel, modem, and communication controls of all stations connected to the communication channel, used in the transmission of information between two or more stations. (3) The physical connection and the connection protocols between the host and communication controller nodes via the host data channel. (4) Contrast with *communication line.*

Note: *A communication line is the physical medium; for example, a telephone wire, a microwave beam. A data link includes the physical medium of transmission, the protocol, and associated communication devices and programs–it is both logical and physical.*

data link control (DLC). (1) The noninformation exchanges that set up, control, check, and terminate the information exchange(s) between two stations on a data link. (2) In SNA, one of the constituent parts of the transmission subsystem, and one of two constituent parts of the common network. It initiates, controls, checks, and terminates the data transfer over a data link between two nodes. Two distinct DLCs are defined in SNA: the DLC for the System/370 data channel, and SDLC for serial-by-bit data links. See also *path control* and *transmission control.*

data link control protocol. A set of rules used by two nodes on a data link to accomplish an orderly exchange of information. Synonymous with *line discipline.*

data transfer. In data communication, the sending of data from one point in a communication network and the receiving of the data at another point in the network.

data transmission. The sending of data from one point in a communication network for reception elsewhere.

definite response. In SNA, a form of response requested in the request header for a request unit; the receiver is requested to return a response whether positive or negative. Contrast with *exception response* and *no response.*

definition statement. In ACF/VTAM, the means of describing an element of the communication network.

device control character. (ISO) A control character used for the control of ancillary devices associated with a data processing system or data communication system, for example, for switching such devices on or off.

device-type logical unit. A logical unit residing in a physical unit other than a host computer.

disconnection. (1) In ACF/VTAM, the dissociation of control blocks in such a way as to end a session between an application program and a connected terminal. The disconnection process includes suspending the use of the network path between the program and the terminal. (2) A physical dissociation between two end points.

DLC. Data link control.

domain. In a data communication system, the portion of the total network that is controlled by the SSCP in one telecommunication access method.

E

emulation mode. A function of the network control program that enables a 3704 or 3705 Communications Controller to perform activities equivalent to those performed by an IBM 2701 Data Adapter Unit or an IBM 2702 or 2703 Transmission Control Unit. See also *network control mode.*

end user. The ultimate source or destination of information flowing through a system. An end user may be an application program, an operator (such as a terminal user or a network operator/administrator), or a data medium (such as cards or tapes).

error lock. In the basic mode of ACF/VTAM, a condition in which communication with a non-SNA terminal is suspended until a reset operation occurs.

exception message. See *exception request.*

exception request. In communicating with a logical unit, a message that indicates an unusual condition such as a sequence number being skipped. When ACF/VTAM detects such a condition, it notifies the application program. ACF/VTAM or the application program provides sense information which is included in the response that is sent to the logical unit.

exception response. (1) In SNA, a response requested in the RH for a request unit; the receiver is requested to return a response only if it is negative. Contrast with *definite response.* (2) Synonym for *negative response.*

exit list (EXLST). In VSAM or ACF/VTAM, a control block that contains the addresses of user-written routines that receive control when specified events occur during execution; for example, routines that process logons or I/O errors.

exit routine. In ACF/VTAM, any of several types of special-purpose user-written routines. See *accounting exit routine, authorization exit routine, EXLST exit routine, logon-interpret routine,* and *RPL exit routine.*

EXLST exit routine. In ACF/VTAM, a type of user-written routine whose address has been placed in an exit list (EXLST) control block. See also *RPL exit routine.*

expedited flow. In SNA, a data flow that is independent of and controls the normal flow. Data flow is split into normal and expedited flows. Requests and responses on a given flow (normal or expedited) are usually processed sequentially within the path, but the expedited flow traffic may be moved ahead of the normal flow traffic within the path. Contrast with *normal flow.*

external domain. A domain controlled by a different system services control point (SSCP).

F

FID. Format identification. FID0, FID1, FID2, FID3. See *format identification.*

field-formatted. In ACF/VTAM, pertaining to a logon or logoff command that is encoded into fields, each having a specified format such as binary codes, bit-significant flags, and symbolic names. Contrast with *character-coded.*

format identification (FID) field. In SNA, a field in a transmission header (TH) that defines the subsequent format of the header and the type of TH fields involved with a transmission. FID0 (for pre-SNA product support) and FID1 are the header formats used between host and communications controller nodes and between two communications controller nodes. FID2 and FID3 are the header formats used between communications controller nodes with boundary function and cluster controller and terminal nodes.

formatted system services (FSS). A portion of ACF/VTAM that provides certain system services as a result of receiving a field-formatted command, such as an Initiate or Terminate command. Contrast with *unformatted system services (USS).* See also *field-formatted.*

function management (FM). (1) In SNA, the layer of functional capability between the application layer and the transmission subsystem. It includes data flow control and function management data (FMD) services. See also *application layer, transmission subsystem.* (2) In ACF/VTAM, the insertion of control information within messages so that the messages sent to a particular type of terminal are in the required format and so that messages received from that type of terminal are handled properly.

function management data (FMD) services. In SNA, the component of function management responsible for request/response units marked as "FM data." This includes presentation services and logical unit services (within the logical unit), physical unit services (within the physical unit), and network services (within the system services control point). Contrast with *data flow control.*

H

host computer. (1) The primary or controlling computer in a multiple computer operation. (2) A computer used to prepare programs for use on another computer or on another data processing system; for example, a computer used to compile, link-edit, or test programs to be used on another system. (3) In a data processing system that includes ACF/VTAM or ACF/TCAM, the computer in which ACF/VTAM or ACF/TCAM resides.

host system. (1) A data processing system that is used to prepare programs and the operating environments for use on another computer or controller. (2) The data-processing system to which a communication system is connected and with which the system can communicate.

I

inactive. In ACF/VTAM, pertaining to a major node that has not been made known to ACF/VTAM and is unavailable for use, or pertaining to a minor node that is not connected to nor available for connection to an application program. Contrast with *active.*

intermediate node. In SNA, a physical unit that is capable of routing path information units to another subarea.

interpret table. In ACF/VTAM, a user-defined correlation list that translates an argument into a string of eight characters. Interpret tables can be used to translate logon data into the name of an application program for which the logon is intended.

L

LDO. Logical device order.

leading graphics. From one to seven graphic characters that may accompany an acknowledgment sent to or from a BSC terminal in response to the receipt of a block of data.

line. See *communication line.*

line control. The scheme of operating procedures and control signals by which a communication network is controlled.

line discipline. Synonym for *data link control protocol.*

line group. A collection of one or more communication lines of the same type.

local. (1) Pertaining to the attachment of devices directly by I/O channels to a host computer. Contrast with *remote.* (2) In data communication, pertaining to devices that are attached to a controlling unit by cables, rather than by data links.

local address. In SNA, an address transformed to or from a network address by the boundary function (for example, in a communications controller node) for use by a cluster controller node or terminal node. See also *network address, boundary function.*

local NCP. An NCP that is channel-attached to a host computer. Contrast with *remote NCP.*

local non-SNA major node. In ACF/VTAM, a major node whose minor nodes are locally attached non-SNA terminals.

local SNA major node. In ACF/VTAM, a major node whose minor nodes are locally attached physical and logical units.

local-to-local link. A data communication link between two local communications controllers. The link can be either a cross-domain link (communications controllers in different domains) or it can exist within a domain between local communications controllers controlled by the same system services control point.

local 3270 major node. See *local non-SNA major node.*

logical device order (LDO). In ACF/VTAM, a set of parameters that specify a data-transfer or data-control operation to local non-SNA 3270 Information Display Systems and certain kinds of start/stop or BSC terminals.

logical error. In ACF/VTAM, an error condition that results from an invalid request; a program logic error.

logical unit. In SNA, one of three types of network addressable units (NAUs). It is the port through which an end user accesses function management in order to communicate with another end user. It is also the port through which the end user accesses the services provided by the system services control point (SSCP). It must be capable of supporting at least two sessions – one with the SSCP, and one with another logical unit. It may be capable of supporting many sessions with other logical units. ACF/VTAM application programs must communicate with logical units in record mode. See also *physical unit, system services control point.*

log off. In ACF/VTAM, to request that a terminal be disconnected from an application program.

logoff. In ACF/VTAM, a request that a terminal be disconnected from an application program.

log on. In ACF/VTAM, to request that a terminal be connected to an application program.

logon. In ACF/VTAM, a request that a terminal be connected to an application program. See also *automatic logon* and *simulated logon.*

logon data. In ACF/VTAM: (1) The data portion of a field-formatted or character-coded logon from an SNA terminal or from a non-SNA 3270 terminal for which PU=YES has been specified. (2) The entire logon sequence or message from a non-SNA terminal.

logon-interpret routine. In ACF/VTAM, a user-written exit routine associated with a logon-interpret table entry that translates logon data. It may also verify the logon. Synonymous with *APPLID routine.*

logon message. Synonym for *logon data.*

logon mode. In ACF/VTAM, the communication protocols that govern a session between a logical unit and an ACF/VTAM application program or between two application programs. Synonymous with *session parameters.*

logon mode name. In ACF/VTAM, the symbolic representation of a logon mode.

logon mode table. In ACF/VTAM, a set of macro-generated constants making up one or more logon modes. Each logon mode is associated with a logon mode name.

LU-LU session. In SNA, a session between two logical units in the network. It allows communication between two end users, each associated with one of the logical units.

M

major node. In ACF/VTAM, a set of minor nodes that is filed as a member or book of a definition data set and that can be activated and deactivated as a group. See also *minor node.*

message. (1) *An arbitrary amount of information whose beginning and end are defined or implied. (2) For BSC devices, the data unit from the beginning of a transmission to the first ETX character, or between two ETX characters. For start/stop devices "message" and "transmission" have the same meaning. (3) (SC1) A sequence of characters used to convey data. The sequence usually consists of three parts: the heading, the text, and one or more characters used for control or error-detection purposes. (4) A combination of characters and symbols transmitted from one point to another. (5) In SNA, a request/response header and its associated request/response unit. In some ACF/VTAM publications, a distinction is made between messages, responses, and commands, where "message" is used to mean a data request.

minor node. In ACF/VTAM, a uniquely-defined resource within a major node that can be activated or deactivated by the VARY command. Synonymous with *specific node.* See also *major node.*

MTA. Multiple terminal access.

multiple-channel-attached communications controller. A communications controller that can be channel-attached to more than one host computer.

multiple terminal access (MTA). A feature of the network control program that permits it to communicate with a variety of dissimilar, commonly used start-stop terminals over the same switched network connection.

Multisystem Networking Facility. In ACF/VTAM, a feature that supports communication among multiple host computers operating with DOS/VS, OS/VS1, and OS/VS2 (SVS and MVS).

multithread application program. An ACF/VTAM application program that processes many requests from many terminals concurrently. Contrast with *single-thread application program.*

N

NAU. Network addressable unit.

NCP. Network control program.

NCP major node. In ACF/VTAM, a major node defined through NCP generation.

negative response. A response indicating that a request did not arrive successfully or was not processed successfully by the receiver in a session. Synonymous with *exception response.* Contrast with *positive response.*

negative response to polling limit. For a start-stop or BSC terminal, the maximum number of consecutive negative responses to polling that the communications controller accepts before suspending polling operations.

network. (1) (SC1) The assembly of equipment through which physical connections are made between terminal installations. (2) In data communication, a configuration in which two or more locations are physically connected for the purpose of exchanging data.

network address. In SNA, the address, consisting of subarea and element subfields, that uniquely identifies a link or the location of a network addressable unit. The conversion from a local address to a network address, or vice versa, is accomplished as part of the boundary function in the node attached to a cluster controller node or a terminal node. See *local address.* See also *network name.*

network addressable unit (NAU). In SNA, a logical unit, a physical unit, or a system services control point. It is the origin or the destination of information transmitted in the transmission subsystem. Each NAU has a network address that represents it to the transmission subsystem. The transmission subsystem and the NAUs collectively constitute the communication system. See also *network name, network address.*

network control (NC). In SNA, a transmission control component that permits logically adjacent connection point managers to communicate through the common network, using sessions established for other purposes and thereby avoiding special session establishment. See also *connection point manager, session control.*

network control mode. The functions of a network control program that enable it to direct a communications controller to perform activities such as polling, device addressing, dialing, and answering. See also *emulation mode.*

network control program (NCP). A program, generated by the user from a library of IBM-supplied modules, that controls the operation of a communications controller.

network control program generation. The process, performed in a host system, of assembling and link-editing a macro instruction program to produce a network control program.

network definition. In ACF/VTAM, the process of defining the identities and characteristics of each node in the network and the arrangement of the nodes. Network definition is part of ACF/VTAM definition.

network name. In SNA, the symbolic identifier by which a network addressable unit or a data link is referred to by end users. See also *network address.* In ACF/VTAM, for multidomain users, the name of the APPL statement is the network name and must be unique across domains. Contrast with *ACB name.*

network operator. (1) A person responsible for controlling the operation of a communication network. (1) An ACF/VTAM application program authorized to issue network operator commands.

network operator command. A command used to monitor or control the communication network.

network operator console. A system console or terminal in the network from which a network operator controls a communication network.

network operator logon. A logon requested on behalf of a terminal by means of a network operator command.

NIB. Node initialization block.

NIB list. A series of contiguous node initialization blocks.

no response. In SNA, an indication in the RH for a request unit that no response is to be returned to the request, whether or not it is received and processed successfully. Contrast with *definite response* and *exception response.*

node. (1) An addressable point in a data communication network. (2) In ACF/VTAM, a point in a communication network defined by a symbolic name. See also *major node* and *minor node.*

node initialization block (NIB). In ACF/VTAM, a control block associated with a particular terminal that contains information used by the application program to identify the terminal and indicate how communication requests directed at the terminal are to be processed.

node name. In ACF/VTAM, the symbolic name assigned to a specific major or minor node during network definition.

non-SNA terminal. A terminal supported by ACF/VTAM that uses start-stop or BSC protocol or that is part of a local non-SNA 3270 Information Display System.

normal flow. In SNA, a data flow that is used for most requests and responses. Data flow is split into normal and expedited flows. The expedited flow is independent of and used to control the normal flow. Requests and responses on a given flow (normal or expedited) are usually processed sequentially within the path, but the expedited flow traffic may be moved ahead of the normal flow traffic within the path. Contrast with *expedited flow.*

O

orderly closedown. The orderly deactivation of ACF/VTAM and the communication network. An orderly closedown does not take effect until all application programs have been disconnected from ACF/VTAM. Until then, all data transfer operations continue. Contrast with *cancel closedown* and *quick closedown.*

P

pacing. In data communication, a technique by which a receiving connection point manager or boundary controls the rate of transmission of a sending function connection point manager to prevent overrun.

partitioned emulation programming (PEP). A feature of the network control program, versions 2 and later, that allows a local 3704 or 3705 controller to operate as an IBM 2701, 2702, or 2703 control unit (or any combination of the three) for certain data links, while performing network control functions for other links in the communication network.

path. (1) In ACF/VTAM, the intervening nodes and data links connecting a terminal and an application program in the host computer. (2) In defining a switched SNA major node, a potential dial-out port that can be used to reach a physical unit. (3) In defining ACF/VTAM or ACF/NCP routing tables, a route through an adjacent subarea to one or more destination subareas. (4) In SNA, the series of nodes, data links, and common network components (path control and data link control) that form the complete route traversed by the information exchanged between two network addressable units in session.

path control (PC). In SNA, one of the components of the transmission subsystem, and one of two components of the common network. It is responsible for managing the sharing of data link resources of the common network and for routing basic information units (BIUs) through it. It is aware of the location of NAUs in the network and of the paths between them. It maps the BIUs, handled by transmission control, into path information units (PIUs), and then into basic transmission units (BTUs) that are passed between path control and data link control. The unit of control information built by the sending path control component and interpreted by the receiving path control component is a transmission header (TH). See also *data link control, transmission control.*

path information unit (PIU). In SNA, the unit of transmission consisting of a transmission header (TH) and either a basic information unit (BIU) or a BIU segment.

path table. A table, contained in a network node, whose entries contain path information. For example, the ACF/VTAM table constructed from PATH statements, that lists the communications controllers adjacent to the host computer (called *adjacent subareas*) that are used for cross-domain communication and indicates for each controller the subareas in other domains (called *destination subareas*) for which messages are to be routed through that controller.

peer NCP. An NCP that is attached to another NCP through a local-to-local link. Contrast with *remote NCP.* See also *local-to-local link.*

PEP. Partitioned emulation programming.

physical unit. (1) The control unit or cluster controller of an SNA terminal. (2) The part of the control unit or cluster controller that fulfills the role of a physical unit as defined by systems network architecture.

PIU. Path information unit.

positive response. A response that indicates a request was received and processed successfully. Contrast with *negative response.*

primary application program. In a session, an application program that adheres to predefined primary protocols. Contrast with *secondary application program.*

primary end of a session. A network addressable unit (for example, a primary application program) that adheres to predefined primary protocols.

program operator. An ACF/VTAM application program that is authorized to issue network operator commands and receive ACF/VTAM network operator awareness messages. See also *solicited messages* and *unsolicited messages.*

protocol. A set of rules used by the network entities to accomplish an orderly exchange of information and control. See also *data flow control protocol.*

Q

queued for connection. In ACF/VTAM, the state of a terminal that has logged on to an application program but has not yet been accepted by that application program. See also *connection.*

quick closedown. In ACF/VTAM, a closedown in which current data-transfer operations are completed, while new connection and data-transfer requests are canceled. Contrast with *cancel closedown* and *orderly closedown.*

quiesce protocol. In ACF/VTAM, a method of communicating in one direction at a time. Either the application program or the logical unit assumes the exclusive right to send normal-flow requests, and the other node refrains from sending such requests. When the sender wants to receive, it releases the other node from its quiesced state.

R

RDT. Resource definition table.

record mode. In ACF/VTAM, a mode of data transfer in which the application program can communicate with logical units or with local non-SNA or remote 3270 Information Display Systems. Contrast with *basic mode.*

release. In ACF/VTAM resource control, to relinquish control of resources (communications controllers or physical units). See also *resource takeover.* Contrast with *acquire (2).*

remote. In ACF/VTAM, pertaining to devices that are physically connected through a communications controller.

remote NCP. An NCP that is not attached directly through a channel, but is attached through a data link to a local NCP that is channel-attached. Contrast with *local NCP* and *peer NCP.*

reply. In SNA, a request unit sent in reaction to a previously received request unit (command). See also *command (2).*

request. (1) A directive that causes a data transfer or related operation to be performed. Contrast with *response.* (2) In SNA, synonym for *request unit.*

request header. In SNA, a request/response header that indicates a request.

request parameter list (RPL). In ACF/VTAM, a control block that contains the parameters necessary for processing a request for data transfer, for connecting or disconnecting a terminal, or for some other operation.

request/response header (RH). In SNA, a control field, attached to a request/response unit (RU), that specifies the type of RU being transmitted–request or response–and contains control information associated with that RU. See also *request/response unit.*

request/response unit (RU). In SNA, the basic unit of information entering and exiting the transmission subsystem. It may contain data, acknowledgment of data, commands that control the flow of data through the network, or responses to commands.

request unit. In SNA, the request/response unit following a request header. Synonymous with *request.* See also *request/response unit.*

resource definition table (RDT). In ACF/VTAM, a table that describes for a major node the characteristics of each node available to ACF/VTAM and associates each node with an address.

resource takeover. In ACF/VTAM, the action of a network operator to transfer control of resources from one domain to another. See also *acquire (2)* and *release.*

responded output. In ACF/VTAM, a type of output request that is completed when a response is returned. Contrast with *scheduled output.*

response. (1) An answer to an inquiry. (2) The unit of information that is exchanged between ACF/VTAM or an ACF/VTAM application program and an SNA terminal to describe how a request arrived. (3) In SNA, synonym for *response unit.* (4) Contrast with *request.*

response header. In SNA, a request/response header that indicates a response.

response unit. In SNA, the request/response unit following a response header; it is sent in response to a request unit. Synonymous with *response.* See also *request/response unit.*

RH. Request/response header.

RPL. Request parameter list.

RPL-based macro instruction. In ACF/VTAM, a macro instruction whose parameters are specified by the user in a request parameter list.

RPL exit routine. In ACF/VTAM, a user-written routine whose address has been placed in the EXIT field of a request parameter list. ACF/VTAM invokes the routine to indicate that an asynchronous request has been completed. See also *EXLST exit routine.*

RU. Request/response unit.

S

scheduled output. In ACF/VTAM, a type of output request that is completed, as far as the application program is concerned, when the program's output data area is free. Contrast with *responded output.*

SDLC. Synchronous data link control.

SDLC cluster controller. A cluster control unit for a teleprocessing subsystem.

secondary application program. In a session, an application program that adheres to secondary session protocols. Contrast with *primary application program.*

secondary end of a session. A logical unit, secondary application program, or non-SNA terminal.

sequence number. A numerical identifier assigned by ACF/VTAM to each message exchanged between two nodes.

session. (1) The period of time during which a user of a terminal can communicate with an interactive system; usually, the elapsed time from when a terminal user logs on the system until the user logs off the system. (2) The period of time during which programs or devices can communicate with each other. (3) In SNA, a logical connection, established between two network addressable units (NAUs), that allows them to communicate. The session is uniquely identified by a pair of network addresses, identifying the origin and destination NAUs of any transmissions exchanged during the session. (4) In the NCP, a line-scheduling period. See *LU-LU session, SSCP-LU session, SSCP-PU session.*

session control. In SNA, one of the components of transmission control. It is responsible for allocating resources necessary for a session, for purging data flowing in a session if an unrecoverable error occurs, and for resynchronizing the data flow after such an error.

session limit. (1) In the network control program, the maximum number of concurrent line-scheduling sessions on a non-SDLC, multipoint line. (2) In SNA, the maximum number of simultaneous sessions a particular network addressable unit can support.

session parameters. Synonym for *logon mode.*

share limit. The limit of the number of SSCPs that can simultaneously share a resource.

shared. Pertaining to the availability of a resource to more than one user at the same time.

simulated logon. A logon generated for a terminal by ACF/VTAM at the primary application program's request. The primary application program accepts or rejects the terminal as if it had logged on.

single-channel-attached communications controller. A communications controller that is channel-attached to only one host computer.

single-thread application program. An ACF/VTAM application program that processes requests from terminals one at a time. Such a program usually requests synchronous operations from ACF/VTAM, waiting until each operation is completed before proceeding. Contrast with *multithread application program.*

SNA. Systems network architecture.

SNA terminal. In ACF/VTAM: (1) A physical unit, logical unit, or secondary application program. (2) A terminal that is compatible with systems network architecture.

SNBU. Switched network backup.

solicit. In ACF/VTAM, to obtain data from a BSC or start-stop terminal or from a local non-SNA 3270 terminal and move the data into ACF/VTAM buffers.

solicited message. A response from ACF/VTAM to a network operator command entered by a program operator. Contrast with **unsolicited message.**

specific-mode. In ACF/VTAM: (1) The form of read, receive, or solicit request that obtains data from one specific terminal. (2) The form of connection request that connects a specific terminal that has logged on. (3) Contrast with *any-mode.* See also *continue-specific mode.*

specific node. See *minor node.*

SSCP. System services control point.

SSCP ID. An identifying number associated with an SSCP (that must be unique in a multidomain system) that enables a device (especially a dial-in device) to identify an SSCP at a particular location and enables another SSCP to identify this SSCP when establishing a session with it.

SSCP-LU session. A session during which ACF/VTAM (the system services control point, SSCP) and a logical unit (LU) can communicate.

SSCP-PU session. A session during which ACF/VTAM (the system services control point, SSCP) and a physical unit (PU) can communicate.

start options. In ACF/VTAM, the user-specified or IBM-supplied options that determine certain conditions that are to exist during the time an ACF/VTAM system is operating. For example: the size of ACF/VTAM buffer pools, which major and minor nodes are to be traced by the ACF/VTAM trace facility, and which major nodes are to be initially active. Start options can be predefined or specified by the network operator when ACF/VTAM is started.

subarea. A group of addressable elements in the network that have the same subarea ID.

subarea ID. A subfield of network address.

switched network backup (SNBU). An optional facility that allows a user to specify, for certain types of stations, a switched line to be used as an alternate path (backup) if the primary line becomes unavailable or unusable.

switched SNA major node. In ACF/VTAM, a major node whose minor nodes are physical and logical units attached by switched SDLC links.

synchronous operation. In ACF/VTAM, a connection, communication, or other operation in which ACF/VTAM, after receiving the request for the operation, does not return control to the program until the operation is completed. Contrast with *asynchronous operation.*

synchronous request. In ACF/VTAM, a request for a synchronous operation. Contrast with *asynchronous request.*

system services control point (SSCP). In SNA, a network addressable unit that provides services via a set of command processors (network services) supporting physical units and logical units. The SSCP must be in session with each logical unit and each physical unit for which it provides services. It also provides services for the network operators or administrators who control the configuration. The SSCP is commonly located at a host node.

systems network architecture (SNA). The total description of the logical structure, formats, protocols, and operational sequences for transmitting information units through the communication system. Communication system functions are separated into three discrete areas: the application layer, the function management layer, and the transmission subsystem layer. The structure of SNA allows the ultimate origins and destinations of information—that is, the end users—to be independent of, and unaffected by, the specific communication-system services and facilities used for information exchange.

T

TC. Transmission control.

teleprocessing subsystem. In ACF/VTAM, a secondary or subordinate network and set of programs that are part of a larger teleprocessing system; for example, the combination consisting of an SDLC cluster controller, its stored programs, and its attached terminals.

teleprocessing system. A data processing system in combination with data communication facilities.

terminal. (1) A device, usually equipped with a keyboard and some kind of display, capable of sending and receiving information over a communication channel. (2) In ACF/VTAM, the secondary end of a session; that is, a logical unit, a start-stop or BSC device, a local non-SNA 3270 device, or an application program.

terminal component. A separately addressable part of a terminal that performs an input or output function, such as the display component of a keyboard-display device or a printer component of a keyboard-printer device.

terminal system. In a data communication network, a terminal control unit or a cluster control unit and its attached devices. Synonymous with *teleprocessing subsystem.*

TH. Transmission header.

transit node. An intermediate node that is capable of routing path information units to another domain.

transmission. In data communication, one or more blocks or messages. For BSC and start-stop devices, a transmission is terminated by an EOT character. See also *block* and *message.*

transmission control (TC). In SNA, one of three components of the transmission subsystem. It has three subcomponents: the connection point manager, session control, and network control. It establishes, controls, and terminates sessions, and also controls the flow of information into and out of the common network for a session between network addressable units. It provides access to the transmission subsystem; this direct access is used by function management components. A transmission control element exists for each active session. See also *data link control, path control.*

transmission header (TH). In SNA, a control field attached to a basic information unit (BIU) or to a BIU segment, and used by path control. It is created by the sending path control component and interpreted by the receiving path control component. See also *path information unit.*

transmission subsystem. In SNA, the innermost layer of the communication system. It provides the control in each session to route and move data units between NAUs, and to manage the NAUs and their interconnecting paths. Its three constituent parts are data link control, path control, and transmission control. See also *application layer function management.*

U

unformatted system services (USS). A portion of ACF/VTAM that translates a character-coded command, such as a logon or logoff command, into a field-formatted command for processing by formatted system services (FSS). Constrast with *formatted system services (FSS)* and *character-coded.*

unsolicited message. A network operator message, from ACF/VTAM to a program operator, that is unrelated to any command entered by the program operator. Contrast with *solicited message.*

user logon data. Synonymous with *logon data (1).*

Index

Where more than one page reference is given, the major reference is first.

Advanced Communications
Function for VTAM
(ACF/VTAM)
Macro Language Guide

SC38-0256-0

READER'S COMMENT FORM

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. This form may be used to communicate your views about this publication. They will be sent to the author's department for whatever review and action, if any, is deemed appropriate.
IBM shall have the nonexclusive right, in its discretion, to use and distribute all submitted information, in any form, for any and all purposes, without obligation of any kind to the submitter. Your interest is appreciated.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

How did you use this publication?

[ ] As an introduction   [ ] As a text (student)

[ ] As a reference manual   [ ] As a text (instructor)

[ ] For another purpose (explain) ______________________________

Is there anything you especially like or dislike about the organization, presentation, or writing in this manual? Helpful comments include general usefulness of the book; possible additions, deletions, and clarifications; specific errors and omissions.

Page Number: Comment:

What is your occupation? ______________________________

Newsletter number of latest Technical Newsletter (if any) concerning this publication: ______________

If you wish a reply, give your name and address:

IBM branch office serving you ______________________________

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments.)

Cut or Fold Along Line

SC38-0256-0

Reader's Comment Form

Fold

Fold

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

Postage will be paid by:

International Business Machines Corporation
Department 63T
Neighborhood Road
Kingston, New York 12401

Fold

Fold

IBM®

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, N.Y. 10604

IBM World Trade Americas/Far East Corporation
Town of Mount Pleasant, Route 9, North Tarrytown, N.Y., U.S.A. 10591

IBM World Trade Europe/Middle East/Africa Corporation
360 Hamilton Avenue, White Plains, N.Y., U.S.A. 10601

Cut or Fold Along Line

ACF/VTAM Macro Language Guide (File No. S370-30) Printed in U.S.A. SC38-0256-0

SC38-0256-0

ACF/VTAM Macro Language Guide (File No. S370-30) Printed in U.S.A. SC38-0256-0

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, N.Y. 10604

IBM World Trade Americas/Far East Corporation
Town of Mount Pleasant, Route 9, North Tarrytown, N.Y., U.S.A. 10591

IBM World Trade Europe/Middle East/Africa Corporation
360 Hamilton Avenue, White Plains, N.Y., U.S.A. 10601